TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY
Julkaisu 637 • Publication 637

Panu Hämäläinen

Cryptographic Security Designs and Hardware Architectures for Wireless Local Area Networks

Tampere 2006

Tampereen teknillinen yliopisto. Julkaisu 637
Tampere University of Technology. Publication 637

Panu Hämäläinen

Cryptographic Security Designs and Hardware Architectures for Wireless Local Area Networks

Thesis for the degree of Doctor of Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB111, at Tampere University of Technology, on the 8th of December 2006, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2006

ISBN 952-15-1685-2 (printed)
ISBN 952-15-1736-0 (PDF)
ISSN 1459-2045

ABSTRACT

Wireless Local Area Networks (WLAN) have developed to widely utilized technologies for short-range telecommunications. While the technologies enable various new services, the wireless environment and the constraints of WLAN devices set new requirements for network security and its realization. In addition to a security specification, a security processing implementation has a key role in protecting WLANs. This Thesis presents designs and implementations for protecting WLANs using cryptographic mechanisms.

The focus of the Thesis is on the security of the standard WLAN technologies which have recently been driving the markets and the research work. The technologies, their problems, and proposed improvements are surveyed. A design specifically addressing the vulnerabilities of Bluetooth is presented. Furthermore, designs for protecting stored data and maintaining time synchronization in WLANs are developed. The generally accepted security design practices and the constraints of the WLAN devices are respected throughout the presented designs.

Cryptographic software implementations cannot often provide security with high performance and usability while meeting the restrictions of WLAN devices. Therefore, the Thesis presents cryptographic hardware architectures that can efficiently be used for securing WLANs. The architectures support the cryptographic mechanisms of the standard WLANs as well as the mechanisms proposed in the Thesis. Several solutions providing different trade-offs between performance and resource consumption are developed for Advanced Encryption Standard (AES), Triple Data Encryption Standard (3DES), and RC4 as well as for the modular exponentiation of public-key schemes. Related implementations are surveyed and compared.

As an example of a full, security-oriented application, the separate components are integrated into a novel wireless Real-Time Betting (RTB) application. It utilizes the security designs and implementations on the wireless data link layer as well as on the application layer in order to support efficient embedded terminal implementations. The RTB application is especially seen well-suited for providing local services through WLANs.

PREFACE

The research work for this Thesis was carried out in the Institute of Digital and Computer Systems in Tampere University of Technology during the years 2000–2006. Ironically, a large part of the Thesis was written in an Internet cafe in Rome using a WLAN connection which had security turned off.

I express my greatest gratitude to my supervisor Docent Dr. *Marko Hännikäinen* for his professional guidance and comments during the research. I am also most indebted to Prof. *Timo D. Hämäläinen* for his guidance and providing me the opportunity to carry out the research in the DACI research group. I am grateful to both for that I could always concentrate on the research itself and did not have to worry about financial aspects. Sincere acknowledgments go to Prof. *Juha Röning* and Prof. *Jorma Skyttä* for reviewing and helping me to improve the manuscript of the Thesis.

Many thanks to all my colleagues for their assistance and for creating enjoyable working atmosphere. Special thanks to Mr. *Mauri Kuorilehto*, M.Sc., Mr. *Timo Alho*, M.Sc., Mr. *Jari Heikkinen*, M.Sc., Mr. *Erno Salminen*, M.Sc., Dr. *Tero Kangas*, and Dr. *Tuomas Järvinen* for co-operation and fruitful discussions. Mr. *Peter Groen*, M.Sc., Mr. *Ning Liu*, M.Sc., and Mr. *Risto Sterling*, M.Sc., also deserve thanks for their valuable work for achieving results presented in the Thesis.

The research was financially supported by Graduate School in Electronics, Telecommunications and Automation (GETA), Finnish Funding Agency for Technology and Innovation (Tekes), Nokia Foundation, Tekniikan edistämissäätiö (TES), Heikki ja Hilma Honkasen säätiö, Ulla Tuomisen säätiö, Kaupallisten ja teknillisten tieteiden tukisäätiö (KAUTE), and HPY:n tutkimussäätiö.

I would like to thank my father *Raimo* and brother *Juha* for their love and support. My longing thoughts go to my beloved mother *Leena-Maija* who left us all too soon. Finally, thank you *Katri* for your love and encouragement and for always being there for me.

Tampere, November 2006

Panu Hämäläinen

TABLE OF CONTENTS

LIST OF PUBLICATIONS

This Thesis consists of an introductory part and the following publications. In the introductory part the publications are referred to as [P1], [P2], ..., [P10].

[P1] P. Hämäläinen, M. Hännikäinen, M. Niemi, T. D. Hämäläinen, and J. Saarinen, "Implementation of Link Security for Wireless Local Area Networks," in *Proceedings of the 2001 IEEE International Conference on Telecommunications (ICT 2001)*, vol. 1, Bucharest, Romania, June 4–7, 2001, pp. 299–305.

[P2] P. Hämäläinen, M. Hännikäinen, and T. D. Hämäläinen, "Efficient Hardware Implementation of Security Processing for IEEE 802.15.4 Wireless Networks," in *Proceedings of the 2005 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS 2005)*, Cincinnati, Ohio, USA, Aug. 7–10, 2005, pp. 484–487.

[P3] T. Järvinen, P. Salmela, P. Hämäläinen, and J. Takala, "Efficient Byte Permutation Realizations for Compact AES Implementations," in *Proceedings of the 13th European Signal Processing Conference (EUSIPCO 2005)*, Antalya, Turkey, Sept. 4–8, 2005, 4 pages.

[P4] P. Hämäläinen, M. Hännikäinen, T. D. Hämäläinen, and J. Saarinen "Hardware Implementation of the Improved WEP and RC4 Encryption Algorithms for Wireless Terminals," in *Proceedings of the 10th European Signal Processing Conference (EUSIPCO 2000)*, vol. 4, Tampere, Finland, Sept. 5–8, 2000, pp. 2289–2292.

[P5] P. Hämäläinen, J. Heikkinen, M. Hännikäinen, and T. D. Hämäläinen, "Design of Transport Triggered Architecture Processors for Wireless Encryption," in *Proceedings of the 8th Euromicro Conference on Digital System Design – Architectures, Methods, and Tools (DSD 2005)*, Porto, Portugal, Aug. 30–Sept. 3, 2005, pp. 144–152.

[P6] P. Groen, P. Hämäläinen, B. Juurlink, and T. D. Hämäläinen, "Accelerating the Secure Remote Password Protocol Using Reconfigurable Hardware," in *Proceedings of the 2004 ACM International Conference on Computing Frontiers (CF'04)*, Ischia, Italy, Apr. 14–16, 2004, pp. 471–480.

[P7] P. Hämäläinen, N. Liu, M. Hännikäinen, and T. D. Hämäläinen, "Acceleration of Modular Exponentiation on System-on-a-Programmable-Chip," in *Proceedings of the 2005 IEEE International Symposium on System-on-Chip (SoC 2005)*, Tampere, Finland, Nov. 15–17, 2005, pp. 14–17.

[P8] T. Alho, P. Hämäläinen, M. Hännikäinen, and T. D. Hämäläinen, "Design of a Compact Modular Exponentiation Accelerator for Modern FPGA Devices," in *Proceedings of World Automation Congress 2006 (WAC 2006) – Special Session on Information Security and Hardware Implementations*, Budapest, Hungary, July 24–27, 2006, 7 pages.

[P9] P. Hämäläinen, N. Liu, R. Sterling, M. Hännikäinen, and T. D. Hämäläinen, "Design and Implementation of an Enhanced Security Layer for Bluetooth," in *Proceedings of the 8th IEEE International Conference on Telecommunications (ConTEL 2005)*, Zagreb, Croatia, June 15–17, 2005, pp. 575–582.

[P10] P. Hämäläinen, M. Hännikäinen, T. D. Hämäläinen, R. Soininen, and R. Rautee, "Design and Implementation of Real-time Betting System with Offline Terminals," *Elsevier Electronic Commerce Research and Applications*, vol. 5, no. 2, pp. 170–188, 2006.

LIST OF ABBREVIATIONS

3DES	Triple-DES
ACL	Access Control List
AES	Advanced Encryption Standard
AP	Access Point
API	Application Programming Interface
AS	Authentication Server
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction set Processor
ATM	Asynchronous Transfer Mode
BRAM	Block RAM
BRAN	Broadband Radio Access Network
CA	Certificate Authority
CBC	Cipher Block Chaining
CBC-MIC	CBC Message Integrity Code
CCM	CTR with CBC-MIC
CCMP	CCM Protocol
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CTR	CounTeR mode
DEA	Data Encryption Algorithm
DES	Data Encryption Standard

DoS	Denial-of-Service
DPRAM	Dual-Port RAM
DSL	Digital Subscriber Line
DSP	Digital Signal Processing
DVB	Digital Video Broadcasting
EAB	Embedded Array Block
EAP	Extensible Authentication Protocol
EAPOL	EAP Over LAN
ECB	Electronic CodeBook
ECC	Elliptic Curve Cryptography
ESB	Embedded System Block
ETSI	European Telecommunications Standards Institute
FMS	Fluhrer–Mantin–Shamir
FPGA	Field Programmable Gate Array
FU	Functional Unit
GF	Galois Field
GPS	Global Positioning System
GSM	Global System for Mobile communications
GTK	Group Transient Key
HCI	Host Controller Interface
HIPERLAN	HIgh PErformance Radio LAN
HIPERMAN	HIgh PErformance Radio Metropolitan Area Network
HMAC	Hashed Message Authentication Code
ICV	Integrity Check Value
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPsec	IP security
IV	Initialization Vector
KCK	Key Confirmation Key
KEK	Key Encryption Key

kgate	kilogate (10^3 gates)
L2CAP	Logical Link Control and Adaptation Protocol
LAN	Local Area Network
LE	Logic Element
LFSR	Linear Feedback Shift Register
LM	Link Manager
LMP	LM Protocol
LUT	Look-Up-Table
MAC	Medium Access Control
Mbit/s	Megabits (10^6 bits) per second
MD5	Message Digest 5
MIC	Message Integrity Code
NIST	National Institute of Standards and Technology
NRE	Non-Recurring Engineering
OCB	Offset CodeBook
OFB	Output FeedBack
OSI	Open Systems Interconnections
PAE	Port Access Entity
PDA	Personal Digital Assistant
PIN	Personal Identification Number
PKI	Public-Key Infrastructure
PLD	Programmable Logic Device
PMK	Pairwise Master Key
PNC	PicoNet Coordinator
PRNG	Pseudo-Random Number Generator
PSK	Pre-Shared Key
PTK	Pairwise Transient Key
PU	Processing Unit
QoS	Quality of Service
RADIUS	Remote Authentication Dial-In User Service

RAM	Random Access Memory
RF	Register File
RFID	Radio Frequency IDentification
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RSN	Robust Security Network
RSNA	RSN Association
RTB	Real-Time Betting
S-box	Substitution box
SFU	Special Functional Unit
SHA	Secure Hash Algorithm
SIG	Special Interest Group
SIM	Subscriber Identity Module
SKKE	Symmetric-Key Key Establishment
SoC	System-on-Chip
SRP	Secure Remote Password protocol
SSL	Secure Sockets Layer
TC	Trust Center
TCP	Transmission Control Protocol
TDEA	Triple Data Encryption Algorithm
TGi	IEEE 802.11 Task Group I
TKIP	Temporal Key Integrity Protocol
TLS	Transport Layer Security
TSN	Transition Security Network
TTA	Transport Triggered Architecture
TTLS	Tunneled TLS
TTP	Trusted Third Party
TUTWLAN	Tampere University of Technology WLAN
UMTS	Universal Mobile Telecommunications System
VHDL	VHSIC Hardware Description Language

VHSIC	Very High-Speed Integrated Circuit
VPN	Virtual Private Network
WEP	Wired Equivalent Privacy
Wi-Fi	Wireless Fidelity
WLAN	Wireless LAN
WMAN	Wireless Metropolitan Area Network
WPA	Wi-Fi Protected Access
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network
WWAN	Wireless Wide Area Network
XOR	eXclusive-OR

1. INTRODUCTION

During the recent years, wireless network technologies have achieved an important role as telecommunications media [14]. Whereas wired networks provide only fixed network topologies, wireless networks support low-cost and effortless installations, ad hoc networking, portability of network devices, and mobility of network users. The application area of wireless networks has extended from limited speech services into high-speed data transfer and multimedia along with the growth of network capacities [182]. A need for low-cost, low-rate, and very low-power technologies has emerged at the other end of the wireless technology spectrum as well [183]. Devices supporting multiple wireless technologies are appearing and envisioned to provide ubiquitous network access with a large variety of services [14, 16, 101, 106, 194, 203, P10].

Wireless communication technologies can be categorized according to their typical application fields, data rates, and coverage [105,220]. Table 1 illustrates the generally used classification that originates from the IEEE [123]. In the table, the presented values are not definitive but they are provided for perceiving the relationships of the different classes. The wireless transceiver is assumed to be a radio although other wireless physical layers, such as infrared, can be used as well.

Wireless Wide Area Networks (WWAN) and Wireless Metropolitan Area Networks

Table 1. *Classification of wireless communication technologies.*

Class	Nominal data rate	Radio coverage	Typical applications	Example technologies
WWAN	< 10 Mbit/s	> 10 km	Telephony, mobile Internet	GSM, UMTS, satellite
WMAN	< 100 Mbit/s	< 10 km	Broadband Internet	IEEE 802.16, HIPERMAN
WLAN	< 100 Mbit/s	< 100 m	Wired LAN replacement	IEEE 802.11, HIPERLAN/2
WPAN	< 10 Mbit/s	< 10 m	Personal data transfer	Bluetooth, IEEE 802.15.4
WSN	< 1 Mbit/s	< 1 km	Monitoring, control	proprietary, RFID

(WMAN) provide the widest geographical coverage. The highly utilized WWANs mainly consist of the traditional digital cellular telephone networks and their extensions [105], such as Global System for Mobile Communications (GSM) and Universal Mobile Telecommunications System (UMTS). Communication satellites belong to this class as well. WMANs are emerging technologies developed for broadband network access as an alternative to cable networks and Digital Subscriber Lines (DSL) in homes and enterprises. Examples of WMAN technologies are IEEE 802.16 [124] and HIgh PErformance Radio Metropolitan Area Network (HIPERMAN) [56].

Wireless Local Area Networks (WLAN) have recently gained a significant share of the wireless market [172, 173]. They were originally developed for extending or replacing wired computer LANs. Currently, WLANs are utilized whenever a local connection is needed only temporarily, mobility is desired, or when cabling is costly and inconvenient. WLANs are used in meetings, offices, healthcare, automation, stockpiling, and education. They are also widely employed for providing network access in public buildings and enterprises and for sharing or replacing DSL connections in homes. WLANs commonly support centralized and ad hoc topologies, depicted in Fig. 1. The prevailing WLAN technology is IEEE 802.11 [121]. Another exemplar technology is HIgh PErformance Radio LAN type 2 (HIPERLAN/2) [56], the utilization of which has mainly remained at the level of standardization.

The class closely related to WLANs consists of Wireless Personal Area Networks (WPAN), such as Bluetooth [24] and IEEE 802.15.4 [116]. WPANs are generally targeted at data communications between personal devices, including Personal Dig-

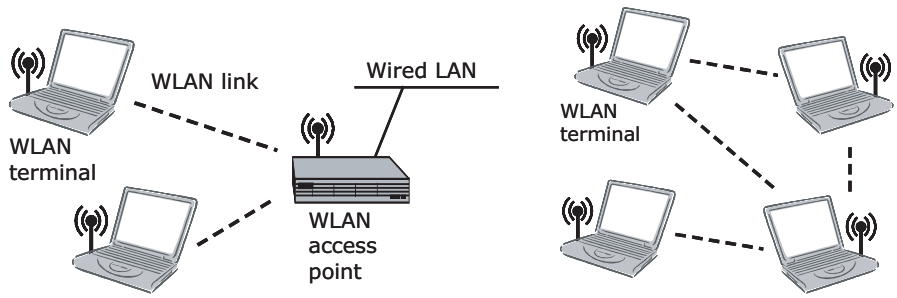


Fig. 1. *Wireless Local Area Network (WLAN) topologies. In the centralized topology (a), one device operates as a coordinator and usually acts as an Access Point (AP) providing connections to a wired infrastructure network. In the ad hoc topology (b), the devices are equal and connect directly or through multiple hops to each other for peer-to-peer communications.*

ital Assistants (PDA), mobile phones, and laptops [220]. WPANs are also used for low-rate and low-power communications e.g. in automation and alarm systems [116]. Furthermore, WPAN technologies can be used for providing wireless access to an infrastructure LAN [25] and for enabling high-speed multimedia content delivery [115]. Hence, WPANs are not clearly distinct from WLANs. The differences rather lie in the non-functional requirements, such as cost, power, and range [105].

The fifth and emerging class of wireless technologies is Wireless Sensor Networks (WSN) [152, 225]. WSNs consist of independent, collaborating, highly resource-constrained nodes or actuators that sense, process, and exchange data as well as act according to the collected data content. In contrast to WLANs and WPANs, WSNs are typically larger, self-organizing, and application-oriented. The radio coverage can vary from centimeters to hundreds of meters [152]. Currently, WSNs are mainly implemented as proprietary solutions [152]. Radio Frequency IDentification (RFID) can be placed to the class of WSNs as well.

While the various wireless technologies have enabled new types of services, they have set new requirements for the authentication of users, devices, services, and networks as well as for securing data transfer, information storages, and wireless devices themselves [12, 101, 196, 198, 199, P10]. In wired networks devices have fixed connections and information is exchanged inside a cable. On the contrary, wireless devices and networks are discoverable and transmissions available to anyone within the radio coverage. Transmissions are always broadcast by their nature. Portability exposes wireless devices to additional security threats as they can easily be captured physically [66, 103, 189, P10]. Moreover, the limited power supplies and processing capacities of low-cost wireless devices enable new forms of Denial-of-Service (DoS) attacks, such as battery draining [103]. Hence, new security designs and implementations that accommodate to the requirements and the constraints of the wireless operation environment are needed.

Standardization is required for spreading out designs, advancing commercialism, and specifically for enabling the ubiquitous wireless connectivity through interoperability. However, most standard and standard-like wireless technologies have failed in their security solutions. Even the newest and revised specifications contain shortcomings, still allow using the flawed procedures, and/or leave certain vital security components, such as authentication, unspecified. Furthermore, a new design is always a unique combination of components, which can imply new weaknesses even if the components alone were secure. The security specifications generally define the executed procedures, not how they should be used and combined with other procedures, what security guarantees they offer, or how to implement them efficiently.

The lack of implementation during the specification process often implies inconsistencies, shortcomings, and inefficient design choices. Hence, security development for wireless technologies is a continuing process.

Security procedures are generally among the tasks consuming most of the overall processing capacity in network devices [10, 158, 198, 261]. In addition, the security requirements of wireless environments have increased the amount of security procedures compared to wired networks [66]. The constantly increasing data rates and real-time requirements call for increased throughputs. Therefore, the security processing implementation has a key role in maintaining the desired level of security specifically in the embedded, wireless devices [103, 133, 198–200, 219]. Efficient implementations prevent degenerating response times and communication latencies as well as increase the operating times of battery-powered devices. The system clock rate can be decreased and/or more processing time can be provided to the other tasks of a device. The knowledge that certain algorithms and protocols can be implemented efficiently facilitates the security design process. Furthermore, efficient implementations improve usability, which prevents users from switching security off.

1.1 Objective and Scope of Research

Security can be defined as a state of defense against willful acts of smart adversaries—people. Security implicitly covers *safety* to some extend. Safety is defined as the defense against random events, such as accidents and failures. The terms security and *protection* are used interchangeably in this Thesis. Security design involves specifying a selection of procedures for providing security, such as algorithms, protocols, and their usage. As discussed above, another key aspect of security development, specifically in wireless networks, is to design an efficient implementation of the selected procedures, consisting of their components and interaction.

This Thesis considers the security of short-range wireless communication technologies. Because of their similarities, the WLAN and WPAN classes are combined into a single class and referred to as WLANs. The focus of the Thesis is on the security of WLANs provided through cryptography [171], its usage and implementation for encryption and authentication. The objective of the research has been to develop designs and implementations for efficiently realizing and maintaining cryptographic protection in WLAN devices. The term *efficient* refers to a design (specification) or an implementation which results in a good balance between cost (required processing resources), power consumption, and performance. *Performance* refers to the time re-

quired for performing the task in question. Resource and power consumption are not significantly sacrificed for improving performance. A procedure involving cryptography is considered *secure* if it has been proven secure [77] or it is generally believed to be secure against computationally bounded adversaries or beneficial attacks against the procedure are not known.

The protocol stack of the Thesis along with the Open Systems Interconnections (OSI) reference model [126] is illustrated in Fig. 2. The OSI layers directly related to WLANs are the physical layer, which is assumed to be a radio, and the data link layer. Although the data link layer often corresponds to more than one layer in WLANs, in this Thesis all these layers are regarded to belong to a single Medium Access Control (MAC) layer. The terms *(data) link layer* and *MAC layer* are used interchangeably. This Thesis concentrates on the cryptographic security of WLANs at the MAC layer. The other addressed layer is the application layer. An application-driven design is presented in [P10].

The Transmission Control Protocol/Internet Protocol (TCP/IP) suite is often utilized between the application layer and the lower protocol layers in WLAN devices. Hence, it is included in Fig. 2. In addition, WLAN MAC layers have adopted solutions originally purposed for protecting TCP/IP connections, such as [2]. However, security issues specific to the TCP/IP layers are not discussed in this Thesis. From the point of the MAC layer, all the upper protocol layers are seen as a part of the application

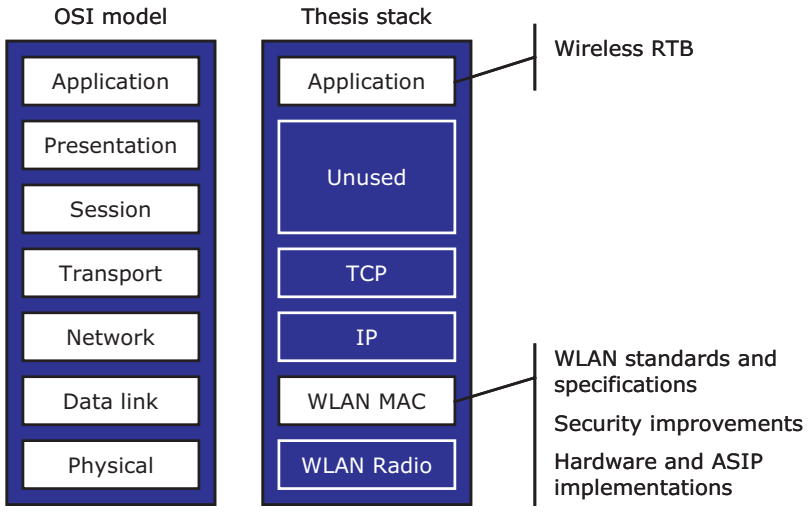


Fig. 2. *Protocol stack of the Thesis along with the Open Systems Interconnections (OSI) reference model.*

layer. Fig. 2 shows which layers the topics considered in this Thesis are related to.

The Thesis especially concentrates on the efficient implementation of cryptographic processing in WLAN devices through hardware design. The implementations are carried out in Field Programmable Gate Arrays (FPGA) and in a configurable processor architecture called Transport Triggered Architecture (TTA) [37]. TTA and its design environment are aimed at embedded Application Specific Instruction set Processor (ASIP) development.

The implementations in FPGA technologies are targeted at the different prototype generations of TUTWLAN [132, 148, 231], which is a WLAN developed in the Institute of Digital and Computer Systems at Tampere University of Technology (TUT) [105]. TUTWLAN has been developed for research purposes, not to meet any specific standard. This allows freely experimenting with different functionalities, ranging from the application layer to radio selection and including both standard and non-standard solutions. Instead of being a single, fixed WLAN technology, TUTWLAN can rather be seen as a development environment in which various solutions related to WLANs are examined. FPGA-based prototypes have enabled this wide range of experiments.

FPGA technologies are currently not considered mature for wireless end-user products because of their high power consumption [151]. Specifically, static power dissipation is significantly higher than in dedicated hardware [63, P2]. Therefore, due to the usage of FPGAs as prototyping and verification platforms, power consumption is not discussed in detail in the Thesis. The focus is on hardware resource consumption and performance. In FPGA technologies power consumption reductions can be achieved with compact implementations, particularly if they allow switching to a smaller FPGA device.

A security design for TUTWLAN has also been specified [212]. However, instead of this or other proprietary WLAN security solutions, the Thesis concentrates on various algorithms and protocols that relate to commercially significant and emerging standard WLAN technologies. The Thesis specifically addresses IEEE 802.11 [112] and its security improvements [118], Bluetooth [24], and IEEE 802.15.4 [116]. TUTWLAN is used as a prototyping and verification environment.

As one example of a full, security-oriented application, the Thesis combines the separate cryptographic components into a novel wireless Real-Time Betting (RTB) application [97, 104, P10]. It utilizes the security designs and implementations on the wireless link layer as well as on the application layer in order to support efficient embedded terminal implementations. The RTB application is especially seen well-

suited for providing local services through WLANs. The design includes features that are regarded advantageous in future WLAN devices, related to protecting stored data and maintaining reliable time synchronization.

The security of WLANs includes the security considerations of all the other computer systems. Hence, it is a broad subject even when limited to the security provided through cryptography. However, the security aspects related to mobility (e.g. handoffs, roaming, and interworking between technologies), routing, network architectures, intrusion detection, Virtual Private Networks (VPN), and firewalls are regarded as tasks of other protocol layers than the ones considered in this Thesis. Furthermore, security policies, accounting, digital rights management, anonymity, software threats (e.g. viruses and worms), and design of physical protection (e.g. tamper-resistant hardware and prevention of side channel attacks), and mechanisms for random number generation are out of the scope. Nevertheless, the designs and implementations of this Thesis can be used for addressing also these aspects, even though not explicitly discussed.

To summarize, the statement is that *the designs and implementations presented in this Thesis can be used for efficiently securing WLANs*. The security designs can be used for securing WLANs in general. The implementations ensure that the security level does not have to be decreased due to the limitations of WLAN devices. Even though the Thesis considers specific WLAN technologies, their link layer, and specific prototyping platforms, the designs and implementation methods can be generalized and applied in other environments, protocol layers, and technologies, also beyond WLANs. For example, in order to share processing resources, the encryption implementations can be utilized throughout the protocol stack of a wireless device.

1.2 Main Contributions and Outline of Thesis

The Thesis consists of an introductory part and ten publications. The introduction provides the technical background, motivates the work, and reviews related work. The publications present the main results. As a summary, the main contributions of the Thesis are:

- A review of the security designs and problems of the most significant and emerging WLAN technologies. Proposed improvements are examined as well.
- An up-to-date survey of resource-efficient hardware architectures and specialized processor architectures for the secret-key cryptography of WLANs.

- The design and implementation of dedicated hardware architectures for efficient cryptographic processing in WLANs, related to encryption and authentication. Both public-key and secret-key algorithms as well as various encryption modes are considered.
- The design and implementation of compact, reconfigurable ASIPs utilizing TTA for supporting the secret-key algorithms of WLANs.
- The design and prototype implementation of a novel security layer for Bluetooth utilizing the presented hardware implementations. The design improves security, increases performance, and decreases resource requirements compared to the standard Bluetooth design. The proposed solution enables supporting the security processing of all significant WLAN technologies with a single, efficient implementation.
- Design and prototype implementation of the novel wireless offline RTB application. The RTB design is argued to be secure with a proper configuration and shown to overcome the processing and scalability limitations of online systems.

The rest of the introductory part is organized as follows:

Chapter 2 introduces the security considerations related to wireless communications, specifically to WLANs.

Chapter 3 focuses on the field of cryptography, which provides the tools for implementing security as considered in this Thesis.

Chapter 4 surveys the security of the most significant and emerging WLAN technologies.

Chapter 5 concentrates on hardware architectures for WLAN cryptography.

Chapter 6 presents the main results of the research.

Chapter 7 summarizes the contents of the publications and clarifies the contribution of the author.

Chapter 8 concludes the Thesis.

2. SECURITY CONSIDERATIONS IN WIRELESS COMMUNICATIONS

This chapter examines general security considerations related to wireless communications, specifically in the light of WLANs. Wireless security threats as well as common security objectives and services are introduced. The properties of (cryptographic) security realizations on different protocol layers are compared and the importance of a WLAN link layer security realization is argued. Cryptographic mechanisms for security services are covered in Chapter 3. The security mechanisms and issues of specific WLAN technologies are examined in detail in Chapter 4.

2.1 Wireless Security Threats

Four types of security threats and attacks, illustrated in Fig. 3, can be identified for communication systems: *interception*, *modification*, *fabrication*, and *interruption* [134, 223]. Due to the broadcast nature of the communication channel as well as the usage of unlicensed frequency bands, the threats and resulting attacks are specifically inherent in WLANs [42, 201].

The attack types can be categorized into two main classes: *passive attacks* and *active attacks* [134, 223], presented in Fig. 4. A passive attack can either result in the disclosure of message contents (*eavesdropping*) or successful *traffic analysis*. In traffic analysis, the adversary is not able to learn message contents but is able to find out useful information by analyzing e.g. message headers and transmission frequencies [84].

Active attacks consist of modification and fabrication of messages and interruption of transmissions. Masquerading and message replay are the two forms of fabrication attacks. Modification includes changing, delaying, deleting, and reordering messages. The wireless channel, processing constraints, and limited battery life make WLAN devices specifically vulnerable to interruption, i.e. Denial-of-Service (DoS), attacks [42, 94].

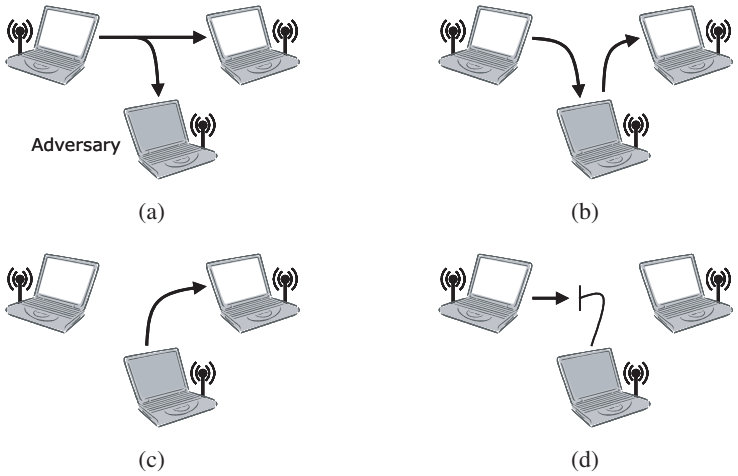


Fig. 3. *Security threats and attacks in communication systems: (a) interception, (b) modification, (c) fabrication, and (d) interruption [223].*

The portability of WLAN devices exposes them to additional security threats compared to wired devices. An adversary can easily gain physical access to a device [66, 103, 189], which calls for protecting data stored in WLAN devices as well as locally limiting the usage to authorized entities [P10].

2.2 Security Objectives and Services

Clearly, protection against the various threats of WLANs is required. The protection is realized by providing *security services*. A security service is defined as a method to provide some specific aspect of security [171]. A security service consists of one or more components called *security mechanisms (procedures)*, which have been

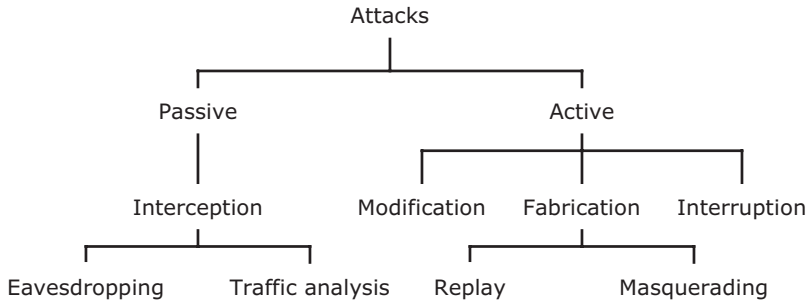


Fig. 4. *Classification of attacks.*

designed to prevent, detect, or recover from attacks. The goal of a communication system implementing security is to achieve the following *security objectives* using corresponding security services [171, 223]:

- *Confidentiality* meaning that information can only be seen by authorized entities.
- *Integrity* ensuring that information has not been corrupted or altered by unauthorized entities.
- *Availability* guaranteeing that information is available to authorized entities within predetermined response times.
- *Authentication* providing assurance of the identities of entities. *Entity authentication* assures that an entity really is the one it claims to be and *data origin authentication* ensures that the data are from the source they claim to be from. Data origin authentication implicitly covers the integrity of data and is often referred to as *data authentication* or *message authentication*.
- *Non-repudiation* preventing an entity from denying previous commitments or actions in case disputes arise.

2.3 Network Security Model

Fig. 5 illustrates the security model of communication networks, including WLANs [223]. The data storage of a WLAN device can also be seen as a form of a communication channel, communicating information into the future [P10].

The goal of the two principals is to exchange messages through the communication channel without that the adversary is able to threaten the security objectives described above. The Trusted Third Party (TTP) is utilized for creating a secure channel between the two principals. TTP manages and distributes secret information (*keys*) required for the establishment of the secure channel. The management procedures of secrets are referred to as *key management* [17]. In some cases, TTP is absent and the principals exchange the establishment information e.g. by meeting in person.

After the two principals have obtained the establishment information, a communication session between the principals typically consists of the following phases:

1. *Entity authentication*: The principals verify each other's identities utilizing the channel establishment information and an authentication protocol.

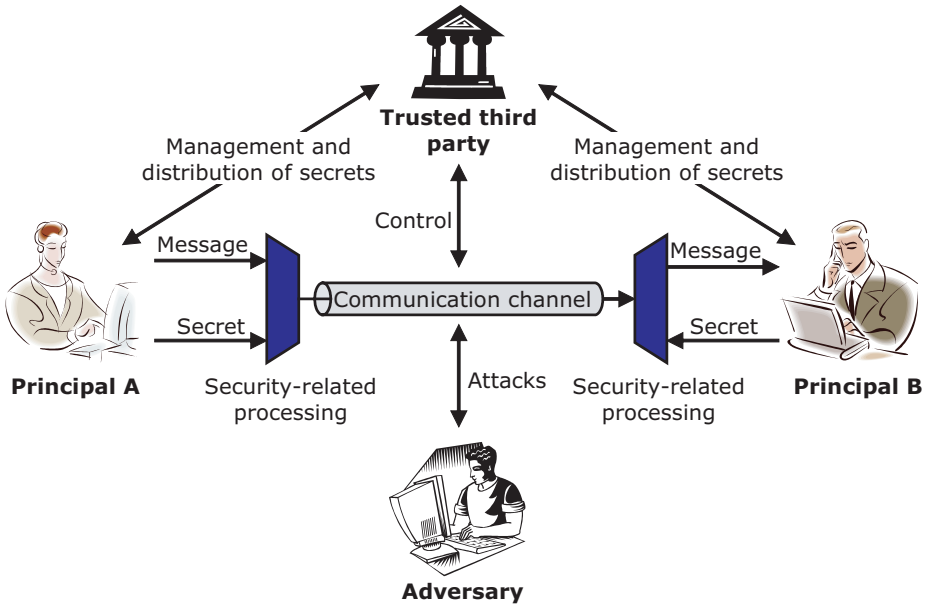


Fig. 5. *Security model of communication networks [223].*

2. *Key agreement*: The principals agree on a temporary secret for protecting the communications during the rest of the session.

3. *Protected communications*: The principals use the temporary secret for the security-related processing in the transmission and reception of messages.

4. *Destruction of session parameters*: The principals destroy the temporary secret in order to prevent compromising the session through leaking the secret later.

2.4 Placement of Security Services

There are several properties that are affected by the choice of the protocol layer for the realization of the security services in WLANs. In this section these choices are compared. The lowest feasible layer for achieving the security objectives by the means of cryptography is the MAC layer.

The higher the protocol layer is the larger the distance for the protection also is. Here the distance refers to the number of network devices through which a protected message travels. The application layer can provide end-to-end protection independent of the distance of the principals. The MAC layer only ensures the security of wireless links.

An end-to-end realization can provide interoperability with the lowest effort since the data protected with security mechanisms appear as regular payload data to the lower protocol layers as well as intermediate network devices [90]. On the contrary, a MAC layer implementation requires that all the intermediate devices are capable of security processing. Nevertheless, the higher protocol layer is chosen, the larger the number of required security implementations gets. Whereas the application layer implies a separate security implementation for each application, the MAC layer can protect all applications with a single implementation.

An application layer implementation can result in a large initial communication overhead over a single WLAN link since each application must negotiate its own protected connection with the peer. On the other hand, a MAC layer implementation means larger overhead per message during the regular data transfer since MAC layer messages are typically smaller than higher layer messages and each of them includes security-related fields, e.g. for integrity verification.

The MAC layer protects the largest amount of data, transparently protecting all the communications of a WLAN device. An application implementing security can only protect itself and, for instance, the Internet Protocol Security (IPsec) [136] and Transport Layer Security (TLS) [43] protocols protect only themselves and the applications run above them. Specifically, the MAC layer has the advantage in preventing traffic analysis as all the higher layer headers can be hidden.

Typically, processing at the MAC layer is carried out by embedded implementations including dedicated hardware/firmware whereas higher protocol layers are implemented in software. Therefore, the MAC layer is far better in energy-efficiency but also more difficult to update. Furthermore, the application layer requires more processing before tampered data protected with an integrity verification field are discarded. The data has to travel through the other protocol layers and assembled from the lower layer fragments. As a result, a larger amount of data becomes discarded as well since complete messages are deduced tampered even if only one of their MAC layer fragments was invalid. On the other hand, the MAC layer has more data to protect, which increases the amount of security-related processing but also significantly improves security as discussed above. MAC layer security implementations provide better protection against DoS attacks due to the early decisions and the processing efficiency.

As a summary, a WLAN MAC layer security implementation should always be utilized as it protects the largest amount of data and the highest number of applications and protocols. It provides high energy-efficiency as well. When the topology of

the connection between the principals is not known and/or the intermediate network devices cannot be trusted, also an application layer implementation is required for ensuring the end-to-end security. Beyond network communications, application layer implementations are required for locally protecting data in WLAN devices [P10].

2.5 Discussion

The most reliable security solutions can be realized by respecting the most recent knowledge and accepted practices [28]. It is beneficial to utilize components that have proven security bounds [77] and that are long-lived, widely-used, and trusted. Instead of adding security services after the other parts of a system are finished, security issues should carefully be considered throughout the system design process. Security designs following these recommendations are presented in [P9, P10].

A typical misconception is that standardized security solutions are always secure. An example of a failure from the WLAN field is the Wired Equivalent Privacy (WEP) protocol of IEEE 802.11 [112], examined in Chapter 4. However, a benefit of a widely utilized and trusted security standard, such as AES [61], for an application designer is that she can safely decide to use it. If the standard turns out to be insecure, no one can blame the designer for making the choice as everybody else has also trusted it. On the contrary, if the designer chooses to use something non-standard which later fails, she will be blamed for not choosing the standard [65].

An interesting aspect about the relationship of academic research and security standards is that it is far more rewarding for researchers to search and publish flaws in established standards than get involved in the standardization process [250]. Therefore, the academic research on WLAN security has also concentrated on evaluating the security of standardized WLAN technologies. The other focused field has been the implementation of the security protocols and algorithms related to standard WLANs.

If a security mechanism of a communication protocol is found flawed, the protocol can either be fully redesigned or a new security layer can be added above it. Even though a completely new design potentially results in higher level of security, there are benefits in the latter approach as well [P9]. If the flawed mechanism is widely deployed and implemented, specifically as fixed hardware components, the approach can be used in order to be backwards compatible with the old design and to enable higher level of security using the already fielded components. The drawback is that the approach is constrained by the existing components, which can require trading security to feasibility and interoperability [31, 250, P9]. Both the methods have been

utilized in the fixed IEEE 802.11 security design [118]. The latter approach is utilized for Bluetooth [24] in [P9].

Due to the numerous security threats of wireless communications and the security problems of standard wireless technologies, National Institute of Standards and Technology (NIST) [181] has published a 120-page special publication discussing the considerations and recommending guidelines for the usage and management of wireless handheld devices in its agencies [134]. The publication specifically considers the security risks associated with WLAN devices utilizing IEEE 802.11 and Bluetooth.

3. INTRODUCTION TO CRYPTOGRAPHY

This chapter introduces the field of cryptography, highlighting the main ideas and common practices related to the work presented in this Thesis.

3.1 Basic Terminology

Cryptography can be defined as a science of mathematical techniques related to information security and aiming at achieving the security objectives of Section 2.2 [171]. One of the basic applications of cryptography is *encryption*, which is used for providing confidentiality. An *encryption algorithm* (*cipher*) transforms legible information (*plaintext*) into an illegible format (*ciphertext*) using a piece of information called an *encryption key*. The inverse process of encryption is called *decryption*. Plaintext cannot be recovered from ciphertext within a reasonable amount of time without knowing the corresponding *decryption key*. Hence, the decryption key provides authorized entities with a *trapdoor*, a secret piece of information that enables reversing the encryption process.

An encryption process can be presented as

$$C = E_e(P), \tag{1}$$

in which C stands for ciphertext, P for plaintext, and E_e refers to encryption using the encryption key e. Similarly, decryption is

$$P = D_d(C), \tag{2}$$

in which D_d denotes decryption under the decryption key d.

3.2 Cryptographic Algorithms

Since encryption algorithms can be utilized for providing other security services beyond confidentiality, they have a significant role in cryptographic security service realizations. Generally, cryptographic algorithms can be divided into three categories:

secret-key algorithms, *public-key algorithms*, and *keyless algorithms*. The categorization is illustrated in Fig. 6. The figure lists only the most utilized subcategories [171].

3.2.1 Secret-Key Algorithms

The decryption key of a secret-key encryption algorithm can effortlessly be determined from the encryption key and vice versa [171]. Typically, the same key is directly applicable to both encryption and decryption, i.e. $e = d$. Thus, these algorithms are often referred to as *symmetric-key algorithms* or *shared-key algorithms*. In the remainder of this Thesis the encryption key of a secret-key encryption algorithm is referred to as K and the decryption process D_K is assumed to include the derivation of the decryption key from K. Secret-key encryption algorithms can be further divided into *block ciphers* and *stream ciphers*.

Block Ciphers

A block cipher transforms a plaintext block to a ciphertext block of the same length. Thus, a block cipher can be thought as a memoryless bijective substitution, deterministically mapping plaintext blocks to the corresponding ciphertext blocks under the influence of the secret key [171]. A complete plaintext is encrypted and ciphertext decrypted by processing a block at a time. For example, the encryption is

$$C_i = E_K(P_i) \quad (i = 0, 1, ..., n-1), \tag{3}$$

in which P_i is a plaintext block, C_i a ciphertext block, and n the length of the plaintext and ciphertext as a number of blocks. For instance, the size of the block and the key can be 128 bits.

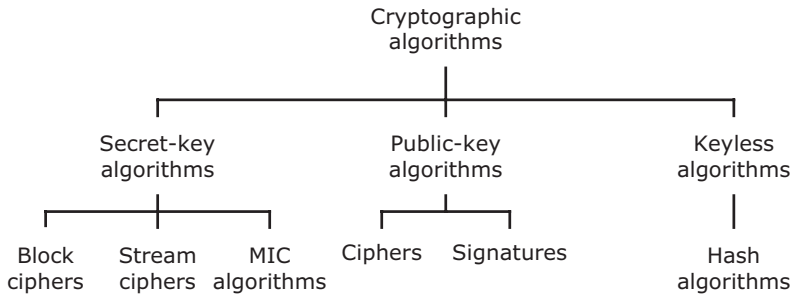


Fig. 6. *Classification of cryptographic algorithms.*

Previously, the most widely used and trusted block ciphers have been Data Encryption Standard (DES) [59] and Triple-DES (3DES) [18], standardized by NIST[1]. Currently DES is deprecated due to its small key size (56 bits) [18] and 3DES is often considered too slow and complex because of its bitwise operations, large number of operations, and small block size (64 bits). However, 3DES is still trusted and used in a number of applications. In the year 2001, the new standard block cipher AES [61] was announced by NIST as a replacement for DES and 3DES. It uses the block size of 128 bits and the key sizes of 128, 192, and 256 bits as well as provides a number of trade-offs for implementations [P2, P5]. AES has rapidly become the default choice in most new applications. It was derived from the block cipher called Rijndael.

Utilizing a block cipher directly for encryption is not secure in most applications. As can be seen in Eq. 3, similar plaintext blocks of a message result in similar ciphertext blocks, which can give an attacker an advantage for breaking the encryption scheme. This method for using a block cipher (*mode of operation*) is called the Electronic CodeBook (ECB) mode [48].

Instead of the ECB mode, block ciphers should usually be utilized in a mode of operation which evolves the encryption function as the processing proceeds [171]. Such modes related to WLANs are CounTeR (CTR) [48], CipherBlock Chaining (CBC) [48], as well as CTR with CBC Message authentication code (CCM) [242], described in [P2, P9]. In this Thesis, the CBC message authentication code technique is referred to as CBC Message Integrity Code (CBC-MIC) as the acronym MAC has been reserved for medium access control. CBC-MIC [58] is a technique for using a block cipher for data authentication.

Stream Ciphers

In contrast to block ciphers, a stream cipher encrypts a bit or a byte of plaintext at a time. The key size is e.g. 128 bits. Similarly to the CTR and CBC modes of block ciphers, the encryption function evolves along with the processing. Hence, a stream cipher contains memory as a form of an internal state [171].

Stream ciphers often generate a pseudo-random key stream which is combined with

[1] Data Encryption Standard (DES) is actually the name of the NIST publication [59]. The publication defines two algorithms, Data Encryption Algorithm (DEA) and Triple Data Encryption Algorithm (TDEA). However, these two algorithms are typically referred to as DES and Triple-DES, respectively. DEA has been deprecated and TDEA reaffirmed in [18].

plaintext to produce ciphertext by performing an eXclusive-OR (XOR, $\oplus$) operation

$$c_i = p_i \oplus k_i \quad (i = 0, 1, ..., m-1), \tag{4}$$

in which p_i is a bit (a byte) of plaintext, c_i a bit (a byte) of ciphertext, k_i a bit (a byte) of key stream, and m the length of the plaintext and ciphertext in bits (bytes). The key stream is based on the encryption key. Decryption is performed in the same way, simply by switching the plaintext into the ciphertext

$$p_i = c_i \oplus k_i \quad (i = 0, 1, ..., m-1). \tag{5}$$

It is necessary that the key stream is never reused under the same encryption key [28]. This same requirement applies to the CTR mode, which alters a block cipher into a stream cipher, operating in the same way on blocks instead of bits or bytes. Violating this requirement has lead into many vulnerabilities in IEEE 802.11 as discussed in Section 4.1.2.

Related to the WLAN technologies addressed in this Thesis, the stream ciphers are RC4 [217] and ciphers based on Linear Feedback Shift Registers (LFSR) [171], such as that of Bluetooth [24]. A LFSR is depicted in Fig. 7. It consist of delay elements, each capable of storing one bit and having one input and one output. The output of the highest element is used as k_i in Eq. 4 and fed back to define the next state of the LFSR. The encryption key defines the initial state of a LFSR-based stream cipher. RC4 is described in [P5].

Other Secret-Key Algorithms

In addition to encryption algorithms, there exist other secret-key algorithms specifically designed for providing data authentication, non-repudiation, and pseudo-random number generation [171]. A secret-key algorithm utilized in WLANs is Hashed Message Authentication Code (HMAC) [145]. It is a construction that can be used for modifying a hash algorithm (Section 3.2.3) to provide data authentication. HMAC produces fixed-size Message Integrity Codes (MIC) for variable-length messages using a secret key.

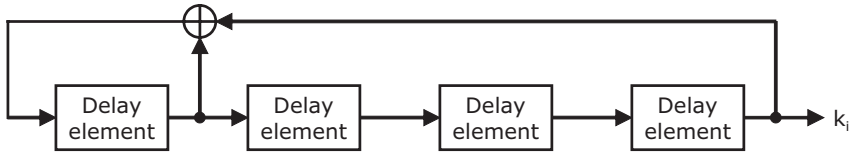


Fig. 7. *Example of a 4-bit Linear Feedback Shift Register (LFSR).*

3.2.2 Public-Key Algorithms

As opposed to secret-key algorithms, it is infeasible to determine the decryption key from the encryption key for a public-key algorithm and vice versa [171]. Therefore, these algorithms are also called *asymmetric algorithms*. The encryption key e is called *public key* and the decryption key d is *private key*. The benefit of the asymmetry is that the public key can be freely distributed and published. While any other entity can use the public key for encryption, only the entity possessing the private key can perform decryption. Compared to secret-key algorithms, the property significantly alleviates key distribution.

Public-key algorithms are considerably slower and they often use significantly larger keys than secret-key algorithms [155]. Hence, in WLANs public key algorithms are typically only used for the entity authentication and key agreement phases of Section 2.3. Another application of public-key encryption algorithms is digital signing which can be used for data authentication as well as non-repudiation services [171, P10].

In communication networks, digital signatures are specifically utilized for creating *public-key certificates*. A public-key certificate is a digitally signed piece of information containing a public key, a validity period, and the identity of the owner of the public-key. In the model of Fig. 5, TTP issues the public-key certificates to the principals and the certificates allow the principals to verify the authenticity of each other's public keys before proceeding with the entity authentication. The principals have earlier obtained the public-key of TTP. In this context, TTP is usually called Certificate Authority (CA) and the overall arrangement is Public-Key Infrastructure (PKI) [87].

Public-key algorithms are based on mathematical problems that can be computed effectively but inverting them is believed to be infeasible without the knowledge of a trapdoor. The most common problems are the integer factorization problem, the discrete logarithm problem, and the elliptic curve discrete logarithm problem [171]. The cryptographic algorithms and protocols commonly related to WLANs and the designs of this Thesis are RSA [206], Diffie-Hellman-based key agreement protocols [45], and Secure Remote Password (SRP) protocol [254]. The techniques are traditionally derived from the integer factorization problem and the discrete logarithm problem. The common operation for the techniques is the modular exponentiation of large integers using an odd modulus.

3.2.3 Keyless Algorithms

The most widely utilized keyless algorithms are *hash algorithms*. A hash algorithm takes in an arbitrarily-length data input and compresses it into a fixed-size output value called *hash*. The output is e.g. 160 bits wide. In contrast to the hash functions of other fields, the distinguishable properties of cryptographic hash algorithms are *preimage resistance*, *2nd preimage resistance*, and *collision resistance* [171]. For a preimage-resistant hash algorithm, it is computationally infeasible to find any input that hashes to a given output. 2nd-preimage resistance means that it is computationally infeasible to find any second input which hashes to the same output as any specified input. Collision resistance ensures that it is computationally infeasible to find any two distinct inputs that hash into the same output.

The hash algorithms used in WLANs are Message Digest 5 (MD5) [205] and the family of Secure Hash Algorithms (SHA) [60]. Recently, MD5 and SHA-1 of the SHA family have been found to provide lower level of security than previously assumed [237, 238]. Hence, other algorithms, e.g. SHA-256 [60], are suggested to be used in new applications.

3.3 Using Cryptography for Protecting Communications

An exemplar protected communication session involving public-key, secret-key, and hash algorithms and using the network security model of Section 2.3 is described below. The principals A and B have generated public key–private key pairs and requested certificates for their public keys from TTP earlier. It must be noted that the procedures below are only simplified examples, aiming at illustrating the main principles of the utilization of cryptography for protecting communications. For example, the first step is vulnerable to a man-in-the-middle attack if the identities of the entities are not included in the encrypted messages [163]. It is particularly easy to design entity authentication protocols which seem secure but turn out completely flawed after more detailed analyses [65].

1. *Entity authentication*: The principals transmit their public-key certificates to each other and verify them using the public-key of TTP. A generates a random number R_A, encrypts it with B's public key, and transmits the result to B. B decrypts the message with her private key and gets the result R'_A. Then B generates another random number R_B and encrypts the concatenation of R'_A and R_B using A's public key. B sends the encryption result to A, who decrypts

the message and verifies that $R'_A = R_A$. The decryption result R'_B is encrypted and sent back to B, who verifies that $R'_B = R_B$.

2. *Key agreement*: A and B generate random numbers K_A and K_B, respectively. B encrypts K_B with A's public key and sends the result to A and A encrypts K_A with B's public key and sends the result to B. Then the principals decrypt the received messages and hash the concatenation of K_A and K_B to yield a temporary secret key K, typically known as a *session key*.

3. *Protected communications*: The session key K is divided into two parts, the first one is used as the encryption key and the second one as the secret key for MIC computation during the communications.

4. *Destruction of session parameters*: The principals destroy all the exchanged parameters, including K. A new session between A and B requires restarting from the entity authentication phase.

The purpose of the random numbers in the example is to provide *freshness*. An object, such as a message, a session, or a key, is fresh if it is new and recently created, not replayed, reused, or expired. Freshness can be seen as a component of authentication. The entity authentication method used above is a *challenge-response* scheme. A principal challenges the other with an encrypted random number, which the other principal decrypts. By presenting the correct value in the response, the principal proves her fresh knowledge of the private key.

In addition to freshness, the key aspects that should typically be ensured by any secure communication channel are the mutual entity authentication, the influence of both the principals in the generation of the session key K, the usage of separate keys for separate purposes, as well as the protection of the actual communications for both confidentiality and data authentication [P10].

Mutual authentication means that both the principals prove their identities to each other. As an opposite, in *one-way authentication* only one of the principals becomes authenticated. The facts that A and B are both involved in determining the shared secret K and that K is a hash of their values ensure that even if one of the principals generates her part of the secret poorly, K is still secure. Separated keys prevent endangering the others if one of them is compromised.

A typical misconception is that encryption equals security. Encryption can directly provide only confidentiality. For example, authentication can often be more important than encryption [65, P10]. An attacker can typically cause more damage by

breaking authentication than by breaking confidentiality. Therefore, an encryption scheme should always include a data authentication mechanism.

All these aspects have been violated in the security designs of WLANs, which has led to various weaknesses and attacks.

4. SECURITY IN WLAN STANDARDS

During the recent years, the intensive development of standards and standard-like specifications has been fueling the research work and the markets of WLANs [105]. This has also reflected to the academic research on WLAN security as it has concentrated on evaluating and implementing the security procedures of standard WLANs as well as proposing improvements for them. Being no exception to a typical security design process, the development of WLAN security has been a constant race between hackers, researchers, vendors, and standardization bodies.

This chapter reviews the security designs of the most significant WLAN technologies and their current status. The focus is placed on IEEE 802.11 [121], Bluetooth [22], which corresponds to IEEE 802.15.1 [120], and the emerging IEEE 802.15.4 [116], which is utilized by the ZigBee specification [262]. IEEE 802.11 is the most widespread [125] and the most intensively researched WLAN technology. The related academic research is reviewed in the sections discussing the security vulnerabilities and improvement proposals of the technologies, in Chapter 5, and in the publications.

Throughout this chapter, messages transmitted by the MAC layer are referred to as *frames*, even if the technology specification in question addressed them with a different term. The messages of the protocol layer above MAC are referred to as *packets*. If necessary, packets are fragmented across several frames. In the text, a piece of a packet, delivered in a single frame, is referred to as a *frame payload*. Furthermore, a secret key used for protecting a wireless link at the MAC layer is referred to as a *link key*. A *pairwise key* is a link key shared only by two principals and a *group key* is a link key shared by more than two principals, possibly by the whole WLAN. The keys and their usage are illustrated in Fig. 8.

4.1 IEEE 802.11

The IEEE 802.11 technology has been developed for more than ten years. During that time improvements have been made to support higher data rates, different frequency

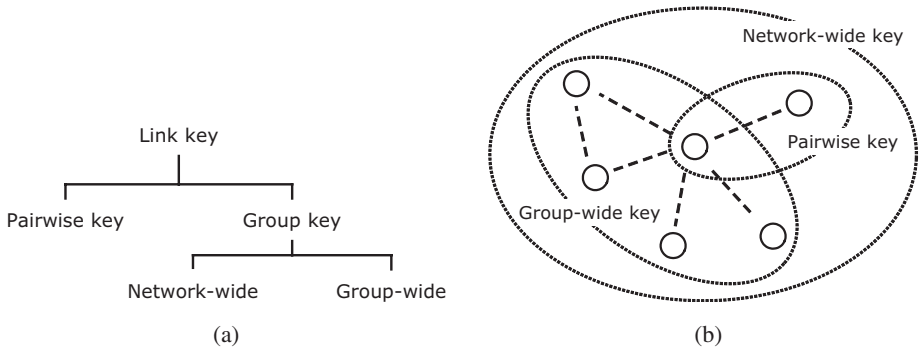


Fig. 8. *Types of link keys (a) and their usage (b) in Wireless Local Area Networks (WLAN).*

band, and better interoperability. However, despite of numerous vulnerabilities, the security design has been modified and improved only recently [118]. The original security design [112] of the IEEE 802.11 technology is referred to as 802.11 and the updated security standard [118] as 802.11i.

The 802.11 standard [112] specifies a single MAC layer and three physical layers, one for infrared and two for 2.4 GHz radios. The physical layers provide up to 2 Mbit/s link rate. The supplement standards, 802.11b [113], 802.11g [117], and 802.11a [114], define additional physical layers for the 2.4 GHz and 5.0 GHz radio bands. The link rates are increased to 11 Mbit/s and 54 Mbit/s. All the variants utilize the common MAC layer specified in the original 802.11 standard. Currently, most 802.11 products include both the radio bands as well as the higher link rates [125]. The technology is being further developed e.g. for supporting Quality of Service (QoS), higher data rates, interworking with other technologies, and data exchange between high-speed vehicles [121]. The 802.11 technology supports centralized and ad hoc topologies.

The 802.11 technology is often referred to as Wi-Fi (Wireless Fidelity). Wi-Fi Alliance is an industry consortium established for promoting the spread and the interoperability of 802.11 WLAN products [243]. The alliance grants Wi-Fi certificates to products meeting its requirements.

4.1.1 Security Design of IEEE 802.11

The 802.11 standard [112] specifies cryptographic mechanisms aiming at providing confidentiality, integrity, and entity authentication. The security is based on pairwise keys and group keys. 802.11 does not include any method for generating or

exchanging keys. Instead, it is assumed that they are pre-shared secrets. The same key, either group or pairwise, is directly utilized for all the services. Confidentiality and integrity are addressed with the WEP protocol through encryption and keyless checksumming. WEP utilizes RC4 for encryption and the checksum is computed with a Cyclic Redundancy Check (CRC) algorithm [171], which does not have the properties of a cryptographic hash function.

Fig. 9 presents the WEP scheme [134]. First, upon the transmission of a frame, a 32-bit Integrity Check Value (ICV) is computed for the frame payload. The payload is concatenated with the ICV to obtain a plaintext input for the encryption phase. Next, the link key (WEP key) is appended to an arbitrarily chosen 24-bit Initialization Vector (IV) to produce a seed for the Pseudo-Random Number Generator (PRNG) of RC4. The purpose of the IV is to supply unique seeds (per-frame keys) and thus increase the lifetime of the link key. After the encryption the ciphertext prefixed with the IV is transmitted in a frame over the wireless link. At the receiving end the encryption process is reversed and the ICV verified.

802.11 uses a challenge-response scheme called *Shared Key authentication* for entity authentication. The scheme utilizes the WEP protocol and the authentication is based on the link key. Principals are identified with their MAC addresses. In order to initiate the procedure, a device (requester) transmits a request to the device

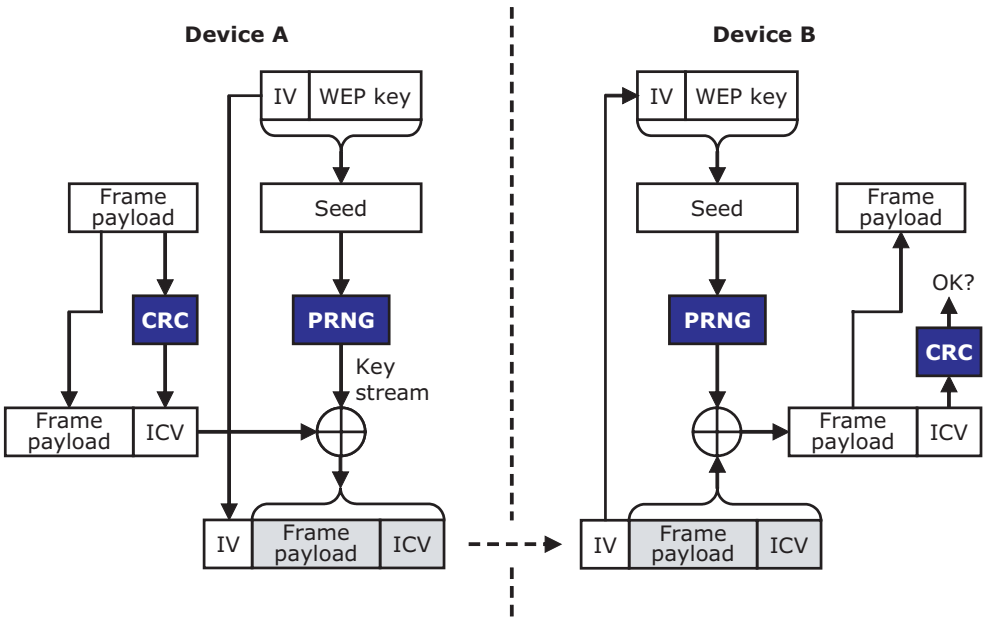


Fig. 9. *Wired Equivalent Privacy (WEP) scheme of IEEE 802.11.*

it wants to communicate with (responder). The responder looks up the link key for the MAC address supplied by the requester and replies with a random challenge. The requester chooses an IV and returns the challenge encrypted with WEP to the responder. If the response is valid, the requester is considered authenticated. As such, the authentication is one-way. For mutual authentication the roles can be switched and the procedure repeated.

The standard also specifies a scheme called *Open System authentication* but since it does not employ any secrets or cryptographic mechanisms, it does not provide any cryptographic security.

4.1.2 Vulnerabilities of IEEE 802.11

The intensive research on the 802.11 technology has shown that the 802.11 security fails in achieving each of its security objectives. The fundamental failure has been that it was developed behind closed doors and no professional cryptographers were involved [28]. The weaknesses are mainly caused by the poorly-defined key management, which results in widely shared and infinitely static link keys [13, 28]. The practice allows insider attacks and dramatically decreases the lifetime of a key [236].

With the combination of the poor key management and short, uncontrolled IVs, the WEP protocol fails in producing unique key streams and IV collisions (key stream repeats) occur frequently [28, 236]. A collision can be found by recording 802.11 traffic with off-the-shelf end-user products, i.e. a WLAN card with a standard software driver. After finding a collision, the attacker can learn the plaintext and the key stream [28, 236]. The key stream can be further used for creating and decrypting frames. It is feasible to collect even a full IV dictionary (i.e. a look-up-table of all possible IVs and their key streams) for real-time decryption [11, 28]. After collecting enough frames, the attacker can eventually recover the link key using the Fluhrer–Mantin–Shamir (FMS) attack [70, 227], which is based on certain weak IV values. A number of software tools for discovering and exploiting key streams as well as link keys are publicly available [239].

The uncontrolled IV usage breaks the security of Shared Key authentication already before the link key has been recovered [13, 28]. Since the challenge is sent out in clear, an eavesdropper can obtain a plaintext-ciphertext pair and thus a part of a key stream (according to Eq. 4). The stream can be used for en/decryption and masquerading. The attack is possible with pairwise keys by eavesdropping and spoofing a MAC address [13]. Furthermore, since the authentication only uses MAC addresses,

users cannot be identified on the 802.11 layer. As the authentication in the Access Point (AP) topology is one-way, an attacker can setup a *rogue AP* for collecting key streams, for DoS attacks, and for collecting application layer usernames and passwords (e.g. for website login) [6, 42, 85].

The encrypted ICV does not ensure integrity. Even without the knowledge of a key stream, the attacker can make chosen modifications to encrypted frames and modify their checksums accordingly [28]. After obtaining a short key stream, the attacker can exploit the ICV for extending the stream length [11, 192]. The attacker can also exploit the ICV and an AP to decrypt captured frames using truncation [143].

The MAC layer headers are not protected and thus freely modifiable, including address spoofing [31, 85]. For example, captured frames can be replayed for collecting IVs. As the management frames are not protected, DoS attacks are possible by exploiting those frames [3, 29, 85]. The globally unique MAC addresses in each frame introduce a threat to a person's location privacy [84].

4.1.3 Patching the Security of IEEE 802.11

A number of non-standardized fixes have been proposed for the 802.11 security. However, most adopted solutions have been vendor-specific and thus interoperability has suffered. They have attempted to improve the 802.11 security without requiring changes in the fielded WLAN hardware.

It has been proposed to increase the key and IV sizes as well as avoid IVs traditionally exploited by the FMS attack [5, 13]. Each of these fixes have failed to improve security. Most attacks are independent of the size of the key or the IV [3, 11, 28, 70, 236], larger sizes require only slightly longer capturing times for the FMS attack [227], and the FMS attack as well as attacking tools have been extended for thwarting the avoidance mechanisms [166, 186]. The avoidance of weak IVs imply that IV collisions occur more frequently [13].

WLAN vendors have implemented non-cryptographic filtering for controlling network access, based on hidden network identifiers and MAC address lists. These mechanisms do not provide real security as they can easily be bypassed through eavesdropping and spoofing [13, 85, 247]. It has been proposed that the sequence numbers of MAC headers could be used for detecting MAC address spoofing in 802.11 networks [85]. MAC address issues related to anonymity and solutions based on temporary addresses are discussed in more detail in [90]. A robust solution for preventing spoofing, which has also been adopted in the IEEE 802.11i standard [118], is

to protect addresses with MICs.

The more advanced vendor-specific mechanisms in the 802.11 AP topology include centralized access control with the distribution of pairwise keys. The schemes are implemented using the IEEE 802.1X architecture and encapsulation [119], described in Section 4.2. Pairwise keys can be updated periodically in order to avoid giving out too many IVs. Even though the solutions have mainly provided better security, some of them have suffered from additional security problems [46]. Moreover, proper key management cannot alone solve all the problems of 802.11 [28].

A common solution in corporate networks has been not to trust the 802.11 security at all, which has also been recommended by academic researchers [28, 227]. Instead, the wireless network has been placed outside a firewall. It has only been possible to access the wired infrastructure using VPN technologies. With a proper setup, VPN prevents outsiders from accessing the wired network but malicious operation within the wireless segment is still possible [46]. The VPN solutions are also limited to the IP protocol suites [90, 109]. The aspects related to different protocol layers have been discussed in Section 2.4.

General recommendations for repairing 802.11 against the various attacks have been presented in [13, 70, 166, 227, 236]. These recommendations have also been followed in the IEEE 802.11i security standard [118] described in the next section.

4.2 IEEE 802.11i

Due to the various flaws, IEEE Task Group I (TGi) was founded for enhancing the security of the 802.11 MAC layer. Finally in summer 2004, TGi released the IEEE 802.11i standard [118]. The standard specifies new methods for key management, entity and data authentication, and confidentiality. 802.11i is expected to overcome the flaws and scalability problems of the 802.11 security.

In the 802.11i standard, a wireless network that permits only the new mechanisms is referred to as Robust Security Network (RSN). For backward compatibility the support for WEP and Shared Key authentication has been maintained. However, they are not recommended and operating in RSN disallows their usage. A network allowing both the deprecated and the new techniques is called Transition Security Network (TSN). A connection protected with the new mechanisms is called RSN Association (RSNA) and with the conventional mechanisms pre-RSNA.

4.2.1 Entity Authentication and Key Management in RSN

RSN makes use of the IEEE 802.1X port-based network access control standard [119] for entity authentication and key agreement. The 802.11i standard adapts Extensible Authentication Protocol (EAP) [1] to the LAN environment by defining EAP over LANs (EAPOL). 802.1X does not define a concrete authentication method—it only defines an architecture and encapsulation for multi-roundtrip authentication protocols. Separate EAP specifications are required for each specific protocol. RSN requires that the authentication is mutual, does not leak information enough to allow impersonation attacks, and that the process produces fresh session keys [226]. Examples of suitable EAP methods are EAP-TLS (Transport Layer Security) [2] and EAP-TTLS (Tunneled TLS) [72], which typically utilize RSA or Diffie-Hellman-based mechanisms [36,43]. The 802.1X technology was chosen for 802.11i in order to support interoperability with other networking technologies, such as GSM/UMTS [90], and various authentication protocols, including future enhancements.

Fig. 10 presents the authentication architecture of 802.1X in RSN. The 802.1X authentication is performed between a *supplicant* and an Authentication Server (AS). The target of the authentication procedure is to establish a secure channel between the supplicant and an *authenticator*. For example, in the centralized topology the supplicant can be a laptop, the authenticator an AP, and the AS a Remote Authentication Dial-In User Service (RADIUS) [202] or Diameter [30] server. In the ad hoc topology, each wireless device has to support both the supplicant and authenticator roles.

In RSN, both the supplicant and the authenticator contain a Port Access Entity (PAE), which runs the algorithms and protocols associated with the authentication mechanisms [119]. The authenticator has two types of logical ports, uncontrolled and

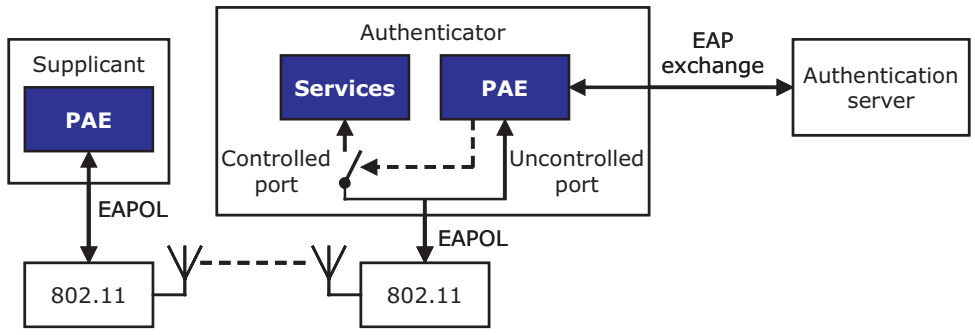


Fig. 10. *Authentication architecture of Robust Security Network (RSN).*

controlled. A single uncontrolled port is used for performing authentications. After a successful authentication, a controlled port is opened and associated for the created connection. Before the supplicant has been authenticated, the authenticator PAE permits its traffic only through the uncontrolled port and accepts only authentication-related messages.

Only the Open System authentication can be performed prior to the 802.1X authentication in RSN. Hence, the original 802.11 layer only operates as a pipe conveying messages between the supplicant and authenticator PAEs.

A typical authentication message exchange in RSN is depicted in Fig. 11 [33, 90]. The EAP messages between the supplicant and the authenticator are encapsulated in EAPOL messages. The requests and responses are exchanged between the supplicant and the AS until the authentication and key exchange protocol in question is finished. After a successful 802.1X authentication, the AS transmits a 256-bit Pairwise Master Key (PMK) to the authenticator. The supplicant generates it from the data exchanged during the authentication process. PMK is the pairwise key between the supplicant and the authenticator.

After the 802.1X authentication, the authenticator and the supplicant perform a 4-

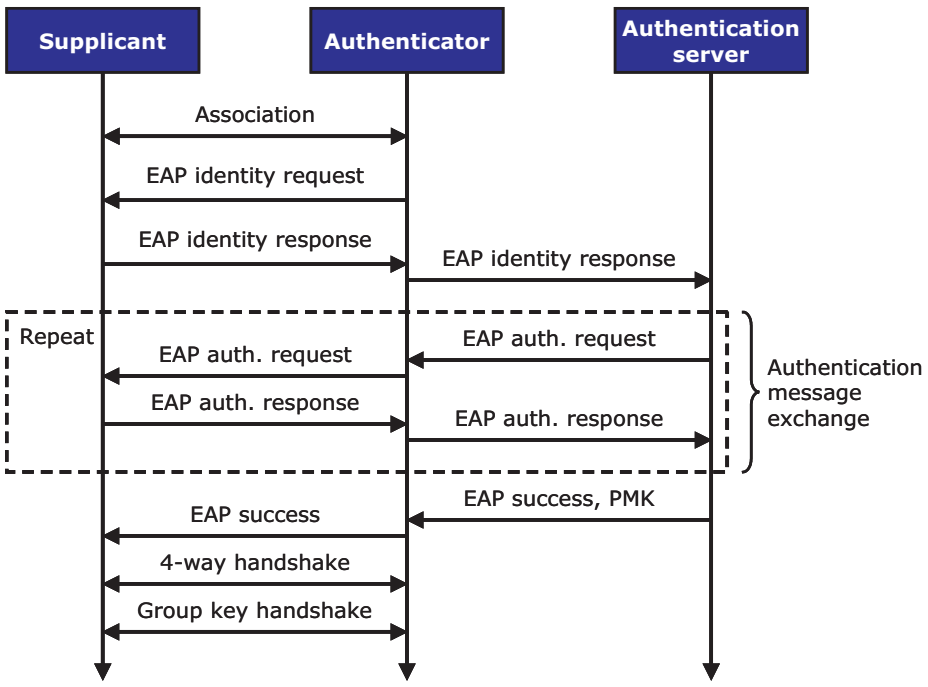


Fig. 11. *Typical message exchange of a Robust Security Network (RSN) authentication.*

way handshake to confirm that they both share the same PMK. During the handshake they generate a new key called a Pairwise Transient Key (PTK). The handshake can be used for refreshing the RSNA with a cached PMK. The 4-way handshake is also used when a Pre-Shared Key (PSK) is installed in the wireless devices. In that case, the 802.1X authentication is omitted and only the handshake is performed using PSK as PMK. Two more handshakes are defined for agreeing on a Group Transient Key (GTK) for group communications and a STAKey for protecting direct device-to-device communications in the centralized topology.

All the handshakes are encapsulated in EAPOL key messages [118, 119] and protected with EAPOL-Key Confirmation Key (KCK) (data authentication) and EAPOL-Key Encryption Key (KEK) (confidentiality). KCK and KEK are parts of PTK. The algorithms used for the protection are RC4 and HMAC-MD5 or AES and HMAC-SHA1, correspondingly to the chosen cipher suites described in the following section. After the key agreement, PTK, GTK, and STAKey are used to derive separate temporary keys for WEP in pre-RSNA and for the confidentiality and data authentication mechanisms of RSNA.

4.2.2 Confidentiality and Data Authentication in RSN

The 802.11i standard addresses the confidentiality and integrity problems of 802.11 with two mechanisms, Temporal Key Integrity Protocol (TKIP) and CCM Protocol (CCMP). Whereas TKIP is a wrapper around WEP constrained by the existing hardware, CCMP is a completely new design. TKIP is optional and CCMP mandatory in RSN.

TKIP

TKIP fixes the WEP flaws in the 802.11 products through software/firmware updates. It adds keyed integrity protection (data authentication), replay protection, and per-frame key mixing on top of WEP. The protocol assumes that the communicating principals share a fresh TKIP key, consisting of one 128-bit key for encryption and two 64-bit keys for data authentication (one in each link direction). The TKIP encapsulation is depicted in Fig. 12. The WEP block contains the encapsulation of Section 4.1.1. The inputs seed and frame payload can be directly mapped to the seed and the payload of Fig. 9, respectively.

TKIP computes a 64-bit MIC for each packet, protecting also the addressing fields. The 802.11i standard defines a new keyed algorithm named Michael for the purpose.

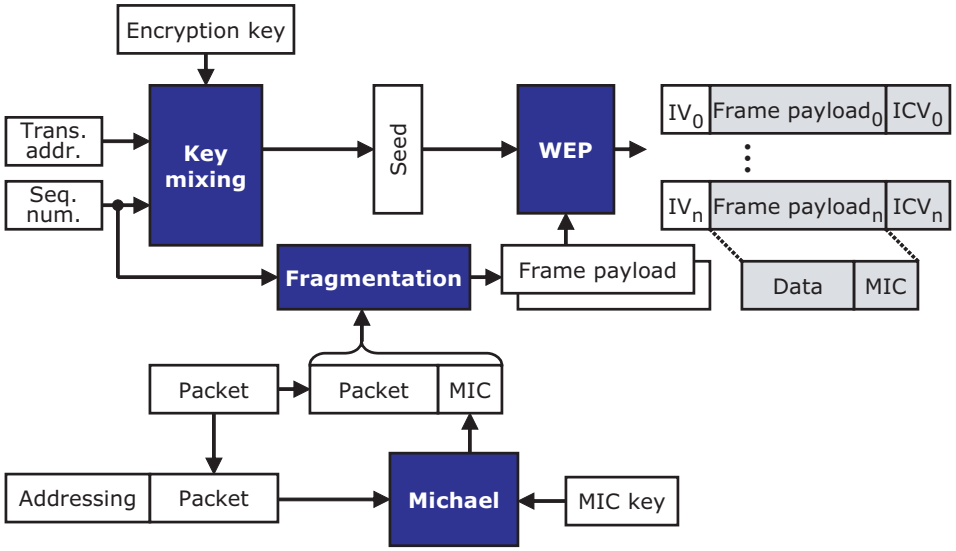


Fig. 12. *Temporal Key Integrity Protocol (TKIP) encapsulation of IEEE 802.11i.*

TGi considered conventional MIC algorithms computationally too expensive for the existing hardware [31]. The packet with the MIC is fragmented into one or more frame payloads.

For encryption, TKIP generates a unique RC4 seed for each frame. The 128-bit seed is created by mixing together the encryption key, transmitter MAC address, and a 48-bit sequence number. The purpose of the mixing is to thwart the FMS attack [31]. During the lifetime of the TKIP key, the sequence number is a unique, monotonically increasing value assigned to each frame. In the transmitted frame, the originally arbitrarily chosen IV is replaced with an extended TKIP IV constructed from the sequence number. The freshness of the sequence number and the correctness of ICV and MIC are verified at the receiving end.

CCMP

CCMP is the default security suite in RSN. It uses AES for both confidentiality and data authentication. In addition, CCMP provides replay protection. In contrast to TKIP, CCMP operates on frames, not on packets. The utilized encryption mode, CCM, has proven security bounds assuming that the underlying cipher does not have weaknesses [129]. CCMP utilizes a single 128-bit key for MIC computation and encryption. Originally, TGi debated specifying a security suite called Wireless Robust Authentication Protocol (WRAP) instead of CCMP. It would have utilized AES in

the Offset CodeBook (OCB) mode [147]. However, the proposal was discarded due to patent issues.

The CCMP encapsulation is presented in Fig. 13. Each frame is assigned a unique 48-bit sequence number, which provides per-connection replay detection in the same way as in TKIP. The 64-bit MIC covers chosen header fields and the payload. For the CTR encryption the source address and the sequence number are combined into a unique *nonce*, which makes per-frame keying unnecessary in CCMP.

4.2.3 Wireless Protected Access

In order to improve the security of 802.11 networks already before TGi had finished its work, Wi-Fi Alliance specified Wi-Fi Protected Access (WPA) [244]. WPA is a simplified, forward-compatible version of the 802.11i standard. It includes the RSN entity authentication and TKIP. For simpler installations, it supports the PSK mode, in which the link keys can be manually entered. The RSN configuration with CCMP is referred to as WPA2 by Wi-Fi Alliance.

4.2.4 Vulnerabilities and Improvements for IEEE 802.11i

The main improvements of 802.11i are the defined key management based on 802.1X, replay protection, as well as the robust confidentiality and data authentication mechanisms using CCMP. Instead of supporting only device authentication, the framework allows authenticating users at the MAC layer as well. However, due to the difficulty of the security design process, even the new standard contains weaknesses.

A formal proof for the correctness of the RSN authentication (i.e. the authentication procedure meets its goals) is presented in [93], assuming that the principals oper-

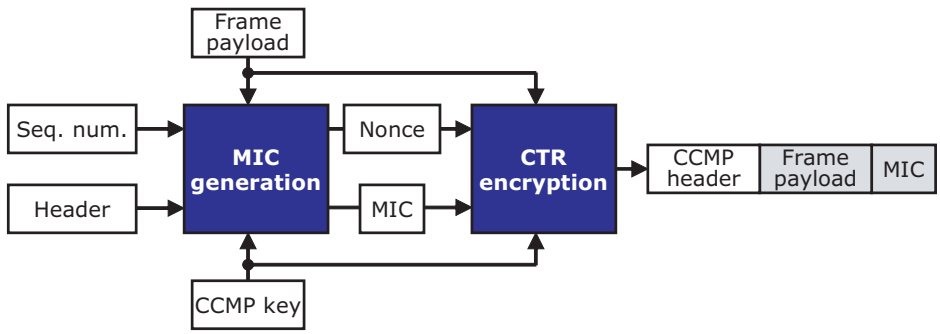


Fig. 13. *CCM Protocol (CCMP) encapsulation of IEEE 802.11i.*

ate honestly and that certain preconditions hold. In the ac hoc topology, a reflection attack can still be possible [92]. In the attack, an adversary replays the 4-way handshake messages to a device that supports both the supplicant and authenticator roles. The adversary manages to perform authentication successfully. The attack does not compromise regular data transfers since the adversary does not gain access to the session key. As a solution, devices supporting the two roles should use different PMKs for each role.

As 802.11i does not protect management and acknowledgment frames, DoS attacks are possible by tampering with those frames [57, 82, 92, 174]. IEEE Task Group W was recently established for extending the security services of 802.11i to cover management frames [121]. A DoS attack against the 4-way handshake is possible by flooding the WLAN network with the first message of the handshake, which makes the supplicants unable to finish their handshakes successfully [91]. Certain EAPOL messages should be ignored to prevent DoS as well [92]. In order to hinder DoS, authentication should also be performed before association and the authentication procedure should not always start from the beginning if one of its phases fails [92].

The network installations using PSKs are vulnerable to key recovery attacks when the installation is configured poorly or when PSKs are derived from user-friendly passphrases [69, 252]. When the same PSK is shared among several wireless devices, a malicious insider is able to force rekeying for any other pair of devices and recover the temporary keys derived from the PSK during the new key agreement [69]. A tool for mounting a passive dictionary attack on passphrases is publicly available [252].

Due to the design constraints, Michael used in TKIP is not a robust MIC algorithm, which its designers have also been well aware of [64]. Its security relies on the fact that the MIC is encrypted [251]. An attacker can recover the complete TKIP key if at least two RC4 seeds generated under partially overlapping sequence numbers are leaked [175]. However, it is not feasible to perform the attack in practice.

Earlier versions of the 802.1X protocol have been found vulnerable to session hijacking and man-in-the-middle attacks [174]. Since 802.1X is not intended to be used on its own but with an EAP method, the attacks are not applicable when a robust method is applied [57, 226]. The current 802.1X standard [119] is also alone better resistant to these attacks.

The TSN configuration should not be used since an adversary can force devices to set up pre-RSNAs even if they supported RSNAs [92]. The location privacy concerns of 802.11 deriving from the usage of globally unique MAC addresses have not been addressed in 802.11i.

4.3 Bluetooth

Bluetooth [24] is a technology defined by the Bluetooth Special Interest Group (SIG) [22]. Originally it was only intended as a simple serial cable replacement for electronic devices. Presently the technology supports various more advanced functionalities, such as ad hoc networking and AP operation for Internet connections. The ongoing development is extending Bluetooth with new features, including support for QoS, higher data rates, multicasting, and lower power consumption [22, 25]. The application area keeps expanding as new products with Bluetooth capability are constantly introduced [22].

The Bluetooth technology consists of several protocol layers ranging from the physical radio and *baseband* to object exchange and service discovery. In addition, Bluetooth SIG has specified a number of *profiles* [25], which define a selection of messages, procedures, and protocols required for supporting a specific service. Communications between Bluetooth devices can be point-to-point or point-to-multipoint. A network composed of Bluetooth devices is called a *piconet*, which consists of a *master* and up to seven active *slave* devices. Piconets can be linked together to form a larger network, *scatternet*, as illustrated in Fig. 14. The Bluetooth technology is further described in [P9].

IEEE has adopted the lowest Bluetooth layers, i.e. the radio, the baseband, Link Manager Protocol (LMP), Link Manager (LM), Host Controller Interface (HCI), and Logical Link Control and Adaptation Protocol (L2CAP), and standardized them in

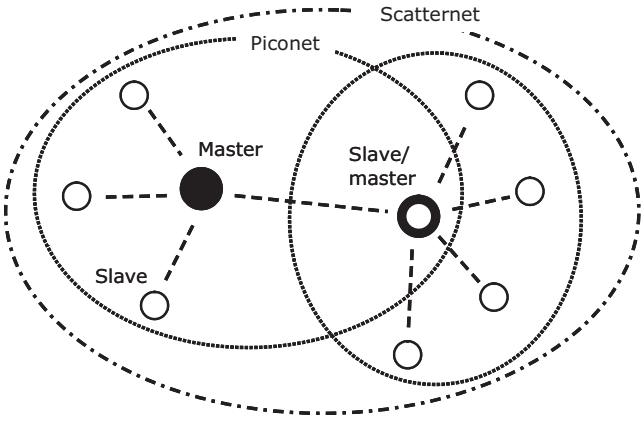


Fig. 14. *Bluetooth network topology. Piconet consists of a master and up to seven active slaves. Piconets can be connected together to form a scatternet. In a scatternet the linking devices belong to two or more piconets.*

its IEEE 802.15.1 standard [120]. The physical layer is constructed of the radio and a part of the baseband, and the MAC layer corresponds to the rest of the baseband and the higher layers up to L2CAP.

4.3.1 Security Design of Bluetooth

A Bluetooth device can operate in three security-related modes. In Mode 1, called *non-secure*, the device does not initiate any security procedures. Mode 2 is called *service level enforced security* and a device operating in this mode does not initiate security procedures before a channel establishment at the L2CAP layer. In Mode 3, named *link level enforced security*, security procedures are initiated before a Bluetooth link has been established. Whereas Mode 2 supports different security policies for parallel applications, Mode 3 enforces the same MAC layer security level for all connections [180]. In addition to the operating modes, security levels for devices and services have been specified [24, 180].

When security is applied, Bluetooth implements key management, entity authentication, confidentiality, and integrity protection. Security processing is carried out at the baseband and controlled by LM according to the requirements of the protocol layers above the MAC layer. Due to the large number of adjustable parameters, Bluetooth SIG has published additional recommendations for configuring the security procedures in different profiles [75]. Furthermore, the ambiguities of the Bluetooth specification related to the encryption of piconet broadcasts have been clarified [178].

Entity Authentication and Key Management in Bluetooth

The Bluetooth security is based on three types of link keys, *initialization keys*, *combination keys*, and *master keys*. The earlier versions of the Bluetooth specification included the fourth type of a link key called *unit key* but its usage has been deprecated in the newest specification [24] due to security problems [76, 127, 234]. The 128-bit link key is utilized in entity authentication. Depending on its type and the desired level of protection, the link key is also used for generating an encryption key. The Bluetooth key hierarchy is illustrated in Fig. 15.

An initialization key is typically only used when two Bluetooth devices establish a connection for the first time. The key is generated from a Personal Identification Number (PIN) code, supplied to both the devices. A combination key is generated from information shared between two Bluetooth devices. The sharing of the generation information is protected with the effective link key. A master key is a temporary

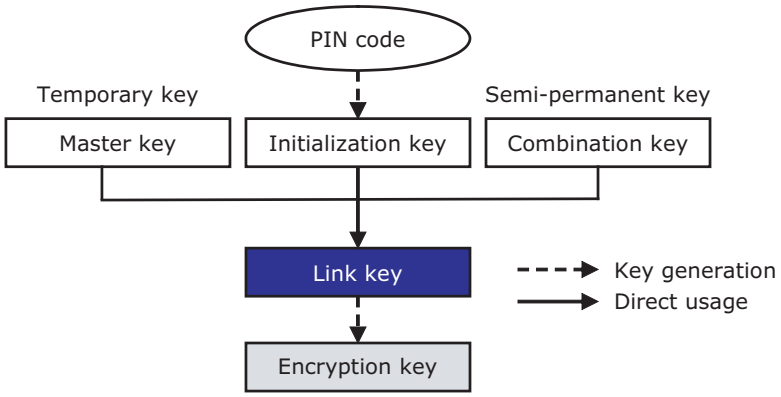


Fig. 15. *Bluetooth key hierarchy. The initialization key is derived from a Personal Identification Number (PIN) and used as a link key for protecting the exchange of another type of a key. The exchanged link key is used for authentication and optionally for generating an encryption key.*

group key distributed by the piconet master and used for protecting broadcast transmissions.

After exchanging a new link key, the devices verify its correctness and each other's identities by subsequently running a challenge-response authentication procedure. The devices are identified by unique Bluetooth addresses. The procedure involving the creation of an initialization key, using it to protect the exchange of a new link key, and running the authentication procedure with the new link key is called *pairing*.

Key generations and the authentication procedure utilize algorithms referred to as E_1, E_{21}, E_{22}, and E_3. They are all based on the SAFER+ block cipher [39]. Compared to Shared Key authentication of WEP, the Bluetooth authentication scheme is more robust [257]. It is believed that an attacker cannot recover useful information from the exchanged authentication messages when proper parameters are used.

Confidentiality and Integrity in Bluetooth

Bluetooth optionally provides confidentiality through encrypting frame payloads. Encryption is performed with a proprietary stream cipher called E_0, which is based on four parallel LFSRs. Before proceeding with encryption, devices agree on the size of the encryption key, which can vary between 8 and 128 bits. The encryption key is derived from the link key and parameters provided by the master. E_0 is initialized with the encryption key and the real-time clock of the master, which ensures that each stream produced with the same key is different. For integrity, a CRC checksum

is computed and appended to the frame payload prior to encryption as in WEP.

4.3.2 Vulnerabilities and Improvements for Bluetooth

The Bluetooth security design has been found vulnerable to a number of attacks. The most significant vulnerabilities derive from the usage of the PIN code and the lack of data authentication. By exploiting the vulnerabilities an attacker can e.g. obtain confidential data from Bluetooth-enabled mobile phones, inject viruses as well as make unauthorized phone calls and send short messages [153, 184, 241]. Not all the exploits are directly caused by the security design but rather its implementation and configuration in end-products.

The Bluetooth security is completely based on the PIN code [127, 234]. Therefore, the PIN should always be long and randomly chosen. However, the Bluetooth specification permits fixed PINs, PINs of only eight bits long, and defines a default value for the PIN code (zero). Since a method for automatically exchanging PINs has not been defined, users tend to choose short (typically four digits) and easily memorizable values [234]. When the PIN code is poorly chosen, an attacker can perform offline search for the code after eavesdropping on pairing [127, 218]. If an attacker has not been present during pairing, she can claim the link key lost and make the victim devices rerun pairing [127, 218].

Due to the weaknesses, Bluetooth SIG recommends performing pairing in a private location [75]. The pairing procedure has formally been analyzed and deduced to be secure if pairing can be performed privately [235]. As more robust and 802.11i-convergent solutions, the 802.1X framework, Diffie-Hellman-based key exchange mechanisms, and public-key protocols based on Elliptic Curve Cryptography (ECC) have been proposed for authentication and link key agreements [76, 110, 221]. The proposals specifically addressing scatternets assume that the linking devices share a secret PIN code [161, 190] or utilize the Diffie-Hellman mechanism [161]. Bluetooth SIG has not standardized entity authentication and/or key management solutions in addition to the ones described in Section 4.3.1. However, it is currently working on a new procedure for improving the security of pairing [23]. The proposed new solution is based on ECC and the Diffie-Hellman protocol.

Similarly to 802.11, a vulnerability in the entity authentication of Bluetooth is that only devices are authenticated. It has been reported that switching the owner and the Subscriber Identity Module (SIM) of a mobile phone does not always require re-establishing the Bluetooth connections of the phone [240]. In addition, it is possible

for an attacker to force a slave to roll back to an old link key by recording the original key agreement phase and then later impersonating as the master [203, 204]. This allows the attacker to replay frames protected with the old link key. As countermeasures, it is suggested that the link key is changed regularly and/or the slave is always made to supply the last challenge in the entity authentication protocol [203, 204].

An attacker is able to fake two Bluetooth devices to believe that they are directly communicating with each other by simply relaying messages between them [157]. The relaying attack is not beneficial if the devices decide to use encryption or if a proper data authentication mechanism is added to the data transfers [P9]. Vulnerabilities related to the lack of data authentication are the same as those in WEP and discussed further in [P9].

The negotiable key size is a threat to the Bluetooth encryption scheme since a malicious party can purposely make a device use a short key and thus alleviate attacking against encryption [88, 137]. Furthermore, the applications and users of Bluetooth devices are not aware of the negotiation procedure or agreed encryption key sizes.

The E_0 cipher provides significantly lower security than a 128-bit-key algorithm should [71, 95, 127, 159, 164, 165, 204]. If the cipher is used outside the Bluetooth technology and allowed to produce long key streams, it is far too weak [165]. Within the constraints of Bluetooth, a practical attack for recovering the encryption key can be performed after discovering the first 24 key stream bits of about $2^{23.8}$ frames [164]. Even though the E_0 attacks have not yet been exploited in practice, these are alerting results as they correspond to the ones that led to the complete insecurity of WEP [70]. The increasing data rates and expanding usage scenarios improve the feasibility of the attacks. The concerns have aroused discussions about replacing E_0 with a better algorithm [203, 204]. A scheme for improving the Bluetooth security including AES-based encryption and data authentication is proposed in [P9]. The scheme can be combined e.g. with the key agreement proposals of [23, 76, 110, 161, 190, 221].

Similarly to 802.11 devices, Bluetooth-enable devices can be tracked as they constantly advertise their unique addresses [127, 234, 240]. A scheme for thwarting this threat has been proposed for Bluetooth in [76]. Bluetooth SIG has discussed addressing this threat [21] but so far support for location privacy/anonymity has not been specified [24]. Preventing DoS attacks performed by tampering with the unprotected management frames requires protecting them with a data authentication mechanism.

4.4 IEEE 802.15.4

IEEE 802.15.4 [116] is a relatively new standard defining the radio and the MAC layer for a low-rate, low-power wireless network. The technology is promoted by an industry consortium called ZigBee Alliance [263]. ZigBee Alliance has also published its own specification [262], which defines protocol layers on top of the 802.15.4 MAC. The 802.15.4 technology is further described in [P2].

4.4.1 Security Design of IEEE 802.15.4

The 802.15.4 MAC layer supports three security-related modes: *unsecured*, *Access Control List* (ACL), and *secured*. The unsecured mode does not include security mechanisms. The ACL mode corresponds to non-cryptographic access filtering using lists of authorized MAC addresses.

The secured mode uses cryptography for providing confidentiality and/or data authentication through the seven security suites described in [P2]. As the schemes are based on the work of TGi, they also utilize AES in related modes of operation. Depending on the desired level of protection, the MAC layer can be configured to either provide confidentiality in the CTR suite, data authentication in the CBC-MIC suites, or both in the CCM suites. The standard does not define entity authentication or key agreement mechanisms. They must be provided by a higher protocol layer. Also, a higher layer protocol controls the confidentiality and data authentication procedures by determining when and how they are used.

The 802.15.4 MAC layer supports two types of addressing. A 64-bit device address (*extended address*) is fixed and universally unique whereas a 16-bit address (*short address*) is a temporal address allocated to a new device in the association by the device coordinating the network. The ACL entries of a device define the agreed security suites and link keys for each peer address. The 128-bit link key is directly used in the confidentiality and data authentication procedures. The same link key is only allowed to be used for one purpose. Each ACL entry contains sequence counters that are used as inputs to the cryptographic procedures and optionally for freshness protection. The default entry is used when a device operating in a network is not explicitly listed but the secured mode is still desired. The default key can be used as a group key.

4.4.2 Vulnerabilities of IEEE 802.15.4

Even though 802.15.4 is a fairly new technology, it has already been identified to contain shortcomings, including the basic weakness of WLANs: the default operating mode is unsecured. Other default settings are problematic as well. The usage of default ACL entries exposes 802.15.4 networks to insider attacks and compromising a single device enables access to the transmissions of a large number of other devices [213].

The CTR suite of 802.15.4 provides only confidentiality, not data authentication or integrity, as also pointed out in the standard. This results in various problems and the usage of the CTR suite is not recommended [167, 213]. Flipping a ciphertext bit flips the corresponding plaintext bit, which can be used for making chosen modification to frames [20, 167]. Furthermore, repeating a key stream breaks the whole security of the CTR mode in the same way as for WEP. Therefore, the CTR suite, as well as the CCM suites, should always be used with well-established key management mechanisms [167, 213]. Using the same key in two or more ACL entries makes key stream repeats very likely, which, on the other hand, makes group keying and replay protection in group communications problematic [213].

The unprotected acknowledgement frames can be used for mounting DoS attacks by injecting them to the network [213]. Thus, an acknowledgement never provides a proof of successful transmission and should not be trusted. Because of the lack of data authentication, the freshness protection of the CTR suite can be exploited for DoS attacks [213].

Correspondingly to the other WLANs, the 802.15.4 technology is susceptible to location privacy attacks. However, some level of anonymity and/or location privacy can be achieved in an established network if short addresses are used.

4.4.3 Security Design of ZigBee Specification

The ZigBee specification [262] makes use of the 802.15.4 MAC security by building on it and slightly modifying it. It defines three additional protocol layers above the 802.15.4 MAC for providing routing and network management beyond single 802.15.4 links. Mechanisms for key management and entity authentication have been specified. Furthermore, the specification includes profiles that specify parameterizations for different application scenarios.

The ZigBee specification uses separate keys for pairwise and network-wide com-

munications. The keys can be either distributed dynamically or pre-installed into devices. Protected key distribution requires that the devices share a key with a Trust Center (TC), which is a device coordinating the network. Entity authentication is carried out with a four-message challenge-response scheme called Symmetric-Key Key Establishment (SKKE). The computations utilize AES with 128-bit keys.

Instead of supporting the separate MIC and CTR suites of 802.15.4, the ZigBee specification specifies a new mode of operation called CCM*. Also CCM* uses AES but in addition to the standard CCM operation, it offers encryption-only and MIC-only configurations, corresponding to the CTR and MIC suites of 802.15.4. The processing in these configurations is equal to CCM but either the encrypted data or the MIC are discarded. The new mode was defined in order to enable the use of a single key throughout the CCM* configurations as well as on the different layers of the ZigBee protocol stack. IEEE is discussing about adopting the CCM* mode into the next release of the 802.15.4 standard [195].

Compared to the 802.15.4 standard [116], the ZigBee specification [262] has improved the resistance against replay attacks as the freshness of outgoing frames is always provided. This includes the MIC-only configuration, which does not support freshness protection in 802.15.4. On the contrary, in spite of the new CCM* mode, the encryption-only configuration of the ZigBee specification is still vulnerable to similar attacks as the CTR suite of 802.15.4. However, since methods for key distribution and agreement as well as entity authentication have been defined, key stream repeats are potentially rarer. In addition, in contrast to 802.15.4, outgoing and incoming frames use separate counters. According to the author's knowledge, security analyses of the ZigBee specification have not been published. Therefore, its analysis is an interesting topic for the future research.

4.5 Other WLAN Standards

IEEE has commercially been the most successful WLAN standardization organization. Therefore, the new IEEE 802.15.3 technology [115] is predicted to achieve popularity. It is specifically aimed at portable consumer digital imaging and multimedia applications [122]. HIgh PErformance Radio LAN type 1 (HIPERLAN/1) and type 2 (HIPERLAN/2) [56] have been significant technologies for the development of WLANs. However, they have failed in market penetration and currently vanishing because of the dominance of IEEE 802.11.

4.5.1 IEEE 802.15.3

The 802.15.3 standard [115] specifies the physical layer and the MAC layer for high-rate, low-power, and low-cost wireless multimedia communications. Similarly to the Bluetooth specification, it refers to networks as piconets. The device controlling the piconet is called PicoNet Coordinator (PNC). Key features are fast connection time, QoS, and ad hoc networking.

A piconet can operate in unsecured and secured modes. Cryptographic procedures are not invoked in the unsecured mode. A device joining a secured piconet is required to establish a secure membership with PNC, which means that they have to end up sharing a pairwise key. Later on, the device can establish secure peer-to-peer relationships with other piconet devices. As the standard does not specify mechanisms for entity authentication or key agreement, the membership and relationship establishments are left to the upper protocol layers. A scheme based on the Diffie-Hellman protocol and public-key certificates is proposed for the establishments in [229].

After a membership or a relationship has been established, the 802.15.3 technology supports pairwise and group keys. In addition, separate link keys are used for protecting management and data frames. Piconet management includes transporting link keys between devices. Link keys are utilized for providing the confidentiality and data authentication of frames as in the CCM suites of 802.15.4 and in CCMP. A possibility for forcing a device to use an old link key has been identified in [203].

4.5.2 ETSI HIPERLANs

The HIPERLAN networks have been developed by the European Telecommunications Standards Institute (ETSI) project Broadband Radio Access Networks (BRAN) [56]. HIPERLAN/1 was the first of the modern WLAN standards. It was aimed at replacing or extending wired LANs and supporting ad hoc networking [106]. The main design objective of HIPERLAN/2 has been high-speed access to broadband core networks, such as IP and Asynchronous Transfer Mode (ATM), with emphasis on QoS [106].

HIPERLAN/1 only specifies an optional confidentiality scheme [53]. It attempts to provide confidentiality in the same ways as WEP. Frames are encrypted with a stream cipher, which is seeded with an encryption key and an IV, both 30 bits long. Instead of RC4, HIPERLAN/1 uses a secret PRNG, which is only made available to implementers through a special agreement. Since mechanisms for key management,

IV management, and data authentication have not been defined, HIPERLAN/1 is prone to the same attacks as WEP. The size of the key allows effective brute force attacks for recovering the key. The secrecy of the PRNG violates good security design practices.

The security design of HIPERLAN/2 has been improved from HIPERLAN/1. It provides methods for key agreement, entity authentication, and confidentiality [54, 55]. The Diffie-Hellman key agreement is used for exchanging an encryption key in the beginning of a connection establishment. The correctness of the agreed key is verified during entity authentication, which is either based on secret keys or performed using RSA signatures. The distribution of authentication keys has not been defined. The secret key authentication as well as encryption key generations utilize HMAC-MD5. Encryption is performed with a key stream which is generated from an IV using DES or 3DES in the Output FeedBack (OFB) mode [48]. The IV material is periodically refreshed by the device controlling the network.

4.6 Summary

WLAN security is intensively being researched in academic, industry, and standardization communities. It has been acknowledged that cryptographic mechanisms are required for protecting WLANs at the MAC layer. The research and development of WLAN security has concentrated on commercially significant standard WLAN technologies. Academic research has particularly focused on identifying vulnerabilities in established WLAN standards and providing general recommendations for repairing them. Currently, the employed improvements are developed in co-operation between academic researchers, industry consortiums, and standardization bodies.

Table 2 summarizes the main security characteristics of the standard WLAN technologies reviewed in this chapter. The provided security levels of the technologies are compared relative to each other. Nevertheless, the reported levels are highly dependent on the chosen parameterizations in network installations. 802.11i in the RSN configuration is considered to provide the highest security of the technologies. The other configurations of 802.11i correspond to 802.11. 802.15.3, 802.15.4, and the ZigBee specification can achieve the same level as RSN but this depends more on the choices the application designer makes. The ZigBee specification appears to be the most complete of those three. 802.11 and HIPERLAN/1 contain significant vulnerabilities and Bluetooth can also be attacked effectively. The security of HIPERLAN/2 seems to be better than that of HIPERLAN/1 and 802.11 but it has not been examined

intensively.

With the current knowledge, it can be concluded that the newest WLAN technologies [115, 116, 118, 262] enable secure installations using their cryptographic mechanisms. This still requires choosing a proper configuration from the supported set of parameters, adopting secure mechanisms for covering the components that have been left undefined in the WLAN specifications, and developing efficient and secure implementations for the mechanisms.

Table 2. *Summary of the main security characteristics of standard WLAN technologies.*

Technology	Security level	Main algorithms	MICs supported	Entity authentication and/or key agreement	Most severe problems	Required fixes
802.11	Low	RC4 (WEP)	No	Challenge-response with secret key	Lack of integrity, key recovery	Proper key and IV management, adding data authentication
802.11i in RSN	High	AES (CCMP), RC4 (TKIP)	Yes	802.1X and 4-way handshake	DoS	Protection of management frames
Bluetooth	Medium/low	LFSR, SAFER+	No	Challenge-response with secret key	Lack of integrity, key recovery	Allowing only long PINs and encryption keys, adding data authentication
802.15.4	Medium	AES (CCM, CTR, CBC-MIC)	Yes	not specified	Missing key agreement and entity authentication, lack of integrity in CTR	Adding key agreement and entity authentication mechanisms, removing CTR suite
ZigBee	Medium	AES (CCM*)	Yes	Challenge-response with secret key	Lack of integrity in encryption-only mode	Removing encryption-only mode
802.15.3	Medium	AES (CCM)	Yes	not specified	Missing key agreement and entity authentication	Adding key agreement and entity authentication mechanisms
HIPERLAN/1	Low	Stream cipher	No	not specified	Similar to 802.11	Similar to 802.11
HIPERLAN/2	Medium/low	DES, 3DES (OFB)	No	Diffie-Hellman, RSA, secret key	Lack of integrity	Adding data authentication

5. HARDWARE ARCHITECTURES FOR WLAN CRYPTOGRAPHY

As discussed in Chapter 1, a security processing implementation has a key role in maintaining the desired level of security in wireless devices. It has been shown that cryptographic software implementations cannot achieve the performance levels required by various communication networks at reasonable costs [10, 99, 100, 103, 187, 198]. Especially in WLAN devices this so-called security processing gap can be wide [199, 200, P2, P5]. For example, the TKIP scheme was included in 802.11i because the already fielded WLAN hardware did not contain enough processing capacity for CCMP [31]. Hardware support for the cryptographic processing is evidently required. In addition to processing efficiency, hardware inherently provides better physical security than software [47, 217], which is particularly beneficial in the portable WLAN devices. It is also easier to protect separate cryptographic hardware modules than an entire processor [197].

An aspect for the constant need of more efficient cryptographic implementations derives from the accepted security design practices. New designs should always be exposed to an intensive peer analysis, which often slows down their spread and acceptance. For example, despite that higher performance public-key algorithms have been developed, the preference is still to use the conventional ones for which the security risks are better known [198]. This calls for more efficient implementations for the conventional algorithms in order to meet the constraints of new applications.

This chapter reviews the related work for the hardware implementations of this Thesis. The chapter extends the comparisons presented in the publications as new implementations have appeared since the implementations of this Thesis were originally published. The first section discusses the characteristics of different hardware technology approaches in the light of cryptographic implementations. For the cryptographic components of WLANs, the focus is on the implementation of AES and its modes of operation, 3DES, RC4, and modular exponentiation. In addition to dedicated hardware, efficient processor architectures specifically designed for cryptographic applications are examined.

5.1 Technology Approaches for Cryptographic Implementations

The term *software implementation* refers to an implementation in a general-purpose processor which does not contain instructions that are specifically included for performing the task in question. The benefit of a software implementation is the free programmability without physical changes in the hardware. However, it is always a trade-off between reconfigurability, area, power, performance, as well as design and manufacturing costs. A software implementation contains overhead due to the instruction fetch and decode, the memory access, and possibly due to an unsuitable instruction set and word size [146].

Compared to other approaches, superior performance with the lowest power, the smallest area, and low cost in high volumes can be achieved with an Application Specific Integrated Circuit (ASIC) dedicated for a specific task. The drawbacks of ASICs are long development times, high Non-Recurring Engineering (NRE) costs, and time-consuming and costly upgrading.

High performance with reconfigurability and short development times are achieved with FPGAs. Previously FPGAs have mainly been utilized for prototyping and as glue-logic. However, according to the recent trend, the technologies are emerging as System-on-Chip (SoC) platforms for implementing complete, embedded applications [248]. The drawback of FPGAs considering embedded WLAN devices is their high power consumption [146, 151]. Despite that the production costs in high volumes are also often regarded as a drawback, FPGAs can turn out cost-effective if upgrades are required. Upgrading has been a continuing trend in WLANs, especially in their security designs. FPGAs can also be converted into ASIC-like implementations, in which the configuration is fixed [8, 255]. The costs, energy consumption, and performance of the fixed implementations are better than those of the original FPGAs.

An effective performance-reconfigurability trade-off between ASICs and FPGAs can be achieved with Application Specific Instruction set Processors (ASIP) [99, 146, 191]. An ASIP is a programmable processor that has been tailored to support a specific application domain or task. It can still be general-purpose or its applicability can be limited. The performance of an ASIP is lower than that of ASICs, sometimes also lower than that of FPGAs. However, the area and power consumption are lower than those of FPGAs and reconfigurability higher than that of ASICs.

The potential benefits of reconfigurable hardware platforms, to which FPGAs and ASIPs are considered to belong, for cryptographic applications can be summarized

as follows [51, 248]:

- *Algorithm-agility*: Cryptographic algorithms can be switched during the operation of the target application. For example, 802.11i supports multiple authentication protocols as well as data authentication and confidentiality schemes. Whereas algorithm agility is costly with traditional hardware, reconfigurable platforms can be reprogrammed on-the-fly.
- *Algorithm upload*: Fielded devices can be upgraded with new algorithms and protocols, even remotely.
- *Architecture-efficiency*: The hardware architecture can be optimized for a specific set of parameters in the design time, still maintaining the possibility to change the parameterization. For example, constant multiplications are more efficient than general multiplications in *Galois Fields* (GF) [51].
- *Resource-efficiency*: The majority of security protocols are hybrid protocols consisting of several cryptographic algorithms. For example, 802.11i can utilize public-key and secret-key algorithms. Since the algorithms are not required simultaneously, hardware resources can be shared through run-time reconfiguration.
- *Algorithm modification*: Some applications can require modifying a standardized algorithm, e.g. for using proprietary permutations. Furthermore, in many cases the mode of operation needs to be changed according to the application.
- *Performance*: Although typically slower than ASICs, reconfigurable implementations are potentially significantly faster than software implementations.
- *Cost-efficiency*: The NRE costs for a reconfigurable implementation of an algorithm are much lower than for an ASIC implementation. Reconfigurability supports low-cost upgrading.

Despite the benefits, FPGAs are currently not mature enough for low-power WLAN devices as discussed in Section 1.1. Therefore, in this Thesis they are mainly utilized for prototyping purposes, in order to facilitate the verification of the implementations in a real, physical platform and in a real application environment, TUTWLAN. FPGAs are also used for comparing different hardware design choices [P1, P2, P9]. The high-level architectures are directly applicable in ASIC implementations. Moreover, since most of the FPGA implementations contain very few and easily separable

technology-specific components (e.g. memories), they can effortlessly be ported into another technology. On the contrary, the developed TTA ASIPs are directly suited for embedded WLAN devices.

5.2 Hardware Implementation of Block Ciphers

The basic techniques for implementing a block cipher with rounds, such as AES and 3DES, are iterated, pipelined, and loop-unrolled architectures [51, 259]. The more advanced structures include partial pipelining and sub-pipelining combined with the basic techniques [51, 228, 260]. The architectures are depicted in Fig. 16.

The iterated architecture leads to the smallest implementations as it consists of one round component which is fed with its output until the required number of rounds is performed. One round is executed per clock cycle and the implementation can only process one block at a time. The throughput is proportional to the cycle time and to the number of rounds.

The pipelined architecture contains all the rounds as separate components with registers in between. As a result, it is the fastest (in terms of throughput) and the largest of the basic structures. After the pipeline is filled, an encrypted block is output at every clock cycle. The throughput depends on the cycle time and the level of pipeline utilization.

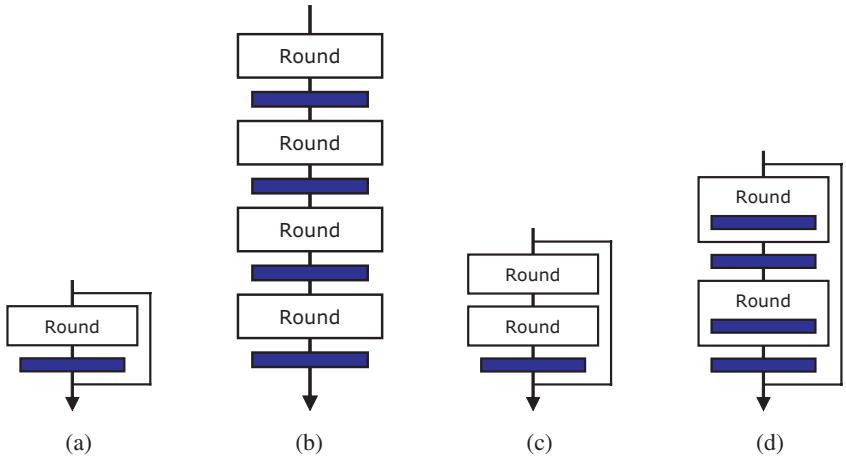


Fig. 16. *Hardware architectures for block ciphers: (a) iterated, (b) pipelined, (c) loop-unrolled, and (d) the combination of partial pipelining and sub-pipelining. The full cipher consists of four rounds.*

The loop-unrolled architectures perform two or more rounds per clock cycle and the execution of the cipher is iterated. The unfolding factor affects the area and the register-to-register delay. Even though the delay of the unrolled loop is larger than that of a single round, the total throughput in an unrolled implementation can be higher than in an iterated implementation [P1]. The decrease in the cycle count increases the throughput more than the increase in the cycle time reduces it. However, the system clock sets the limit to the maximum delay and thus to the unrolling factor. In a pipelined architecture unrolling can only decrease the latency for outputting the first block. Due to the increase in the register-to-register delay, the throughput is decreased. Therefore, unrolling in a fully pipelined architecture is reasonable when less registers are desired or when the (fixed) system clock allows increasing the combinatorial logic delay.

In sub-pipelining registers are placed inside the round component in order to increase the maximum clock frequency. In the partial pipelining scheme the pipeline contains e.g. the half of the rounds with registers in between. Encryption is performed by processing a data block twice through the pipeline.

Due to the feedback loop of the utilized modes of operation, pipelining is not beneficial in most WLAN security schemes [P2]. The large size makes specifically the fully pipelined structures unattractive for low-cost and low-power implementations. In the feedback modes adding registers is only justified when a decrease in the combinatorial logic delay is required. Otherwise iterated and/or loop-unrolled architectures should be used. In some cases additional registers can decrease the total resource consumption in FPGA implementations because routing becomes easier. However, such technology-specific optimizations are not considered in this Thesis.

5.2.1 AES Implementations

In addition to the architectural choices, the design of AES enables a large number of algorithm-specific hardware trade-offs. The trade-offs consist of choosing between memory-based Look-Up-Tables (LUT) and combinatorial logic, decreasing the amount of parallelism, transferring computations into another arithmetic domain, choosing between the precomputation and the on-the-fly computation of *roundkeys*, and sharing resources between encryption, decryption, and roundkey generation data paths. These aspects as well as the high-level architectural choices, the effect of modes of operation, and parallelism are discussed in [P1–P3, P5].

Because of the public standardization process and the popularity of AES as well as

the generally increased interest in information security, a large number of hardware implementations have been published for AES in the academic literature. Table 3 lists the AES implementations that are considered to have achieved the most significant results along with the implementations of this Thesis. Moreover, the implementations published before [P1] and the implementations supporting the modes of operation related WLANs have been included. The publications which contain insufficient information and the FPGA implementations that did not fit into their target devices have been excluded. The pipelined implementations are grayed in the table and considered poorly-suited for WLAN devices.

In addition to the achieved results, the table contains comprehensive information for the design trade-offs, approaches, and functionalities employed in the hardware implementations of AES. Therefore, it enables extensive comparisons. Besides the purposes of this Thesis, the author believes that the table is valuable as a reference for designers evaluating and developing AES implementations. A narrower study concentrating on FPGA implementations has previously been presented in [130].

The marking 'n/a' in a table cell means that the value has not been reported or it is not clear if the method in question has been used. The columns of the table contain the following information:

- The first three columns report the reference *publication*, the *year* of publication, and the used *technology*, i.e. an FPGA device or ASIC process. The rows of the table are organized according to the publishing time. For FPGA devices the name is given as it has been reported in the publication. The FPGA technologies can be differentiated according to the column *logic cells*.

- *Architecture* specifies the high-level architecture, which is either iterated (IT), loop-unrolled (LU), pipelined (PI), partially pipelined (PP), or sub-pipelined (SP). LU-X refers to a structure which contains X unrolled rounds. PI-X is a fully pipelined architecture containing X pipeline stages. PP-X is a partially pipelined architecture with X rounds. SP-X-Y is an architecture which consists of X pipelined rounds, each containing Y sub-pipeline stages.

- *Data path width* reports how many bits of an AES block are processed in parallel. The value *mix* means that the amount of parallelism varies through the different phases of the algorithm.

- *Modes* presents which modes of operation an implementation supports. All the implementations support ECB processing in forward direction, i.e. ECB

encryption. If the column *decryption* is checked, the implementation supports the decryption functionality for its modes of operation.

- *Keys* defines if an implementation supports all the AES keys sizes, i.e. 128, 192, and 256 bits. All the implementations support 128-bit keys. The columns *on-the-fly* and *precomputed* specify which of these methods an implementation uses for its roundkey generation [P2]. If neither of the two is checked, the implementation does not contain the roundkey generation functionality.

- *Resource sharing* reports if an implementation shares hardware resources between the ECB encryption and decryption data paths (ED) and/or between its data path and roundkey generation (DK).

- *Substitution method* specifies which technique is used for the implementation of the SubBytes operation of AES. *S-box* refers to that the AES S-box and/or its inverse are directly implemented as specified in the AES standard [61]. *GF inverse* means that the S-box is divided into two phases, a GF inversion and an affine transformation, which allows sharing the GF inversion between the ECB encryption and decryption data paths [P1]. *T-box* refers to an implementation in which the SubBytes and MixColumns operations of AES [61] are combined into large LUTs [P2, P5]. In the *composite field* technique the GF inversion of SubBytes is computed with the help of another arithmetic domain. The columns *ROM* and *logic* specify whether the previous techniques are implemented as memory-based lookups or as combinatorial logic, respectively.

- *Logic cells* and *memory* report the resource consumption of the implementations. The sizes of ASICs are measured in kilogates (kgates) and this value includes the possibly utilized memory components. The general-purpose programmable resources in Altera FPGAs are called Logic Elements (LE) [9] and in Xilinx FPGAs slices [256]. A LE consists of a 4-input LUT and a register and a slice typically roughly corresponds to two LEs. The column *memory* presents the amount of dedicated memory components utilized in the FPGA implementations. As the type of the memory components as well as the Xilinx slices can vary through FPGA technologies, the reader is referred to the data sheets of the specific FPGA devices for more detailed information.

- *Latency* presents the number of clock cycles required for encrypting a block of data. The value does not include the additional latency possibly caused by roundkey generation. In addition, the latency has not been clearly reported in

all the publications and thus the value reported here may not always be fully exact.

- *Clock* reports the maximum clock frequency of an implementation.
- *Throughput* specifies the throughput of the implementation for the reported clock frequency with 128-bit key and block sizes. The value does not include the additional latency possibly caused by roundkey generation. The throughputs for the implementations employing any form of pipelining assume that the pipeline is fully utilized. For comparison, the throughput T (Mbit/s) of a pipelined implementation for feedback modes of operation can be estimated as

$$T = \frac{f}{L} \cdot B, \tag{6}$$

in which f is the maximum clock frequency (MHz), L latency (cycles/block), and B the block length of the cipher in bits (128 bits for AES).

AES Cores

Most of the publications before [P1] have been targeted at comparing the final candidate algorithms during the last phase of the AES standardization process [35, 41, 49, 67, 73, 74, 111]. They aim at implementing the data path of the candidates in the same way for enabling fair comparison. Roundkey generation is included in [41, 111, 150, 169, 179] and decryption functionality in [67, 68, 111, 169].

Of these earliest publications, [49] presents a comprehensive study of AES encryption data path implementations with different architectural choices but lacks decryption and roundkey generation. Ref. [68] describes compact AES data path implementations in FPGAs, also with decryption functionality but still excludes the roundkey generation. References [111, 169] are the only ones clearly combining encryption, decryption, and roundkey generation. However, they are both very large implementations. Therefore, [P1] has been the first to report a 128-bit-key AES implementation suited for embedded WLAN devices. In addition, it has been the first to utilize resource sharing between the encryption and decryption data paths as well as to implement iterated on-the-fly roundkey generation in both directions. In [169] the roundkeys are generated and stored in forward direction also in decryption.

As it has later turned out, only the forward functionality of AES with 128-bit keys is required in most WLAN modes of operation. Suitable implementations for this

Table 3. *Comparison of AES hardware implementations (pipelined implementations are shown with gray background).*

Publication	Year of publication	Technology	Architecture	Data path width (bits)	Modes				Keys			Resource sharing		Substitution method				Logic cells	Memory	Latency (cycles/block)	Clock (MHz)	Throughput (Mbit/s)	Notes
					Decryption	CTR	CBC/CBC-MIC	CCM	All key sizes	On-the-fly	Precomputed	ED	DK	S-box	GF inverse	T-box	Composite field	ROM	Logic					
Dandalis [41]	2000	Xilinx Virtex	IT	128						x				x				x		5673 slices	n/a	n/a	n/a	353	
Elbirt [49]	2000	XCV1000-BG560-4	IT	128										x					x	3528 slices		11	25	294	
			LU-2	128										x					x	5302 slices		6	14	300	
			LU-5	128										x					x	10286 slices		3	6	237	
			PP-2	128										x					x	5281 slices		11	24	546	
			PP-5	128										x					x	10533 slices		11	20	1166	
			SP-1-1	128										x					x	3061 slices		21	40	492	
			SP-2-1	128										x					x	4871 slices		21	39	949	
			SP-5-1	128										x					x	10992 slices		21	32	1938	
Fischer [67]	2000	FLEX10K-130EQC240-1	IT	128										x				x		1585 LEs	24 EABs	10	n/a	233	
			IT	64										x				x		1604 LEs	12 EABs	20	n/a	122	
			IT	16										x				x		1673 LEs	3 EABs	80	n/a	32	
			IT	128	x									x				x		3348 LEs	24 EABs	10	n/a	179	
			IT	64	x									x				x		3320 LEs	12 EABs	20	n/a	94	
			IT	16	x									x				x		3324 LEs	3 EABs	80	n/a	24	
Gaj [73]	2000	XCV1000-BG560-6	IT	128	n/a						n/a				n/a			n/a		2902 slices	n/a	10	26	332	
Ichicawa [111]	2000	0.35 µm	LU-10	128	x					x				x					x	613 kgates		1	15	1950	
Mroczkowski [179]	2000	EPF10K250A GC599-1	IT	128						x				x				x		1032 LEs	20 EABs	21	45	268	(1)
Chodowiec [35]	2001 Feb	XCV1000-BG560-6	SP-1-8	128	n/a						n/a				n/a			n/a		2057 slices	8 BRAMs	80	99	1265	
Gaj [74]	2001 Apr	XCV1000-BG560-6	IT	128	n/a						n/a				n/a			n/a		2507 slices	n/a	n/a	n/a	414	
Fischer [68]	2001 May	APEX1K400-1	IT	128										x				x		1267 LEs	50 ESBs	11	n/a	964	
			IT	128	x									x				x		2493 LEs	50 ESBs	11	n/a	612	(2)
		FLEX10K-E200-1	IT	128										x				x		1265 LEs	24 EABs	11	n/a	570	
			IT	128	x									x				x		2530 LEs	24 EABs	11	n/a	451	

(Table 3 continued)

Publication	Year of publication	Technology	Architecture	Data path width (bits)	Modes: Decryption	Modes: CTR	Modes: CBC/CBC-MIC	Modes: CCM	Keys: All key sizes	Keys: On-the-fly	Keys: Precomputed	Resource sharing: ED	Resource sharing: DK	Substitution method: S-box	Substitution method: GF inverse	Substitution method: T-box	Substitution method: Composite field	Substitution method: ROM	Substitution method: Logic	Logic cells	Memory	Latency (cycles/block)	Clock (MHz)	Throughput (Mbit/s)	Notes
Fischer [68]	2001 May	ACEX1K100-1	IT	64										x				x		1461 LEs	12 EABs	22	n/a	282	
			IT	64	x									x				x		2923 LEs	12 EABs	22	n/a	212	
		APEX1K400-1	IT	128														x		529 LEs	86 ESBs	11	n/a	750	(2)
			IT	128	x											x		x		845 LEs	86 ESBs	11	n/a	750	
		ACEX1K50-1	IT	32												x		x		824 LEs	10 EABs	44	n/a	115	
			IT	32	x											x		x		1213 LEs	10 EABs	44	n/a	115	
Kuo [150]	2001 May	0.18 µm	IT	256					x	x				x				x		173 kgates		10	100	1280	(3)
McLoone [169]	2001 May	XCV3200E-CG1156-8	PI-10	128	x					x								x		7576 slices	102 BRAMs	10	25	3239	(4)
		XCV812E-BG560-8	PI-10	128						x				x				x		2222 slices	100 BRAMs	10	54	6956	
This Thesis [P1]	**2001 Jun**	**XCV800-FG676-6**	**IT**	**128**					**x**				**x**					**x**		**2272 slices**		**11**	**31**	**364**	
			IT	**128**	**x**				**x**		**x**				**x**					**3267 slices**		**11**	**21**	**246**	
Satoh [215]	2001 Dec	0.11 µm	IT	32	x				x		x	x					x		x	5.40 kgates		54	131	311	
			IT	32	x				x		x	x					x		x	6.29 kgates		44	138	400	
			IT	64	x				x		x	x					x		x	8.00 kgates		32	137	549	(5)
			IT	64	x				x		x	x					x		x	8.84 kgates		22	137	798	
			IT	128	x				x		x	x					x		x	12.5 kgates		11	145	1691	
McLoone [170]	2003	XCV600E-BG432-8	IT	128	x		x							x					x	4681 slices	20 BRAMs	n/a	n/a	310	(6)
Saggese [210]	2003	XCV2000E-BG560-8	IT	128					x					x					x	446 slices	10 BRAMs	10	79	1000	
			SP-1-5	128					x					x					x	648 slices	10 BRAMs	50	142	1820	
Chodowiec [34]	2003	XC2S30-6	IT	32	x														x	222 slices	3 BRAMs	46	60	166	
Standaert [224]	2003	Virtex 3200E	SP-1-8	128						x							x		x	1767 slices		82	167	2085	
			PI-21	128						x				x					x	2784 slices	10 BRAMs	21	92	11776	
Hodjat [108]	2003	0.18 µm	PI-41	128						x			x					x		473 kgates		41	606	77600	
Rouvroy [209]	2004	XC2V40-6	IT	32	x						x	x				x				146 slices	3 BRAMs	46	123	358	

(Table 3 continued)

Publication	Year of publication	Technology	Architecture	Data path width (bits)	Modes			Keys			Resource sharing		Substitution method					Logic cells	Memory	Latency (cycles/block)	Clock (MHz)	Throughput (Mbit/s)	Notes
					Decryption	CTR	CBC/CBC-MIC	CCM	All key sizes	On-the-fly	Precomputed	ED	DK	S-box	GF inverse	T-box	Composite field	ROM	Logic					
Hodjat [107]	2004	0.18 µm	IT	128	x	x	x	x		x				x				x		73.0 kgates		11	295	3430	
Gurkaynak [86]	2004	0.25 µm	IT	128	x		x		x	x				x					x	119 kgates		11	166	2120	(6)
Pramstaller [197]	2004	0.6 µm	IT	128	x		x			x		x	x				x		x	15.9 kgates		35	50	183	
		XCV1000E-BG560-8	IT	mix	x		x		x	x		x	x		x			x		1125 slices		n/a	161	215	
Jang [128]	2004	Virtex E	IT	128	x			x			x			n/a				n/a		3750 slices	n/a	n/a	50	232	(7)
Morioka [177]	2004	0.13 µm	IT	128	x		x									x			x	282 kgates		10	885	11300	
			IT	128	x		x							x					x	62.0 kgates		10	699	8900	
Feldhofer [62]	2004	0.35 µm	IT	8						x				x	x				x	3.60 kgates		1016	100	13	
Zambreno [258]	2004	XC2V4000	IT	128						x				x				x		387 slices	10 BRAMs	n/a	110	1410	
		LU-5		128						x				x				x		1532 slices	50 BRAMs	n/a	72	4640	
Kotturi [144]	2005	XC2VP70-7	PI-60	128						x				x				x		5408 slices	200 BRAMs	60	233	29770	
This Thesis [P9]	**2005**	**EPXA10-F1020C2**	**IT**	**128**						**x**				**x**				**x**		**1246 LEs**	**20 ESBs**	**11**	**44**	**507**	
			IT	**128**	**x**	**x**	**x**	**x**		**x**				**x**				**x**		**3527 LEs**	**28 ESBs**	**26**	**44**	**214**	(7)
This Thesis [P2]	**2005**	**EP1C4-F324C6**	**IT**	**128**						**x**				**x**				**x**		**1368 LEs**	**10 M4Ks**	**10**	**110**	**1408**	
			IT	**32**							**x**		**x**	**x**				**x**		**512 LEs**	**7 M4Ks**	**55**	**116**	**270**	
			IT	**32**	**x**	**x**	**x**	**x**			**x**		**x**	**x**				**x**		**1434 LEs**	**11 M4Ks**	**112**	**78**	**90**	(7)
Good [80]	2005	XC3S2000-5	PI-70	128	x					x		n/a					x		x	17425 slices		70	196	25107	
		XC2S15-6	IT	8	x					n/a	n/a	n/a	n/a				x		x	124 slices	2 BRAMs	n/a	67	2	
Feldhofer [63]	2005	0.35 µm	IT	8	x					x		x					x		x	3.40 kgates		1032	80	10	
Lopez-Trejo [162]	2005	XC3S4000-4	IT	128	x	x	x	x		x				x				x		2154 slices	106 BRAMs	12	100	1051	
Hämäläinen [96]	2006	0.13 µm	IT	8						x							x		x	3.10 kgates		160	152	121	

(1) The paper presents also a separate decryption implementation. (2) Presumably, the roundkey generation has not been implemented even though it has been discussed. (3) Supports all Rijndael block sizes. (4) Other key sizes supported via resynthesis for each key size. (5) Only area optimized results presented here. (6) Supports also OFB and CFB modes of operation. (7) Throughput for CCM.

at the time of publishing [P1] have been reported in [41, 150, 179]. Compared to the encryption-only implementation of [P1], the size of [41] is significantly larger. Even though [P1] and [150] have been implemented in different hardware technologies and direct comparison is not possible, it can be determined that [150] results in significantly larger size because of the support for all the Rijndael block and key sizes. Comparing [P1] with [179] is difficult because of the utilization of memory components and different FPGA technologies. The high-level architectures of [P1] and [179] are similar but the throughput of [P1] is higher. The throughput of the smallest pipelined implementation [35] is lower than that of both the implementations of [P1] in a feedback mode of operation. The other pipelined implementations are significantly larger.

After [P1], implementations tuning AES for compact size and low power [34, 62, 63, 80, 197, 209, 210, 215, 258] as well as improving throughput [80, 108, 144, 224, 258] have appeared. For the compact size and low power, the implementations have decreased the data path width and utilized the composite field techniques. For higher throughput, the number of pipeline stages has been increased and larger LUTs have been utilized. A compact and low-power 8-bit AES implementation that utilizes the byte permutation structures of [P3] is presented in [96].

AES Implementations with Modes of Operation

The AES architectures [P2, P9] concentrate on supporting the WLAN modes of operation, specifically CCM. The references implementing other modes of operation, including the separate components of CCM, are [86, 170, 177]. Even though different hardware technologies have been used, [86] requires a considerably higher amount of resources than [P2, P9] as it contains separate AES cores for encryption and decryption. Compared to [P2], the data path of [86] is four times wider. The implementation of [P2] is significantly smaller than [170], still yielding adequate throughput.

Comparing with [177] is difficult due to the different technologies. In [177] the authors have concentrated on optimizing the SubBytes operation of AES for speed and implemented a very high-speed iterated architecture at the cost of hardware size. The paper reports that their method results in higher resource consumption than a LUT-based approach, which has been used in [P2, P9]. The data path in [177] is four times wider than in [P2]. Earlier the authors of [177] have reported AES hardware architectures well-suited for WLAN devices [215].

According to the author's knowledge, the only detailed CCM implementations, be-

sides in this Thesis, have been published in [107, 128, 162]. For the CCM mode, the architectures of [107, 162] include two AES cores which encrypt and compute the MIC in parallel. Both result in high throughput at the cost of hardware area. Ref. [128] presents a combined implementation of the CCM and OCB modes and includes an AES core that is shared between the two modes. The core has separate entities for the encryption and decryption functionalities of AES. When only the encryption functionality and CCM are needed, the design unnecessarily consumes resources due to the OCB capability.

5.2.2 3DES Implementations

Compared to AES, 3DES provides less parallelism and possibilities for algorithm-specific trade-offs. 3DES processes data as 64-bit blocks and contains eight different S-boxes for performing a substitution operation during each of its 48 rounds [18]. The encryption and decryption data paths are similar, only the order of roundkeys is reversed. The roundkey generation does not contain high-level components that could be shared with the encryption data path.

As DES is almost 30 years old, its implementation has been studied in a number of publications. However, very few have included results for 3DES. The earliest DES hardware studies, such as [44, 245], concentrated on the design of key search machines for brute-force breaking the algorithm. Later on the majority of DES implementations has been based on pipelined structures as the goal has been to maximize the throughput [35, 135, 138, 139, 168, 188, 208, 216, 232, 246]. Along with the 3DES implementations of this Thesis, Table 4 lists selected implementations of DES and 3DES. The implementations containing any form of pipelining are shown with gray background. For those the feedback throughput can be estimated according to Eq. 6 by setting $B = 64$. The included columns correspond to those of Table 3. None of the publications has reported having utilized dedicated memory blocks in FPGAs.

Prior to [P1], compact DES implementations suited for WLAN devices in feedback modes have been presented in [131, 156, 168, 249]. However, none of them reports results for 3DES and direct size and throughput comparisons with the 3DES implementations of [P1] are not possible. Of those reference publications, the best results have been achieved in [131]. It can be estimated that by processing with the fastest non-pipelined DES implementation of [131] (LU-4) three times in a row, the throughput is lower than that of the LU-2 implementation in [P1]. The iterated 3DES implementations in [P1] use similar architectural approaches as [131] has used for DES.

Before [P1] the hardware implementation of 3DES has been studied in [35, 154] of the references. The goal of [154] has been a very high-speed iterated ASIC implementation for ATM networks. The high throughput has been achieved by carefully

Table 4. *Comparison of DES/3DES hardware implementations (pipelined implementations are shown with gray background).*

Publication	Year	Technology	Archit.	Logic cells	Latency (cycles/ block)	Clock (MHz)	Throughput (Mbit/s)		Notes
							DES	3DES	
Leonard [156]	1997	XC4025-4	IT	938 slices	16	5	20		
			IT	640 slices	16	6	24		(1)
Kean [135]	1998	XC6216	SP-1-6	n/a	96	23	57		
Kaps [131]	1998	XC4008E-3-PG223	IT	262 CLBs	16	25	99		(2)
		XC4013E-3-PG223	LU-2	443 CLBs	8	19	148		
		XC4028EX-3-PG299	LU-4	722 CLBs	4	12	185		
		XC4028EX-3-PG299	PP-n/a	741 CLBs	16	25	403		
Wong [249]	1998	XC4020E	IT	438 CLBs	24	10	27		
I. Kim [139]	1999	0.35 µm	PI-n/a	n/a	n/a	88	5600		
Wilcox [246]	1999	0.6 µm	PI-n/a	n/a		n/a	9280		
Patterson [188]	2000	XCV150-6	PI-35	1584 slices	35	168	10752		(1)
Leitold [154]	2000	0.6 µm	IT	32000 trans.	108	250		155	
Trimberger [232]	2000	XCV300E-8	PI-48	4216 LUTs 5573 reg.	48	189	12000		
McLoone [168]	2000	XCV1000-4BG560	IT	500 slices	16	n/a	150		
			PI-16	6446 slices	16	60	3873		
Chodowiec [35]	2001 Feb	XCV-1000BG560-6	IT	356 slices	48	46		62	
			SP-1-4	375 slices	192	102		135	
			PI-144	12288 slices	144	116		7469	
This Thesis [P1]	**2001 Jun**	**Virtex-V800FG676-6**	**IT**	**740 slices**	**51**	**36**		**45**	
			LU-2	**833 slices**	**27**	**29**		**69**	
			PI-24	**6689 slices**	**24**	**46**		**2912**	
Rouvroy [208]	2003	XCV2V1000-6	PI-37	4255 LUTs 4128 reg.	37	333	21300		
Rouvroy [207]	2003	XCV2V1000-5	IT	189 slices	20	274	974		(3)
			IT	604 slices	58	258		917	(4)
Satoh [214]	2003	0.13 µm	IT	5504 gates	48	251		334	(5)
			LU-2	6917 gates	24	154		410	
			LU-4	9688 gates	12	105		560	
		XCV1000E-8	IT	668 slices	48	103		137	
			LU-2	828 slices	24	90		239	
			LU-4	1122 slices	12	54		290	
Kitsos [140]	2003	V1600EBG560	PI-48	14142 slices	48	115		7360	
		V400EFG676	PP-16	4715 slices	48	115		2450	
		V200EBG352	IT	405 slices	48	91		121	
H. Kim [138]	2004	n/a	PP-4	732 slices	48	n/a		267	(6)
Schaffer [216]	2004	0.6 µm	PI-16	123104 trans.	16	110		7040	

trans. = transistor, reg. = register

(1) Key-specific implementation, changing the key requires reconfiguration. (2) In these FPGAs a Configurable Logic Block (CLB) is equal to a slice. (3) Throughput for non-feedback modes, outputs at every 18th cycle. (4) Consists of three DES cores, throughput for non-feedback modes, outputs at every 18th cycle. (5) Area-optimized results. (6) Presumably BRAMs used but the amount not reported.

tuning for high clock frequency. The cost has been the increase in the cycle count for processing a block of data. The iterated implementation of [35] consumes fewer resources than the LU-2 implementation of [P1] but its throughput is lower. The pipelined implementation of [35] achieves a very high throughput but it consumes almost twice the amount of resources of the pipelined implementation in [P1]. Despite of the lower clock frequency, the performance of [P1] is better in feedback modes.

After [P1] the research work has been more concentrated on the implementation of 3DES and improved results for non-pipelined, compact designs have been reported [140, 207, 214]. Specifically, [207] maps DES/3DES very effectively to FPGAs.

5.3 Hardware Implementation of RC4

In contrast to AES and 3DES, RC4 contains much less parallelism and only operations that are inherent in general-purpose processors [P5]. Furthermore, consecutive operations in RC4 are strictly dependent on each other [P5]. Hence, it cannot benefit from a dedicated hardware implementation as much as the other ciphers in terms of throughput.

Hardware implementation of RC4 has been studied only in few publications. Similarly to the earliest DES publications, most of them have concentrated on the implementation of high-performance key search engines using parallel RC4 units [78, 149, 233]. The implementations [78, 149] published before the RC4 implementation of [P4] do not report size or performance figures for a single RC4 unit. Hence, [P4] has been the first to present a detailed hardware implementation of a single unit suited for WLAN devices.

After [P4] improved hardware implementations have been reported in [141, 233]. Both the implementations use the same structure. Only [141] presents results for a single RC4 implementation instead of parallel key search units. In the Xilinx 2V250FG256 FPGA the throughput is 176 Mbit/s at 64 MHz, excluding the key setup latency. The implementation consumes 138 slices and three Block RAMs (BRAM). In [233] only one BRAM is used per RC4 unit.

5.4 Hardware Implementation of Modular Exponentiation

Without applying a proper algorithm and implementation, performing a modular exponentiation can consume a substantially large amount of time and space. For example, computing a power without any reductions in between can result in very large

intermediate results even if the modulus was small. Therefore, large modular exponentiations are typically computed as a series of multiplications and modular reductions [171].

The most widely utilized algorithm combining multiplication with modular reduction is Montgomery multiplication [19, 248], originally published in [176]. This Thesis concentrates on the hardware implementation of modular exponentiation using Montgomery's algorithm. A comprehensive review of the hardware approaches for the implementation of Montgomery's algorithm as well as modular exponentiation can be found in [19]. Montgomery's algorithm and its usage for exponentiation are described in [P6–P8]. The publications contain up-to-date reviews for the related work of their implementations. Therefore, the implementation of modular exponentiation is not further discussed here.

The publication [P6] focuses on the hardware acceleration of the SRP protocol using a modular exponentiation architecture based on the architectures of [26, 27], which have proved to provide high performance in FPGAs. To the knowledge of the author, a hardware-accelerated SRP implementation has not previously been reported. In [P7] the exponentiation implementation of [P6], as well as that of [26], is further improved by removing unused clock cycles. A different version of Montgomery's algorithm and a different hardware architecture for the modular exponentiation have been used for trading the high performance to compact size in [P8].

5.5 Specialized Processor Architectures

Various reconfigurable architectures as well as ASIPs have been proposed for accelerating cryptographic applications. Many designs, such as [40,50,89], have traded size and power consumption to reconfigurability and/or performance, which makes them unsuited for WLAN devices. Hardware architectures specifically designed for IEEE 802.11i processing have been published in [15, 222]. AES, RC4, Michael, and CRC hardware implementations are integrated into a single design and control for 802.11i processing is added. The papers do not report novelties for the separate components compared to the implementations reviewed in Section 5.2 and Section 5.3. Resource-efficient and potentially low-power processor architectures for the cryptographic processing of WLANs have been presented in [52,81,83,160,185,193,200,253]. These designs are reviewed and compared with the TTA ASIPs of [P5].

An energy-efficient reconfigurable processor for public-key cryptography is presented in [81]. The reconfigurability is provided by the data path that can be configured to

perform different operations of public-key algorithms, such as Montgomery multiplication and GF multiplication. For example, a 1024-bit modular exponentiation is computed in 32.1 ms at 50 MHz. The operation takes several tenfolds longer as a software implementation on a typical embedded processor [7, 102, P6, P8]. Since the reconfigurability is limited to the specific application domain, the performance as well as the energy consumption of the reconfigurable processor are significantly better than those of FPGAs. A design considering the acceleration of public-key algorithms using instruction set extensions is presented in [83] and discussed in [P8]. In contrast to [81, 83], the ASIPs of this Thesis [P5] have been development for the secret-key cryptography of WLANs, especially for mechanisms based on AES and RC4.

The ASIP implementation reported in [200] uses the Xtensa configurable processor architecture from Tensilica [79]. Xtensa includes similar features as TTA but is based on the traditional operation triggered programming paradigm. In [200], the execution of DES/3DES, AES, and RSA is accelerated by extending the instruction set with algorithm-specific instructions. As a result, the performance is improved by several tenfolds from the original (however, the original implementations are fairly poor). For example, the achieved throughput for AES is 17 Mbit/s (88 cycles/byte) at 188 MHz. In [P5], the throughput of the AES ASIP, which also supports RC4, is 68 Mbit/s (12 cycles/byte) at 100 MHz using precomputed roundkeys. Significant speed-up has also been achieved for DES using instruction set extension in Xtensa [79].

An ASIP architecture using similar types of configurable functional units as [P5] has been published in [160]. The design is based on the 32-bit MIPS processor architecture. A special unit supporting fast LUT functionality is included for accelerating RC4 and AES. The unit consists of a two 1024×32 bit RAMs, which require significantly larger area than the RAM-based LUTs of [P5]. For accelerating DES/3DES, [160] proposes a very large configurable permutation unit consisting of 512 32×1 bit multiplexers. The achieved throughput for AES is around 64 Mbit/s at 100 MHz.

High-speed processor architectures that include functional units for the same purposes as [P5] are presented in [185, 253]. Both the architectures imply high throughput but are also likely to consume a larger amount of hardware resources than the ASIPs of [P5]. However, gate counts have not been reported for direct comparison. The reconfigurable processor in [185] consists of two parallel computing clusters, both having 64-bit data paths. Special operations are included for fast LUT/permutation functionality as well as for AES-specific logical operations. The throughput for AES is 730 Mbit/s at 400 MHz. The processor in [253] contains four parallel 32-bit

processing units, each consisting of an adder, units for logical operations, and RAM for LUT functionality. Two of the units include a multiplier. The throughput is 512 Mbit/s at 360 MHz for AES.

A microcoded cryptoprocessor designed for executing DES, AES, and ECC has recently been published in [52]. The data path contains an expansion/permutation unit, a shifter, four memory-based LUTs, two logic units, and a register file consisting of sixteen 256-bit registers. The processor can be reconfigured by modifying the microcoded program and the contents of the LUTs. The encryption throughput for AES is 1.83 Mbit/s at 13.56 MHz with on-the-fly roundkey generation, which means a significantly higher cycle count per processed block than in the AES ASIP of [P5]. The hardware area has not been clearly reported for comparison.

A coarse-grained architecture in which a complete AES implementation is integrated into a 32-bit RISC processor is presented in [193]. The implementation can be reconfigured to allow the processor to use the memory and computational resources of the AES core for other tasks as well, e.g. error correction. The throughput for the AES encryption is 252 Mbit/s at 126 MHz. In [P5], a finer grained approach is used, which allows developing a wide scale of processor configurations for different application requirements. Instead of implementing full algorithms in hardware, the ASIPs of [P5] contain compact special operations, the usage of which is defined by the software running in the ASIP. Also in [P5] the operations are applicable beyond the targeted encryption algorithms.

6. RESEARCH RESULTS

This chapter summarizes the main results of the research reported in this Thesis. The detailed results are presented in the publications [P1–P10]. The main area of the research has been the development of hardware architectures for efficiently supporting WLAN cryptography. The other research topic has been the development of security designs for improving the cryptographic protection of WLANs and their applications. The presented security designs can be implemented utilizing the developed hardware architectures.

The presentation in this chapter mainly follows the order of the performed research, starting from the hardware development and finishing with the design of a full application, the offline RTB. Section 6.1 focuses on the design and implementation of hardware support for the secret-key cryptography of WLANs. The developed modular exponentiation architectures for public-key cryptography are described in Section 6.2. The security designs developed during the research work, i.e. a security enhancement layer for Bluetooth and the RTB application, are summarized in Section 6.3.

6.1 Hardware Architectures for Secret-Key Cryptography in WLANs

The secret-key algorithms examined in this work are AES, 3DES, and RC4 as well as the AES modes of operation utilized in WLANs. Various hardware architectures have been designed and implemented, providing different trade-offs between hardware area, performance, and power consumption.

6.1.1 AES Architectures

The development of hardware architectures for AES and its modes of operation in WLANs has been one of the main themes in the research. The results of the AES research are reported in [P1–P3, P9].

Full-Width Architectures

Publication [P1] presents two iterated AES architectures (Fig. 16(a)) using 128-bit data paths. The first architecture supports only the 128-bit-key encryption functionality of AES. The second one supports both encryption and decryption. Both the designs compute the roundkeys on-the-fly in parallel with the encryption data path. In the second architecture, the roundkeys are generated on-the-fly also in the reverse direction (i.e. for decryption). This requires computing the roundkeys once in the forward direction before the decryption can begin. While the first architecture implements the SubBytes operation with table lookups, in the second architecture the operation is separated into two phases: a GF inverse table lookup and an affine transformation. This allows sharing the GF inverse implementation between encryption and decryption, which decreases the total hardware area.

The two designs were implemented in the Xilinx XCV800FG676-6 FPGA of the TUTWLAN prototyping platform described in [231]. In both the architectures encrypting a block of data takes 11 clock cycles. The same applies for the decryption in the second architecture as long as at least one encryption is performed with the new key first. Otherwise the latency for the first decryption is 22 clock cycles. Loop-unrolling and/or pipelining were not considered as they were estimated to exhaust the resources of the used FPGA device. The encryption-only implementation consumes 2272 slices and processes 364 Mbit/s at 31 MHz. The encryption-decryption implementation occupies 3267 slices and processes 246 Mbit/s at the maximum clock of 21 MHz.

The encryption-only architecture has been utilized in the prototype implementation of the Enhanced Security Layer (ESL) for Bluetooth (Section 6.3.1) reported in [P9]. In the prototype, the design was synthesized to the Programmable Logic Device (PLD) of the Altera EPXA10F1020C2 chip, in which the area consumption of the AES implementation is 1246 LEs and 20 Embedded System Blocks (ESB). The implementation encrypts 507 Mbit/s at 44 MHz.

In [P2] the encryption-only architecture of [P1] is further tuned and synthesized to the low-cost Altera Cyclone EP1C4F324C6 FPGA. Encrypting a block of data takes 10 clock cycles resulting in the throughput of 1408 Mbit/s at 110 MHz. Compared to the architecture of [P1], one clock cycle per block is saved by performing the first AddRoundKey operation while inputting the data block. The implementation consumes 1368 LEs and 10 M4K memory blocks of the Cyclone FPGA.

32-bit Architecture

In order to trade throughput to lower area and power consumption, a 128-bit-key AES encryption architecture with a 32-bit data path was designed in the research work reported in [P2]. The design, shown in Fig. 17, is based on the work of [34]. In [P2], the decryption functionality is removed and the AES core is used for building a hardware for the security processing of IEEE 802.15.4 (see below).

In contrast to the architectures discussed above, in this one the roundkeys are precomputed, which allows utilizing the encryption data path for computing the roundkeys. Precomputing requires setup time and storage space (a 32-bit RAM with 44 memory slots) but it can also imply power savings as the key logic has to operate only once per key [34]. The ShiftRows operation can be performed with addressing logic as the intermediate results are stored in a Dual-Port RAM (DPRAM) in the encryption data path. Structures for further tuning the amount of required storage space are discussed in [P3].

Compared to full-width AES architectures, significant area and power savings were

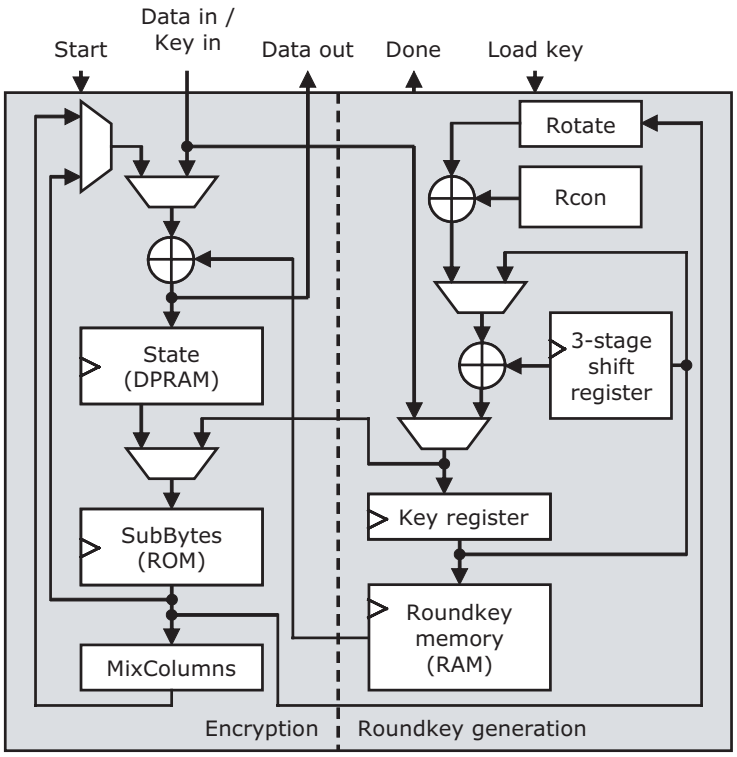


Fig. 17. *Architecture of the 32-bit AES encryption core.*

achieved with the 32-bit architecture [P2]. In the Altera EP1C4F324C6 FPGA the 32-bit design consumes 512 LEs and 7 M4Ks. Due to the reduced parallelism, the throughput decreases to 270 Mbit/s at the maximum clock of 116 MHz (55 cycles per block excluding the roundkey generation).

Support for Modes of Operation

Instead of the AES algorithm itself, [P2, P9] concentrate on the hardware architectures supporting various AES modes of operation used and proposed to be used in WLANs. The designs utilize the AES encryption cores discussed above.

The publication [P9] presents the hardware architecture shown in Fig. 18. It supports AES processing in the CBC-MIC mode as well as three combinations of the CTR mode and the CBC-MIC computation. In the combined modes, the MIC can be computed prior to or after the CTR encryption using separate keys. The third option is CCM, which uses the same key for encryption and MIC computation. The architecture utilizes the encryption-only AES core of [P1].

The security processing architecture was implemented as a part of the ESL prototype in the Altera EPXA10F1020C2 programmable chip [P9]. The same prototyping

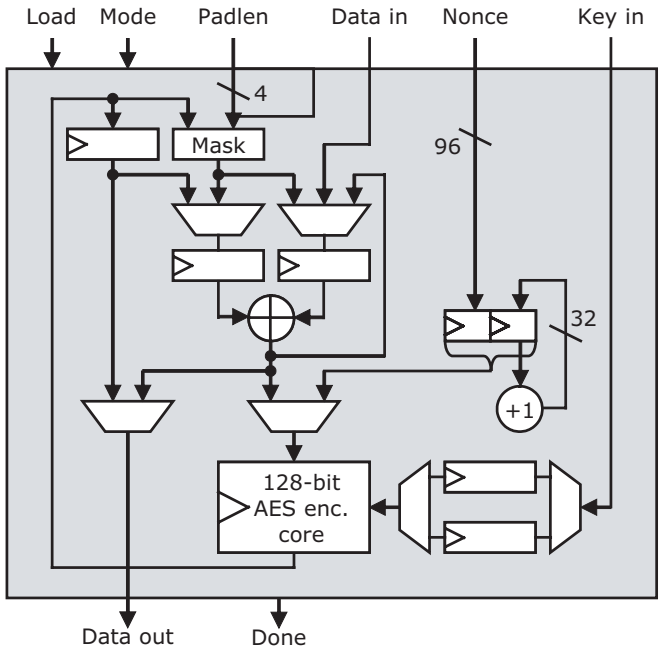


Fig. 18. *Security processing architecture of the Enhanced Security Layer (ESL) prototype.*

platform has also been used in the TUTWLAN protocol development [148]. In the FPGA, the resource consumption of the AES core with the ESL modes of operation is 3527 LEs and 28 ESBs. In the combined modes the throughput is 214 Mbit/s at 44 MHz (26 cycles per block). The hardware design can be used for the security processing of standard WLAN technologies as well, e.g. for 802.11i and 802.15.4.

A compact architecture supporting the security processing of 802.15.4 is reported in [P2]. The architecture, illustrated in Fig. 19, supports the CTR, CBC-MIC, and CCM modes of operation. The AES core used in the design is the 32-bit architecture of Fig. 17. The total area of the design is 1434 LEs and 11 M4Ks in the Altera EP1C4F324C6 FPGA. It processes 176 Mbit/s (57 cycles per block) in the CTR and CBC-MIC modes and 90 Mbit/s (112 cycles per block) in the CCM mode at 78 MHz, excluding the roundkey generation latency, which is 48 cycles for a new key.

Compared to typical WLAN processors, the prototype FPGA implementation and an estimate of an ASIC implementation offer significantly higher performance and lower energy consumption [P2]. Compared to the processing hardware of [P9], lower resource consumption is achieved at the cost of performance. In addition to 802.15.4,

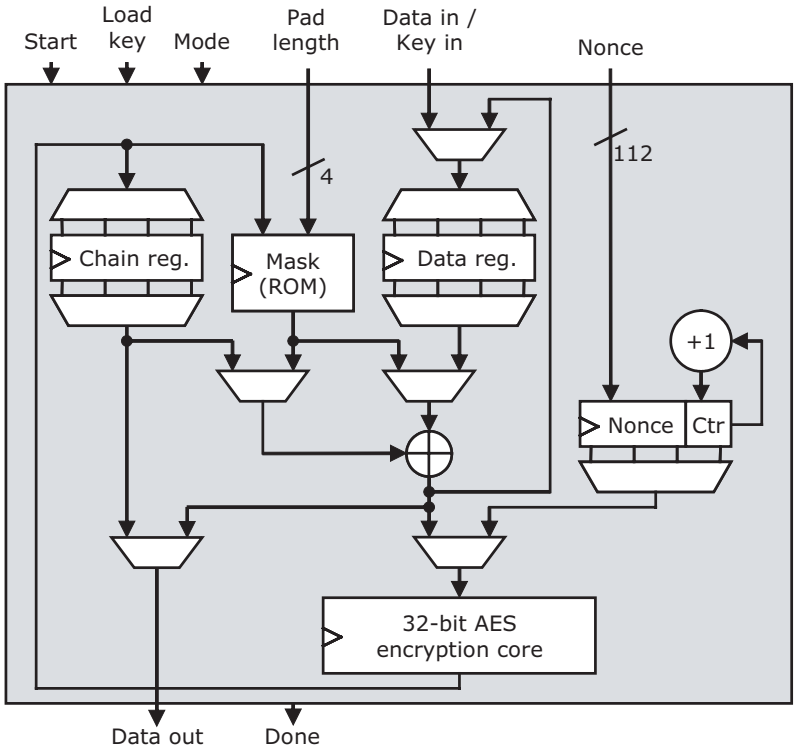


Fig. 19. *Security processing architecture for IEEE 802.15.4.*

this design supports also the security processing of the technologies derived from 802.11i.

Permutation Architectures

The research reported in [P3] focuses on the design of efficient architectures for storing the intermediate results during AES processing. Both register-based and memory-based approaches are considered. The architectures are designed so that the ShiftRows operation is automatically performed when the data are read from the storage. Previous works, such as [34, 197, 209, P2], have concentrated on 32-bit architectures and required separate memories (or registers) for reads and writes. In the designs of [P3], the same storage space is used for both, implying area-efficiency. In addition, [P3] describes storage structures for 8-bit, 16-bit, 32-bit, and 64-bit data paths. Besides plain hardware implementations, ASIPs can benefit from the proposed architectures as a form of special operations/register files.

The 8-bit register-based design is utilized in [96], which reports a very compact and low-power 8-bit AES encryption core. Compared to previous 8-bit designs, the AES core provides significantly higher performance at corresponding area. It has previously been believed that an 8-bit AES architecture results in an inefficient implementation [215].

6.1.2 3DES Architectures

In [P1], three hardware architectures with 64-bit data paths are reported for 3DES. They were implemented in the Xilinx XCV800FG676-6 FPGA of the TUTWLAN prototype presented in [231]. In addition to [P1], details about the 3DES architectures can be found in [100].

The first 3DES en/decryption design is an iterated architecture (Fig. 16(a)) consisting of a data path and a roundkey generation component that operate in parallel, i.e. roundkeys are generated on-the-fly. A block of data is processed in 51 clock cycles. In the target FPGA, the throughput is 45 Mbit/s at 36 MHz and the resource consumption 740 slices.

The second architecture uses the LU-2 structure (Fig. 16(c)) in which two rounds are performed per clock cycle. Also the roundkey component produces two keys per cycle. The cycle count for processing a block is 27. Even though the maximum clock frequency was decreased from the first design, the throughput was still increased.

The throughput of the design is 69 Mbit/s at 29 MHz and the resource consumption 833 slices.

In order to maximize the throughput, pipelining was utilized in the third architecture (Fig. 16(b)). According to the results of the two other implementations, executing two iteration rounds per clock cycle was considered the most effective approach for the round component. Hence, the pipelined design consists of 24 consecutive double-iteration rounds with registers in between and the latency is 24 clock cycles. With this design the throughput increases considerably as after the first 24 clock cycles a new block is output on every clock cycle. The throughput is 2900 Mbit/s at the maximum clock frequency of 46 MHz. The implementation consumes 6689 slices of the FPGA. However, as stated earlier, the pipeline cannot be fully utilized in feedback modes of operation.

Instead of heavily tuning the algorithm implementations, especially with technology-specific optimizations, the purpose of [P1] was to compare the resource requirements and performances of 3DES and AES in comparable implementation architectures. At the time of the research was performed, the Rijndael algorithm was selected as AES, the successor of DES/3DES. As a summary for 3DES in [P1], it can be deduced that 3DES requires a significantly smaller amount of resources but implies lower performance than AES. 3DES can even be implemented as a fully pipelined architecture without extremely high hardware resource consumption.

6.1.3 RC4 Architecture

The main purpose of the research reported in [P4] was to evaluate the suitability of the RC4 stream cipher for hardware implementation. The algorithm consists of two main components: a key-dependent function and a 256-byte array, which maintains the internal state of the cipher [217, P5]. The designed hardware architecture is composed of a state machine that implements the functionality of the key-dependent function and a memory component that implements the byte array.

In [P4], the design was synthesized to the Xilinx XC4013EPQ208-2 FPGA of the first TUTWLAN prototyping platform [132]. The throughput, excluding the initialization of the byte array, is 17 Mbit/s at the maximum clock frequency of 17 MHz. The implementation consumes 224 Configurable Logic Blocks (CLB) of the FPGA. Compared to software implementations of RC4 on commercial processors of the WLAN domain [P5], lower cycle count per produced pseudo-random byte was achieved.

For evaluating the cipher on modern hardware technologies, the same RC4 design

was later synthesized to the Xilinx XC2V250FG256-4 FPGA and to a 0.13 μm CMOS standard-cell ASIC technology [P5]. In this FPGA the throughput is 103 Mbit/s at the maximum clock frequency of 103 MHz and in the ASIC technology 200 Mbit/s at 200 MHz. In the ASIC technology the hardware area of the design is 3.0 kgates. These results show that despite of the low cycle count, the critical path of the design is still short, allowing operating on high system clock frequencies.

6.1.4 Transport Triggered Architecture Processors for WLAN Encryption

Transport Triggered Architecture (TTA) processors [37], utilized in the ASIP development in this work [P5], offer a cost-effective trade-off between the size and the performance of ASICs and the programmability of general-purpose processors. A TTA processor is depicted in Fig. 20. The Central Processing Unit (CPU) is organized as a set of Functional Units (FU), Special Functional Units (SFU), and Register Files (RF). The data transfers between the entities are performed by an interconnection network consisting of a variable number of buses and bus connections.

In contrast to traditional, operation-triggered processor architectures, in a TTA processor operations occur as side effects of data transports [37]. The execution begins when data are written to the operand registers of a FU. Only a single instruction, *move*, is required for low-level programming. The mirrored paradigm enables applying new scheduling, bypassing, and resource allocation techniques in high-level language compilers. The TTA development framework provides various tools for ASIP design, including automatic design space exploration [38].

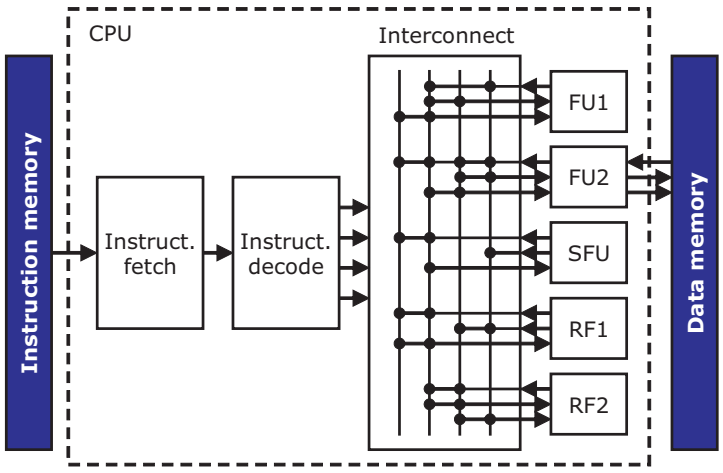


Fig. 20. *Organization of a Transport Triggered Architecture (TTA) processor.*

In [P5], TTA processors for the RC4 and AES encryption algorithms of the 802.11i security standard are designed. Whereas the AES-based security design of 802.11i is completely new, RC4 is included for backward compatibility with the fielded hardware. Thus, support for both the algorithms is required in new products. In this work, special operations efficiently supporting the ciphers are developed and the TTA design flow is utilized for finding processor configurations with the best performance-area ratios for the algorithms. In addition to high encryption performance-cost ratios, the designed processors as well as most of the SFUs are general-purpose, enabling executing other applications in them as well.

Two SFUs were designed for the RC4 acceleration. A single-cycle SFU, LUT-SFU, was included for maintaining the 256-byte array of the algorithm. The SFU provides short memory access latency and avoids the computation of the absolute main memory addresses when referring to the array. In addition, since only 32-bit operations were implemented in the standard TTA hardware and RC4 operates on bytes, a simple byte adder was also implemented as an SFU.

Also the AES implementations in TTA make use of LUT-SFU for performing fast table lookups when performing the SubBytes operation. In addition, two SFUs were designed for increasing the AES performance. CONV-SFU can be used for converting between byte and word representations as both the access formats are required during the execution. MIXADD-SFU is an AES-specific SFU for performing a combined 32-bit MixColumns and AddRoundKey operation in a single clock cycle.

A large number of TTA configurations with varying area and performance combinations were produced for both the algorithms using the design space explorer tool of the TTA development framework. Adding the SFUs increased the performance significantly. Also, the performance-area ratio was improved after adding the SFUs, which means that the designed SFUs improved the performance more than they affected the hardware area.

The size of the chosen TTA configuration efficiently supporting AES and RC4 is 69.4 kgates and the throughput 100 Mbit/s for RC4 and 68.5 Mbit/s for AES at 100 MHz in the used 0.13 μm CMOS technology. Compared to commercial processors of the same wireless application domain, higher throughput is achieved at significantly smaller area and lower clock speed, which also results in decreased energy consumption [P5]. The designs can be extended further e.g. with SFUs derived from the structures presented in [P3].

6.2 Modular Exponentiation Architectures for Public-Key Cryptography

In this work, two high-level approaches have been examined for the implementation of modular exponentiation of large integers using Montgomery's reduction algorithm [176]. The first one constitutes of a systolic array that provides high performance but also results in large area [P6, P7]. The second approach trades the performance to lower hardware area [P8].

6.2.1 Systolic Arrays

The exponentiation accelerator presented in [P6] is based on the work of [26]. The high-level organization of the design is presented in Fig. 21. It constitutes of a systolic array of 8-bit Processing Units (PU). The array performs modular multiplications using Montgomery's algorithm and the control logic feeds the array with suitable inputs in order to compute exponentiations. The intermediate results are stored in RAM components. Exponentiations are computed with a binary square-and-multiply algorithm [P6]. An exponentiation is composed of squarings and multiplications that are interleaved in the PUs. A PU operates in a two-cycle mode in which the first clock cycle is used for a squaring and the other one for a multiplication.

The main purpose of the research in [P6] was to accelerate the execution of SRP, which is an authentication and key-exchange protocol designed for password verification and session key generation over an insecure communication channel. It is potentially a well-suited and secure protocol for user authentication in WLANs through EAP and/or TLS adaptation [32, 230]. In addition to the hardware architecture for modular exponentiation, [P6] reports a full implementation of SRP, consisting of the hardware accelerator, its interface, and software routines.

The SRP accelerator was implemented in the Altera EPXA10F1020C2 programmable

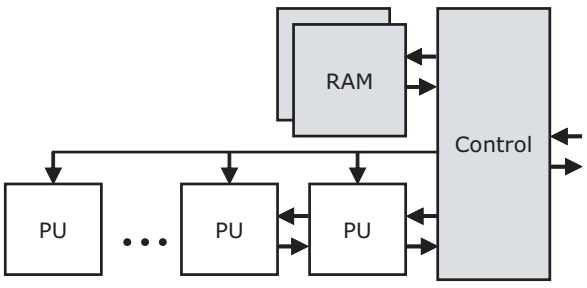


Fig. 21. *High-level organization of the systolic array exponentiation architecture.*

device. With 1024-bit operations, the area consumption of the exponentiation hardware is 9644 LEs and 5120 RAM bits. The maximum clock frequency is 65 MHz. It was measured that a full exponentiation, in which the exponent and the modulus are of the same length, takes 32 ms in the hardware. Compared to a software implementation with a tuned cryptographic library in the ARM922 processor core of EPXA10F1020C2, the SRP execution was significantly improved by exploiting the exponentiation accelerator.

In [P7], the performance of the exponentiation architecture of [P6] is improved. The new design uses the same systolic array architecture. However, whereas a PU of [P6] operates in the two-cycle mode, in [P7] a PU operates in a single-cycle mode. In this mode, the PUs compute the multiplications and squarings of the binary exponentiation algorithm in series, which saves clock cycles. In the design of [P6], the multiplication phase of an exponentiation is always performed even if the multiplication result was not needed. In [P7], the multiplication is only performed when needed. In addition, the control logic is improved for higher performance.

In the Altera EPXA10F1020C device the 1024-bit exponentiation implementation including an ARM922 interface consumes 13960 LEs and 4096 RAM bits. The maximum clock frequency is 63 MHz. A full exponentiation is performed in 25 ms on the average [P8]. In terms of clock cycles, the design of [P7] is statistically 25% faster than the design of [P6]. Whereas the total cycle count of [P6] is only dependent on the lengths of the operands, in [P7] the cycle count depends also on the number of ones in the binary representation of the exponent.

6.2.2 Compact Exponentiation Architecture

The previous systolic array designs resulted in high performance at the cost of hardware area. Decreasing the hardware resource consumption while providing good performance was addressed in the research reported in [P8]. The designed architecture is targeted to modern FPGA devices which contain advanced functional blocks for high-speed and area-efficient implementations. The design especially makes use of fast Digital Signal Processing (DSP) multipliers.

Whereas the architectures of [P6, P7] compute modular multiplications required for an exponentiation using full-width operands, in the design of [P8] operands are decomposed into k-bit digits. The architecture is shown in Fig. 22. Due to the reduced parallelism, the total cycle count required for an exponentiation increases. In order to map the digit-serial design effectively to the DSP multipliers of the target FPGA,

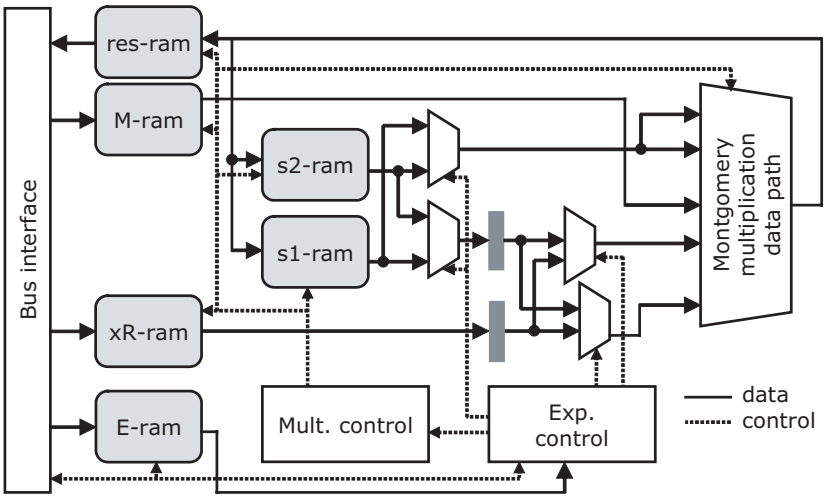


Fig. 22. *Compact architecture for modular exponentiations.*

Altera Stratix EP1S40, k was chosen to be 18.

The accelerator consumes 341 LEs, one DSP block, and 13604 RAM bits in the FPGA. It is able to perform modular exponentiations with up to 2250-bit integers and scales easily to larger exponentiations by increasing the memory size. Excluding pre and post processing, 1024-bit and 2048-bit exponentiations are performed in 28 ms and 212 ms, respectively, at the maximum clock frequency of 198 MHz. Due to its compactness, standard interface, and support for different clock domains, the accelerator can also effortlessly be integrated into a larger system implemented in the same FPGA.

Despite the increase in the total cycle count, high performance was achieved with the compact exponentiation architecture. A full 1024-bit exponentiation can be performed almost as fast as in [P7]. This is due to the compact size, the resource-efficient mapping to the FPGA, and the utilization of the DSP multipliers, which resulted in very high operating frequency.

In [7], the implementation is further evaluated and tested against a software exponentiation implementation in a NIOS II CPU in the same FPGA. Compared to the exponentiation performed fully in software, the accelerated version is about 20 times faster. When the accelerator is clocked with the same frequency as the CPU, the speedup is still tenfold.

6.3 Security Designs

This section presents the two security designs developed in the research work. The first one is an Enhanced Security Layer (ESL) for Bluetooth. The second is the RTB application. For allowing the free movement of users, wireless networks are preferred in providing the RTB services. Even though the RTB application is not specifically designed to be provided through WLANs, WLANs are seen as one of the best-suited technologies for its implementation.

6.3.1 Enhanced Security Layer for Bluetooth

In order to address the weaknesses of the Bluetooth security design reviewed in Section 4.3, ESL for protecting Bluetooth links is proposed in this work [P9]. ESL specifically aims at fixing the shortcomings of the Bluetooth encryption algorithm and the lack of cryptographic integrity protection. The Bluetooth stream cipher is replaced with AES, which is utilized in providing confidentiality as well as data authentication for transmissions. The components of ESL and its placement in the Bluetooth protocol stack are depicted in Fig. 23.

ESL supports four well-scrutinized operation modes from which the application can choose the preferred one according to its security and processing requirements. The CBC-MIC mode is suited for applications which do not require the confidentiality of data. Confidentiality and data authentication are provided by the other three modes:

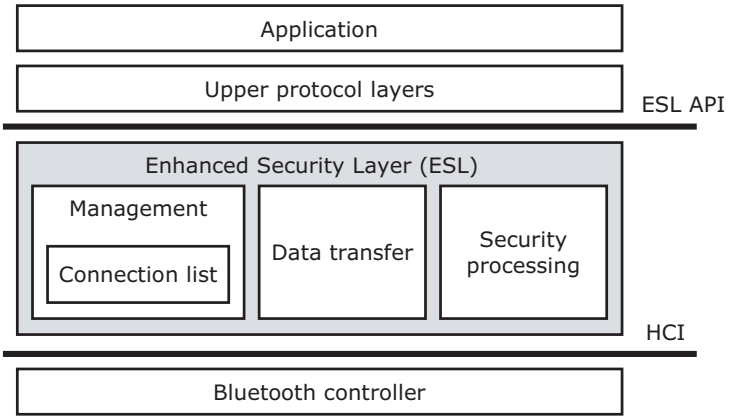


Fig. 23. *Architecture of the Enhanced Security Layer (ESL) for Bluetooth. ESL is placed on top of the standard Host Controller Interface (HCI) and accessed through the ESL Application Programming Interface (API).*

CTR encryption followed by CBC-MIC, CBC-MIC followed by CTR encryption, and CCM. The security-related inputs for the operation modes, such as keys and nonces, are provided by the application run above ESL. The support for the standard Bluetooth security is also included in ESL for the interoperability with devices that do not contain the enhanced features.

The chosen cipher (AES) and the modes of operation used in ESL are purposely consistent with the other new WLAN technologies, such as 802.11i and 802.15.4. This way the low-level security processing components of ESL are compatible with those of the other significant WLANs, which enables efficient resource-sharing in devices supporting multiple WLAN technologies. Currently, Bluetooth is the only widely-used WLAN technology that has not been updated to use AES in its security schemes.

ESL is placed on the top of HCI, which means that it can be integrated as an additional module into any standard Bluetooth implementation. In addition, the placement results in low transmission overhead as the added protocol fields, such as MICs, are transmitted in HCI packets instead of every baseband packet. On the other hand, this also means that ESL can only protect data coming from above HCI.

A prototype of ESL, consisting of hardware and software components, an Application Programming Interface (API), and a standard Bluetooth controller, was implemented in the Altera Excalibur EPXA10 DDR Development Kit, which contains the EPXA10F1020C2 programmable device [P9]. The prototype implementation is illustrated in Fig. 24. The security processing component contains the hardware architecture of Fig. 18.

Compared to the hardware implementation of the standard Bluetooth security [142], the higher security level of ESL is achieved at significantly lower hardware resources and higher performance [P9]. Instead of using E_0 for encryption and SAFER+ for authentication and key generation, in ESL all three procedures can utilize AES. This reduces the hardware resources and also implies shorter latencies due to the higher performance. The hardware resource consumption can be further decreased e.g. by switching the used AES core to the 32-bit core of [P2].

ESL as reported in this Thesis has not been specified to include an entity authentication and/or a key agreement protocol. However, the work has later been extended to include mechanisms for these purposes as well [98]. Also, the extended design only allows safer combinations of the standard Bluetooth security parameters.

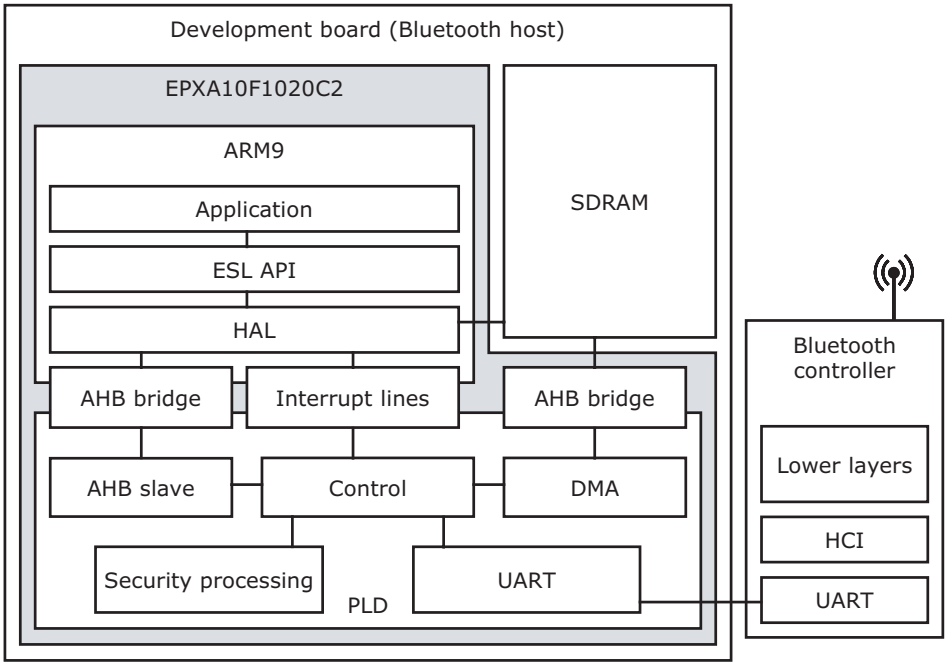


Fig. 24. *Organization of the Enhanced Security Layer (ESL) prototype.*

6.3.2 Real-Time Betting Application

Even though enabling electronic betting as an expanding entertainment field, the advanced communication and e-commerce technologies have not changed the nature of betting itself. The targets in electronic systems are still the same as in manual systems, and often provided in both ways. 'Will this penalty kick result in a goal' is an example of an unsuitable incident in a soccer match in current systems. There is not enough time or processing resources available at reasonable costs in user terminals, networks, and servers for handling such a bet in a way that provides fair, equal possibilities for betting to all participants. This has been demonstrated in this work with simulations in WLAN environment [P10].

In order to change the characteristics of betting itself, the wireless RTB application has been developed in this work [P10]. It changes betting services into interactive sessions by supporting short, unpredictable target incidents as well as crediting winnings in real-time. The architecture of the RTB application is illustrated in Fig. 25. In contrast to existing online applications, the RTB application, referred to as *offline RTB*, does not require uplink communication from users to the betting organizer during the betting session. Instead, the user terminals only receive broadcast transmis-

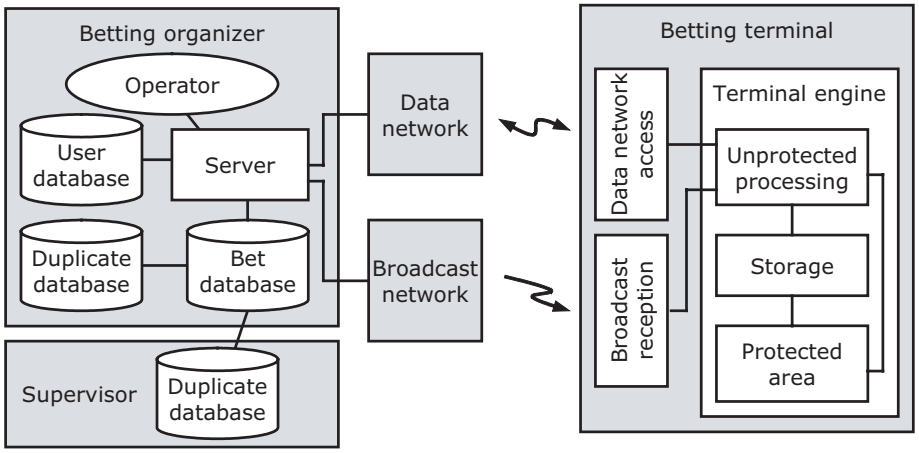


Fig. 25. *Architecture of the offline Real-Time Betting (RTB) application.*

sion from the organizer server and locally time-stamp and store the placed bets. The server collects the bet records after the target event is finished and incident outcomes already revealed. Therefore, reliable methods are required for ensuring the authenticity of the operations during the session, especially at the user terminals. For this purpose, an offline RTB terminal includes a *protected area*, which only an authentic server can access through the protected data and broadcast links. The protected area is preferably implemented with integrated hardware components for increasing the level of physical security and performance.

In order to test the offline RTB in practice, prototypes for WLAN and Digital Video Broadcasting (DVB) environments have been developed [101, P10]. They enable evaluating design alternatives, network technologies, user interfaces, and end-user behavior in real environments. The prototypes have been experimented by several test groups. Valuable information on the attractiveness of the concept as well as its usability and applicability has been collected. Main aspects have been to investigate how near the closing time bets are placed, what kinds of bets are suitable, and what a suitable interval for announcing bets is.

The general operation of the offline RTB application as well as the designed cryptographic protection and timing methods are overviewed below. The details of the operations are presented in [P10].

General Operation

Participating betting in the offline RTB comprises of three stages: *registration*, *betting session*, and *bet record collection*. In the registration phase, the user accesses the betting server, chooses an event to participate, and reserves credit for it through the data network. In order to guarantee valid operation during the betting session, the protected area and the server agree on cryptographic keys as well as synchronize their clocks. The keys are required for the data authentication of broadcast transmissions as well as for protecting the records of placed bets during the betting session. The synchronized clock is used for verifying the timing of broadcast transmissions as well as for time-stamping the stored bet records.

The betting session utilizes only the broadcast link. As a suitable incident emerges, the betting operator broadcasts an open bet packet, which contains the information of the betting target. If the user places a bet, the placement information is given to the protected area for time-stamping. Before the outcome is settled, the bet is closed with a close bet packet. If a bet was placed, the protected area verifies that it was placed before the close time received in the close bet packet. When the outcome is clear, the terminal receives a bet result packet, which includes the realized bet option. The user is credited with the possible winnings in real-time. The protected area verifies the authenticity and timing of each broadcast packet and disables betting if a set violation threshold is exceeded.

The bet record collection utilizes again the data network. The server requests the terminal to transmit its stored bet records, checks that the data are valid, and computes the final outcome for the session. Possible winnings are added to the user's betting account.

Cryptographic Protection

The offline RTB application uses cryptography for protecting the broadcast and data links as well as the locally stored, time-stamped bet records.

The protected and unprotected areas in Fig. 25 share the two-way data link by creating their own logical connections to the server. Both the connections are mutually authenticated and fresh session keys are generated between the communicating parties. In addition, the utilized security protocols must ensure the confidentiality, freshness, and data authentication of the communications following the entity authentication. Commonly trusted secure channels, such as TLS or Secure Sockets Layer (SSL) and the mechanisms of 802.11i, can be used for the purpose.

In contrast to the data link, broadcast packets carry data to both the protected area and the unprotected area. As the broadcast packets do not contain confidential information, it is enough to ensure only the authenticity of the data, which is provided with a digital signature in each packet. For example, signature mechanisms using modular exponentiations are suitable for this. The signature verification is performed by the protected area whereas the unprotected area processes the fields related to betting itself.

In order to reduce the resource requirements and cost of the protected area, the storage for the session data is located in the unprotected area of the terminal. The storage entries are cryptographically chained in the protected area. The chain ensures the authenticity of the data since altering or deleting an entry makes also the rest of the storage invalid. The chain is computed using a MIC algorithm, such as AES in the CBC-MIC mode. The chaining key is generated during the registration and updated for every entry in the storage using a hash algorithm, such as SHA-256. The previous chaining key is always irretrievably destroyed. Therefore, even compromising a terminal will not allow changing entries that have been generated with a passed key. After collecting the session data, the server verifies that the storage chain is valid, based on the recorded registration data.

Timing

Compared to existing distributed systems, the time synchronization problem is different in the offline RTB. Usually, a user wishes the synchronization to be as accurate as possible. On the contrary, in the offline RTB a malicious user wants the synchronization to be inaccurate so that bets could be placed after the results of target incidents are available. The offline RTB application also changes the synchronization requirements and possibilities from many existing systems. The absolute long-term synchronization accuracy does not have to be very high since the synchronization has to be maintained for a fairly short period of time. In addition, the timing resolution can be moderate as only the timing of actions performed by human users is considered. For a successful attack, a human attacker must have enough time to first observe the result of an incident and then place a winning bet.

The synchronization procedures developed in this work address maintaining a reliable relationship between the time of the protected area and the time of the server and detecting attacks on timing within the considerations above.

All the time-related processing in the protected area is based on a *protected counter*.

The main purpose of the counter is to provide time-stamps for bet records. The requirements are that the counter is monotonic, relatively stable and accurate, it does not roll over between the registration and the bet record collection, its precision is at sub-millisecond level, and its frequency cannot be significantly affected by changing the operating conditions.

The protected area uses the counter for two purposes. First, it receives periodic broadcast packets for controlling that broadcast transmission is actually received and it is not delayed. Each periodic packet has a predefined time window, inside which it must be received. Even though it is required that it is difficult to tamper with the protected area, these verifications are used for detecting changes in the counter rate.

The other use for the counter is to maintain an estimate of the server time during the betting session. The estimate is required for time-stamping bet records and verifying the timing of broadcast packets, which prevents recording and replaying a complete session. The error of the estimate is not allowed to exceed set limits. The estimate is synchronized to the server time during the registration through the data network. Regardless of the method used in the synchronization, it is required that the protected area reasonably accurately knows the network round-trip delay of the message exchange and discards messages exceeding a certain tolerance level. Otherwise an attacker can try to delay the synchronization messages to make the clock estimate loose time.

The periodic broadcast packets and clock verification methods detect all the possible timing attacks within secure limits when only the protected counter rate or only the broadcast delay can be tampered with. The detection accuracy depends on the chosen timing parameters, the stability of the counter, and the accuracy of the knowledge of the utilized networking technologies. Successful attacks can only be implemented if the attacker is able to modify the counter rate unlimitedly and delay the broadcast transmission simultaneously. Such an attack can technically be prevented by randomly reopening the two-way data link during the betting session or integrating the wireless network transceiver or e.g. a reference Global Positioning System (GPS) receiver inside the protected area. In that case, tampering with the counter rate also prevents synchronization to the wireless medium, which is detected as a connection failure of the betting terminal. As a non-technical protection, the betting organizer should keep a log on winnings so that if a user starts succeeding suspiciously often, the authenticity of her activities can be verified.

7. SUMMARY OF PUBLICATIONS

This chapter summarizes the contents of the publications and clarifies the contribution of the author. The co-authors have seen and agree with the descriptions. The publications are based on the work of the author during the years between 2000 and early 2006. None of the publications have previously been used as a part of a doctoral thesis. The organization of the publications follows the presentation orders of Chapter 5 and Chapter 6.

The publications [P1–P8] focus on the design and implementation of hardware support for WLAN cryptography. These publications can be categorized into two sections. The publications [P1–P5] concentrate on the implementation of symmetric-key cryptography, focusing on confidentiality and data authentication procedures. The implementation of public-key schemes, including entity authentication, is addressed in [P6–P8].

The publications [P9, P10] propose and implement novel WLAN security designs. The focus is on the usage and integration of the cryptographic components of the other publications in full applications. The security designs follow the recommended design practices and take into account the security problems of WLANs reviewed in the introductory part of the Thesis.

Publication [P1] discusses further the importance of MAC layer security services and their hardware implementation in WLANs. Reconfigurable platforms, 3DES, and AES are proposed to be used for securing WLANs. Hardware architectures for the algorithms are designed and implemented on a TUTWLAN prototyping platform. The implementation results are compared and the suitability of the algorithms for WLAN devices is discussed.

The author chose the examined algorithms in co-operation with Docent Dr. Marko Hännikäinen, Prof. Timo D. Hämäläinen, and Mr. Markku Niemi, M.Sc. The author designed the hardware architectures for the algorithms, carried out the implementations, and analyzed the results. The publication was written by the author. Docent Dr. Marko Hännikäinen and Prof. Timo D. Hämäläinen provided comments and revised

the text. Mr. Markku Niemi, M.Sc., provided technical details. Prof. Jukka Saarinen improved the writing style.

Publication [P2] describes the design and implementation of a compact and energy-efficient AES hardware architecture. The architecture is specifically designed for supporting the modes of operation utilized in IEEE 802.15.4. Due to the convergence of the security designs, the hardware architecture is suited for the security processing of other WLANs as well.

The author designed and implemented the hardware architectures presented in the publication, analyzed the results, and wrote the publication. Docent Dr. Marko Hännikäinen provided technical details and suggested improvements for the draft version of the publication. Prof. Timo D. Hämäläinen improved the writing style.

Publication [P3] presents novel permutation architectures suited for compact AES implementations in WLAN devices. Both register-based and memory-based architectures are considered. The architectures can be exploited in hardware implementations and in ASIPs. The 8-bit permutation architecture has successfully been used for developing a compact and low-power AES encryption core in [96].

Dr. Tuomas Järvinen, Mr. Perttu Salmela, M.Sc, and the author co-designed the architectures, Dr. Tuomas Järvinen being the main architect. The author suggested and motivated the research on AES, reviewed the related work on AES, and provided reference VHDL descriptions for the verification of the permutation designs. Dr. Tuomas Järvinen and the author were the main contributors to the text. The research on permutation architectures for encryption algorithms was suggested by Prof. Jarmo Takala.

Publication [P4] is the first publication of the author. It presents hardware architectures and implementations of RC4 and a mixing function named IWEP (see Appendix). The implementations are targeted at the first TUTWLAN prototyping platform [132].

The author designed and implemented the hardware architectures, analyzed the results, and wrote the publication. Docent Dr. Marko Hännikäinen and Prof. Timo D. Hämäläinen gave comments during the research and revised the text. Prof. Jukka Saarinen improved the writing style.

Publication [P5] presents the ASIP designs of the Thesis. The need of efficient cryptographic hardware implementations for WLANs with certain level of reconfigurability is discussed. The publication presents TTA and its design environment and detailed parallelism analyses for RC4 and AES. In addition to efficiently supporting

the ciphers, the ASIPs are maintained general-purpose for enabling also other applications in them. Furthermore, most of the special operations can be utilized in other algorithms and applications. The developed AES ASIP supports RC4 and is well-suited for 802.11i devices.

The author performed the parallelism analyses, designed and implemented the ASIPs, and analyzed the results. Mr. Jari Heikkinen, M.Sc., provided hints for using the TTA design flow and synthesized the final TTA configurations, developed by the author, for achieving accurate area results. The author wrote the publication. The results and their presentation were commented and revised by Docent Dr. Marko Hännikäinen and Prof. Timo D. Hämäläinen.

Publication [P6] concentrates on the hardware acceleration of the SRP protocol. Algorithms and implementation methods for modular exponentiations are discussed. The publication contains a full implementation of SRP, consisting of a hardware accelerator, its interface, and software routines. The implementation is carried out on a TUTWLAN prototyping platform that contains a FPGA and an embedded processor. The implementation is compared to a software implementation and a significant performance improvement is achieved. Further improvements for the accelerator are discussed.

Mr. Peter Groen, M.Sc, and the author co-designed the SRP implementation architecture, including the hardware and software components, analyzed the results, and wrote the publication. Mr. Peter Groen, M.Sc, carried out the implementation. The author provided the initial software routines as well as supervised and assisted with the implementation. Prof. Ben Juurlink improved the text. Prof. Timo D. Hämäläinen provided comments during the research.

Publication [P7] presents the design and implementation of a hardware architecture for computing large modular exponentiations using Montgomery's algorithm. The importance of modular exponentiation in cryptographic protocols, including those of WLANs, as well as its efficient implementation are discussed. The publication improves the exponentiation implementation of [P6] using the same TUTWLAN platform.

The author specified the functional requirements and designed the improvements for the previous implementation as well as supervised and assisted with the implementation. The implementation was carried out by Mr. Ning Liu, M.Sc. The author analyzed the results and wrote the publication. Docent Dr. Marko Hännikäinen suggested improvements for the draft version of the publication. Prof. Timo D. Hämäläinen improved writing style.

Publication [P8] presents a compact accelerator architecture for modular exponentiations using Montgomery's algorithm. The implementation is carried out on a modern FGPA. The publication discusses different hardware architectures for modular exponentiation as well as the role of FPGAs as implementation technologies.

The author and Mr. Timo Alho, M.Sc., co-designed the accelerator, Mr. Timo Alho, M.Sc., being the main designer. The implementation was carried out by Mr. Timo Alho, M.Sc. The author provided background information and references, supervised the research, defined the functional requirements, provided technical assistance, and analyzed the results. The author and Mr. Timo Alho, M.Sc., wrote the publication. The author was the main contributor to the text. Docent Dr. Marko Hännikäinen and Prof. Timo D. Hämäläinen suggested improvements for the draft version of the publication.

Publication [P9] proposes and implements the ESL for Bluetooth. The security is improved by adding AES-based confidentiality and data authentication mechanisms. The prototype implementation is carried out on the same TUTWLAN platform as [P6]. The hardware part of the prototype includes an AES implementation with support for various modes of operation. Compared to the original Bluetooth design, higher security level is achieved at significantly lower hardware resources and higher performance. In addition, as the proposed design includes AES and the CCM mode, it enables the further convergence of the security designs of WLANs.

The author designed ESL, analyzed the implementation results, and wrote the publication. The author designed and implemented the security processing hardware of the prototype. The rest of the prototype was co-designed with Mr. Ning Liu, M.Sc., and Mr. Risto Sterling, M.Sc., who carried out the rest of the implementation. The functional requirements were outlined with Docent Dr. Marko Hännikäinen, who also suggested improvements for the draft version of the publication. Prof. Timo D. Hämäläinen improved the writing style.

Publication [P10] fully covers the design and prototype implementation of the novel wireless RTB application, which is specifically considered well-suited for providing local services using WLANs. The design is argued to overcome the processing and scalability limitations of the traditional online betting systems. The novel situation is achieved by broadcasting bet announcements, time-stamping and storing placements locally at user terminals, and collecting them only after the target event has been finished. Whereas solving the capacity problems of online systems, this so-called *offline* operation makes RTB a challenging application due to its security requirements. The publication specifies methods for time-stamping, storing bets, and verifying the au-

thenticity of user operations. Cryptographic hardware implementations are required for efficiently supporting real-time processing and securing RTB terminals. The designed mechanisms are considered advantageous for WLANs in general, beyond this specific application.

The author has been the main architect for the security design of the RTB application. The security-related requirements were defined in co-operation with Docent Dr. Marko Hännikäinen and Prof. Timo D. Hämäläinen. The rest of the functionalities of the RTB application and the prototype implementation were designed by the author, Docent Dr. Marko Hännikäinen, and Prof. Timo D. Hämäläinen. The prototype was implemented by the author, Mr. Mauri Kuorilehto, M.Sc., Mr. Jouni Sonninen, and Mr. Ari Suonpää, M.Sc. The author developed and simulated a model of an online system, organized test sessions, analyzed results, and wrote the publication. Docent Dr. Marko Hännikäinen and Prof. Timo D. Hämäläinen suggested improvements for the draft version of the publication. Mr. Riku Soininen, Lic.Tech., and Mr. Risto Rautee, M.Sc., provided technical details.

8. CONCLUSIONS

Cryptographic security services are required in WLANs and their data link layer, which has been recognized in standardization, industry, and research communities. Through the intensive research and development, the level of WLAN security has clearly improved as enhanced mechanisms for entity authentication, key management, data authentication, and confidentiality have been developed. However, the price has been the increased amount of security-related processing as services which have previously been disregarded have now been included in standard WLAN technologies. This combined with the processing capacity requirements of cryptographic procedures, increasing network data rates, and the demand for low cost and high usability requires efficient security designs and implementations. In this Thesis, efficient cryptographic hardware architectures as well as cryptographic security designs that support efficient implementations were developed for securing WLANs.

Because of the commercial success, IEEE and its standards, specifically IEEE 802.11, have been driving the research and development of WLAN security. Due to the completely flawed 802.11, IEEE has become cautious in its security designs. Particularly, any single solution has not been adopted for entity authentication and key agreement throughout the newest WLAN designs. The technologies either support multiple protocols or the tasks have been left unspecified and as responsibilities of higher protocol layers. This is partly because of supporting different usage scenarios as well as interoperability and convergence with other technologies and also because redesigning and updating fielded devices would once again be required if a specified protocol turned out weak. Despite (or because of) the numerous problems, 802.11 has also had positive impacts due to the immense attention it has achieved. People's awareness of information security in general has increased and the security research on 802.11 has produced new results and more reliable mechanisms.

In contrast to various alternatives for entity authentication, AES with CCM is becoming the default solution for providing data authentication and confidentiality in WLANs. This, in turn, allows the convergence of the security processing implementations in WLAN devices, supporting low-cost implementations and effective

resource-sharing between different WLAN technologies. As AES has been developed through an extensive, public process involving recognized experts in cryptography and CCM constitutes of well-known and scrutinized components, these choices for WLANs are examples of meeting good design practices.

Bluetooth is currently the only exception of this trend. It is also the only significant WLAN technology the security design of which has not significantly been revised despite of numerous vulnerabilities. In this Thesis, an improved security design for Bluetooth was presented. The design supports AES and the CCM mode, which enables efficient implementations and the convergence of Bluetooth with the security designs of other WLAN technologies.

AES has become the default choice also in other technologies and protocol layers beyond WLANs and their link layers. Hence, its inclusion in the WLAN link layers allows resource-sharing with higher protocol layers as well. For example, in addition to the link layer security, for end-to-end protection many WLAN devices run TLS, SSL, or IPsec which all can utilize AES in their security services. Furthermore, these higher layer *de facto* standards can share their entity authentication procedures and processing with the WLAN link layers.

Design and implementation of hardware architectures for AES and its modes of operation in WLANs was one of the main themes in the research work reported in this Thesis. Various architectures were developed for providing different combinations of performance and resource consumption. In all the designs, a good balance was maintained between the quantities, which makes the architectures well-suited for embedded WLAN devices. In addition to AES, RC4 and 3DES were examined and efficient hardware architectures for them were developed. Despite that 3DES is not currently utilized at the link layers of the commercially significant WLANs, it is still widely used e.g. in TLS, SSL, and IPsec.

For entity authentication and key agreement in WLANs, the research focused on the design of hardware architectures for public-key protocols based on modular exponentiation. The acceleration of the SRP protocol, which provides a password-based entity (user) authentication mechanism, was specifically examined. It can be concluded that high performance requires a large amount of hardware resources for exponentiation. The symmetric-key algorithms of WLANs, which can also be used for entity authentication and key agreement, can be implemented with significantly lower resources and imply higher performance. By decreasing the exponentiation performance, the amount of computational resources can be reduced, which improves the efficiency. Since public-key algorithms are typically only used during connection

establishments in WLANs, extremely high performance is often not the most significant design goal. On the other hand, if public-key-based authentication occurs frequently, high performance for exponentiation can be required. This can be the case when digital signatures are used for broadcast authentication or when a WLAN terminal is traveling trough the coverage areas of different APs.

Because of the numerous discovered vulnerabilities as well as the support for different algorithms and protocols, reconfigurability would be beneficial in WLAN products. Software implementations do not provide high performance at reasonable costs and it is not feasible to upgrade ASICs in fielded devices. The utilization of software and ASIC solutions has required maintaining support for flawed mechanisms and trading security to lower processing requirements in the revised WLAN security designs. The TTA processor architecture examined in this Thesis can be used for efficient cryptographic implementations while providing domain specific reconfigurability. With the further development of power consumption aspects, FPGA-type technologies may also become feasible platforms in WLAN devices [4].

The development of WLAN technologies and their security has enabled new types of applications and services, e.g. the RTB application of this Thesis. However, the security services provided by WLANs and their link layers are not always enough for meeting all the security requirements of applications. In addition to secure wireless data transfer, RTB requires mechanisms for maintaining time synchronization, time stamping, and storing data. Solutions for these purposes were developed considering the constraints of WLANs. The solutions can be useful in other WLAN applications as well.

Despite that cryptography can be used for securing information systems including WLANs, it cannot provide the security alone. Numerous attacks against information systems are currently performed e.g. through social engineering, which can bypass any cryptographic protection mechanism. Therefore, as commonly stated, it must be remembered that security is not a product that can be finalized and sold. It is a continuing process that requires constant research, development, management, and education.

BIBLIOGRAPHY

[1] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, "Extensible authentication protocol (EAP)," RFC 3748, June 2004.

[2] B. Aboba and D. Simon, "PPP EAP TLS authentication protocol," RFC 2716, Oct. 1999.

[3] B. Aboba. (2001, May) WEP2 security analysis. IEEE doc 802.11-00/253. [Online]. Available: http://www.drizzle.com/∼aboba/IEEE/11-01-253r0-I-WEP2SecurityAnalysis.ppt

[4] (2006, Aug.) Actel brings portable market in from the cold with industry's lowest power FPGA family. Press Release. Actel Corporation. [Online]. Available: http://www.actel.com/company/press/2006pr/IGLOO.html

[5] *ORiNOCO WEPplus Whitepaper*, Agere Systems, Inc., 2001.

[6] (2005) AirSnarf website. [Online]. Available: http://airsnarf.shmoo.com

[7] T. Alho, P. Hämäläinen, M. Hännikäinen, and T. D. Hämäläinen, "Compact modular exponentiation accelerator for modern FPGA devices," 2006, unpublished (submitted to International Journal of Intelligent Automation and Soft Computing).

[8] (2005) HardCopy II structured ASICs: ASIC gain without the pain. Altera. [Online]. Available: http://www.altera.com/products/devices/hardcopyii

[9] (2006) Altera website. [Online]. Available: http://www.altera.com

[10] G. Apostolopoulos, V. Peris, P. Pradhan, and D. Saha, "Securing electronic commerce: Reducing the SSL overhead," *IEEE Network*, vol. 14, no. 4, pp. 8–16, 2000.

[11] W. A. Arbaugh. (2001, May) An inductive chosen plaintext attack against WEP/WEP2. IEEE doc 802.11-01/230. [Online]. Available: http://www.cs.umd.edu/∼waa/attack/v3dcmnt.htm

[12] ——, "Wireless security is different," *IEEE Computer*, vol. 36, no. 8, pp. 99–101, 2003.

[13] W. A. Arbaugh, N. Shankar, Y. C. J. Wan, and K. Zhang, "Your 802.11 wireless network has no clothes," *IEEE Wireless Communications*, vol. 9, no. 6, pp. 44–51, 2002.

[14] D. I. Axiotis, T. Al-Gizawi, K. Peppas, E. N. Protonotarios, F. I. Lazarakis, C. Papadias, and P. I. Philippopoulos, "Services in interworking 3G and WLAN environments," *IEEE Wireless Communications*, vol. 11, no. 5, pp. 14–20, 2004.

[15] D. Bae, J. Kim, S. Park, and O. Song, "Design and implementation of IEEE 802.11i architecture for next generation WLAN," in *Proc. 1st SKLOIS Conf. Information Security and Cryptology (CISC 2005), Beijing, China, Dec. 15–17, 2005*, ser. Lecture Notes in Computer Science, D. Feng, D. Lin, and M. Yung, Eds., vol. 3822. Springer-Verlag, 2005, pp. 346–357.

[16] P. Bahl, W. Russel, Y.-M. Wang, A. Balachandran, and G. M. Voelker, "PAWNs: Satisfying the need for ubiquitous secure connectivity and location services," *IEEE Wireless Communications*, vol. 9, no. 1, pp. 40–48, 2002.

[17] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "Recommendation for key management—part 1: General," National Institute of Standards and Technology (NIST), Special Publication 800-57, Aug. 2005.

[18] W. C. Barker, "Recommendation for the Triple Data Encryption Algorithm (TDEA) block cipher," National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, Special Publication 800-67, Version 1, May 2004.

[19] L. Batina, S. B. Örs, B. Preneel, and J. Vandewalle, "Hardware architectures for public key cryptography," *Elsevier Integration, the VLSI Journal*, vol. 34, no. 1-2, pp. 1–64, 2003.

[20] S. M. Bellovin, "Problem areas for the IP security protocols," in *Proc. 6th USENIX Security Symp.*, San Jose, CA, USA, July 22–25, 1996, pp. 1–16.

[21] (2003, June) Bluetooth SIG presents new specification, and two implementation guides. Press Release. Bluetooth SIG. [Online]. Available: http://www.bluetooth.com/news/sigreleases.asp?A=2&PID=870&ARC=1&ofs=20

[22] (2005) Bluetooth Special Interest Group (SIG) website. [Online]. Available: http://www.bluetooth.com

[23] "Simple pairing whitepaper," White Paper V10r00, Bluetooth SIG Core Specification Working Group, Aug. 2006.

[24] *Specification of the Bluetooth System*, Bluetooth Special Interest Group (SIG) Std. 2.0 + EDR, 2004.

[25] (2005) The official Bluetooth membership website. [Online]. Available: http://www.bluetooth.org

[26] T. Blum and C. Paar, "Montgomery modular exponentiation on reconfigurable hardware," in *Proc. 14th IEEE Symp. Computer Arithmetic (ARITH 14)*, Adelaide, Australia, Apr. 14–16, 1999, pp. 70–77.

[27] ——, "High radix Montgomery modular exponentiation on reconfigurable hardware," *IEEE Trans. Computers*, vol. 50, no. 7, pp. 759–764, 2001.

[28] N. Borisov, I. Goldberg, and D. Wagner, "Intercepting mobile communications: the insecurity of 802.11," in *Proc. 7th Annu. Int. Conf. Mobile Computing and Networking*, Rome, Italy, July 16–21, 2001, pp. 180–189.

[29] B. Brown, "802.11: The security differences between b and i," *IEEE Potentials*, vol. 22, no. 4, pp. 23–27, 2003.

[30] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko, "Diameter base protocol," RFC 3588, Sept. 2003.

[31] N. Cam-Winget, R. Housley, D. Wagner, and J. Walker, "Security flaws in 802.11 data link protocols," *Communications of the ACM*, vol. 46, no. 5, pp. 35–39, 2003.

[32] J. Carlson, B. Aboba, and H. Haverinen, "EAP SRP-SHA1 authentication protocol," IETF Internet Draft, July 2001, work in progress. [Online]. Available: http://www.drizzle.com/∼aboba/IEEE/draft-ietf-pppext-eap-srp-03.txt

[33] J.-C. Chen, M.-C. Jiang, and Y.-W. Liu, "Wireless LAN security and IEEE 802.11i," *IEEE Wireless Communications*, vol. 12, no. 1, pp. 27–36, 2005.

[34] P. Chodowiec and K. Gaj, "Very compact FPGA implementation of the AES algorithm," in *Proc. 5th Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003), Cologne, Germany, Sept. 8–10, 2003*, ser. Lecture Notes in Computer Science, C. D. Walter, C. K. Koc, and C. Paar, Eds., vol. 2779. Springer-Verlag, 2003, pp. 319–333.

[35] P. Chodowiec, P. Khuon, and K. Gaj, "Fast implementations of secret-key block ciphers using mixed inner- and outer-round pipelining," in *Proc. 2001 ACM/SIGDA Int. Symp. Field Programmable Gate Arrays (FPGA'01)*, Monterey, CA, USA, Feb. 11–13, 2001, pp. 94–102.

[36] P. Chown, "Advanced Encryption Standard (AES) ciphersuites for Transport Layer Security (TLS)," RFC 3268, June 2002.

[37] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA*. West Sussex, England: John Wiley & Sons, Inc., 1998.

[38] H. Corporaal and M. Arnold, "Using transport triggered architectures for embedded processor design," *Integrated Computer-Aided Engineering*, vol. 5, no. 1, pp. 19–38, 1998.

[39] (1998, Aug.) SAFER+: Cylink corporation's submission for the advanced encryption standard. Presentation at the First Advanced Encryption Standard Candidate Conf. (AES1), Ventura, CA, USA. Cylink. [Online]. Available: http://csrc.nist.gov/CryptoToolkit/aes/round1/conf1/saferpls-slides.pdf

[40] A. Dandalis and V. K. Prasanna, "An adaptive cryptographic engine for internet protocol security architectures," *ACM Trans. Design Automation of Electronic Systems*, vol. 9, no. 3, pp. 333–353, 2004.

[41] A. Dandalis, V. K. Prasanna, and J. D. P. Rolim, "A comparative study of performance of AES final candidates using FPGAs," in *Proc. 3rd AES Candidate Conference (AES3)*, New York, NY, USA, Apr. 13–14, 2000. [Online]. Available: http://csrc.nist.gov/encryption/aes/round2/conf3/papers/23-adandalis.pdf

[42] S. Das, F. Anjum, Y. Ohba, and A. K. Salkints, "Security issues in wireless IP networks," in *Mobile Internet: Enabling Technologies and Services*, A. K. Salkintzis, Ed. USA: CRC Press, 2004, ch. 9.

[43] T. Dierks and C. Allen, "The TLS protocol 1.0," RFC 2246, Jan. 1999.

[44] W. Diffie and M. Hellman, "Exhaustive cryptanalysis of the NBS data encryption standard," *IEEE Computer*, vol. 10, no. 6, pp. 74–84, 1977.

[45] ——, "New directions in cryptography," *IEEE Trans. Information Theory*, no. 22, pp. 644–654, 1976.

[46] T. Dismukes. (2002, July) Wireless security blackpaper. Ars Technica. [Online]. Available: http://arstechnica.com/articles/paedia/security.ars

[47] R. Doud. (1999, Apr.) Hardware crypto solutions boost VPN. EETimes. [Online]. Available: http://www.eetimes.com/news/latest/showArticle.jhtml?articleID=18301583

[48] M. Dworkin, "Recommendation for block cipher modes of operation—methods and techniques," National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, Special Publication 800-38A, Dec. 2001.

[49] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA implementation and performance evaluation of the AES block cipher candidate algorithm finalists," in *Proc. 3rd AES Candidate Conference (AES3)*, New York, NY, USA, Apr. 13–14, 2000. [Online]. Available: http://csrc.nist.gov/encryption/aes/round2/conf3/papers/08-aelbirt.pdf

[50] A. J. Elbirt and C. Paar, "An instruction-level distributed processor for symmetric-key cryptography," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 5, pp. 468–480, 2005.

[51] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," *IEEE Trans. VLSI Systems*, vol. 9, no. 4, pp. 545–557, 2001.

[52] Y. Eslami, A. Sheikholeslami, P. G. Gulak, S. Masui, and K. Mukaida, "An area-efficient universal cryptography processor for smart cards," *IEEE Trans. VLSI Systems*, vol. 14, no. 1, pp. 43–56, 2006.

[53] *Broadband Radio Access Networks (BRAN); HIgh PErformance Radio Local Area Network (HIPERLAN) Type 1; Functional specification*, ETSI Std. EN 300 652 V1.2.1, 1998.

[54] *Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Data Link Control (DLC) Layer; Part 1: Basic Data Transport Functions*, ETSI Std. TS 101 761-1 V1.2.1, 2000.

[55] *Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Data Link Control (DLC) Layer; Part 2: Radio Link Control (RLC) sublayer*, ETSI Std. TS 101 761-2 V1.3.1, 2002.

[56] (2005) ETSI BRAN website. [Online]. Available: http://portal.etsi.org/portal_common/home.asp?tbkey1=BRAN

[57] M. Falk, "Fast and secure roaming in WLAN," Master's thesis, Linköping University, Linköping, Sweden, 2004.

[58] *Computer Data Authentication*, Federal Information Processing Standards (FIPS) Std. 113, 1985.

[59] *Data Encryption Standard*, Federal Information Processing Standards (FIPS) Std. 46-3, 1999.

[60] *Secure Hash Standard*, Federal Information Processing Standards (FIPS) Std. 180-2, 1999.

[61] *Advanced Encryption Standard (AES)*, Federal Information Processing Standards (FIPS) Std. 197, 2001.

[62] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong authentication for RFID systems using the AES algorithm," in *Proc. 6th Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004), Boston, USA, Aug. 11–13, 2004*, ser. Lecture Notes in Computer Science, M. Joye and J.-J. Quisquater, Eds., vol. 3156. Springer-Verlag, 2004, pp. 357–370.

[63] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "AES implementation on a grain of sand," *IEE Proc. Information Security*, vol. 152, no. 1, pp. 13–20, 2005.

[64] N. Ferguson. (2002, Jan.) Michael: an improved MIC for 802.11 WEP. IEEE doc 802.11-02/020r0. [Online]. Available: http://grouper.ieee.org/groups/802/11/Documents/DocumentHolder/2-020.zip

[65] N. Ferguson and B. Schneier, *Practical Cryptography*. Indianapolis, IN, USA: Wiley Publishing, Inc., 2003.

[66] E. B. Fernandez, S. Rajput, M. VanHilst, and M. M. Larrondo-Petrie, "Some security issues of wireless systems," in *Revised papers from 5th IEEE Int. Symp. and School on Advanced Distributed Systems (ISSADS 2005),*

Guadalaraja, Mexico, Jan. 24–28, 2005, ser. Lecture Notes in Computer Science, F. F. Ramos, V. L. Rosillo, and H. Unger, Eds., vol. 3563. Springer-Verlag, 2005, pp. 388–396.

[67] V. Fischer, "Realization of the round 2 AES candidates using Altera FPGA," in *Proc. 3rd AES Candidate Conference (AES3)*, New York, NY, USA, Apr. 13–14, 2000. [Online]. Available: http://csrc.nist.gov/encryption/aes/round2/conf3/papers/24-vfischer.pdf

[68] V. Fischer and M. Drutarovsky, "Two methods of Rijndael implementation in reconfigurable hardware," in *Proc. 3rd Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), Paris, France, May 14–16, 2001*, ser. Lecture Notes in Computer Science, C. K. Koc, D. Naccache, and C. Paar, Eds., vol. 2162. Berlin, Germany: Springer-Verlag, 2001, pp. 77–92.

[69] G. Fleishman. (2003, Nov.) Weakness in passphrase choice in WPA interface. Wi-Fi Net News. [Online]. Available: http://wifinetnews.com/archives/002452.html

[70] S. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the key scheduling algorithm of RC4," in *Revised Papers from 8th Annu. Workshop on Selected Areas in Cryptography (SAC 2001), Toronto, Ontario, Canada, Aug. 16–17, 2001*, ser. Lecture Notes in Computer Science, S. Vaudenay and A. M. Youssef, Eds., vol. 2259. Springer-Verlag, 2001, pp. 1–24.

[71] S. R. Fluhrer and S. Lucks, "Analysis of the E0 encryption system," in *Proc. 8th Annu. Int. Workshop on Selected Areas in Cryptography (SAC 2001)*, Toronto, Ontario, Canada, Aug. 16–17, 2001, pp. 38–48.

[72] P. Funk and S. Blake-Wilson, "EAP tunneled TLS authentication protocol (EAP-TTLS)," IETF Internet Draft, July 2004, work in progress. [Online]. Available: http://bgp.potaroo.net/ietf/all-ids/draft-ietf-pppext-eap-ttls-05.txt

[73] K. Gaj and P. Chodowiec, "Comparison of the hardware performance of the AES candidates using reconfigurable hardware," in *Proc. 3rd AES Candidate Conference (AES3)*, New York, NY, USA, Apr. 13–14, 2000. [Online]. Available: http://csrc.nist.gov/encryption/aes/round2/conf3/papers/22-kgaj.pdf

[74] ——, "Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays," in *Proc. Cryptographer's Track at RSA Conf. 2001 (CT-RSA 2001)*, San Francisco, CA, USA, Apr. 8–12, 2001, pp. 84–99.

[75] C. Gehrmann, "Bluetooth security white paper," Bluetooth SIG Security Expert Group, White Paper 1.01, May 2002.

[76] C. Gehrmann and K. Nyberg, "Enhancements to Bluetooth baseband security," in *Proc. Nordic Workshop on Secure IT-Systems (NordSec 2001)*, Copenhagen, Denmark, Nov. 1–2, 2001, pp. 39–53.

[77] C. Gentry and Z. Ramzan, "Provable cryptographic security and its applications to mobile wireless computing," *Wireless Personal Communications*, vol. 29, pp. 203–191, 2004.

[78] I. Goldberg and D. Wagner. (1996, May) Architectural considerations for cryptanalytic hardware. [Online]. Available: http://www.mirrors.wiretapped.net/security/cryptography/literature/cracking-des/chap-10_local.html

[79] R. E. Gonzalez, "Xtensa: A configurable and extensible processor," *IEEE Micro*, vol. 20, no. 2, pp. 60–70, 2000.

[80] T. Good and M. Benaissa, "AES on FPGA from the fastest to the smallest," in *Proc. 7th Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005), Edinburgh, UK, Aug. 29–Sept. 1, 2005*, ser. Lecture Notes in Computer Science, J. Rao and B. Sunar, Eds., vol. 3659. Springer-Verlag, 2005, pp. 427–440.

[81] J. Goodman and A. P. Chandrakasan, "An energy-efficient reconfigurable public-key cryptography processor," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 11, pp. 1808–1820, 2001.

[82] S. Grech and J. Nikkanen. (2004, Nov.) A security analysis of Wi-Fi protected access. Presentation at Nordsec 2004 Workshop, Espoo, Finland. [Online]. Available: http://www.tml.tkk.fi/Nordsec2004/Presentations/grech.pdf

[83] J. Grosschädl, "Instruction set extension for long integer modulo arithmetic on RISC-based smart cards," in *Proc. 14th Symp. Computer Architecture and High Performance Computing (SCAB-PAD'02)*, Vitoria, Brazil, Oct. 28–30, 2002, pp. 13–19.

[84] M. Gruteser and D. Grunwald, "Enhancing location privacy in wireless LAN through disposable interface identifiers: A quantitative analysis," *Mobile Networks and Applications*, vol. 10, pp. 315–325, 2005.

[85] F. Guo and T.-C. Chiueh, "Sequence number-based MAC address spoof detection," in *Proc. 8th Int. Symp. Recent Advances in Intrusion Detection (RAID 2005), Seattle, WA, USA Sept. 7–9, 2005*, ser. Lecture Notes in Computer Science, A. Valdes and D. Zamboni, Eds., vol. 3858. Springer-Verlag, 2005, pp. 309–329.

[86] F. K. Gurkaynak, A. Burg, N. Felber, W. Fichtner, D. Gasser, F. Hug, and H. Kaeslin, "A 2 Gb/s balanced AES crypto-chip implementation," in *Proc. 14th Great Lakes Symp. VLSI (GLSVLSI 2004)*, Boston, MA, USA, Apr. 26–28, 2004, pp. 39–44.

[87] P. Gutmann, "PKI: It's not dead, just resting," *IEEE Computer*, vol. 35, no. 8, pp. 41–49, 2002.

[88] C. T. Hager and S. F. Midkiff, "An analysis of Bluetooth security vulnerabilities," in *Proc. IEEE Wireless Comm. and Networking Conf. (WCNC 2003)*, New Orleans, LA, USA, Mar. 16–20, 2003, pp. 1825–1831.

[89] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao, "The Chimaera reconfigurable functional unit," *IEEE Trans. VLSI Systems*, vol. 12, no. 2, pp. 206–217, 2004.

[90] H. Haverinen, "Interworking between wireless LAN and GSM/UMTS cellular networks: Network access control, mobility management and security considerations," Ph.D. dissertation, Tampere Univ. of Tech., Tampere, Finland, Dec. 2004.

[91] C. He and J. C. Mitchell, "Analysis of the 802.11i 4-way handshake," in *Proc. 2004 ACM Workshop on Wireless Security (WiSe'04)*, Philadelphia, PA, USA, Oct. 1, 2004, pp. 43–50.

[92] ——, "Security analysis and improvements for IEEE 802.11i," in *Proc. 12th Annu. Network and Distributed System Security Symp. (NDSS'05)*, San Diego, CA, USA, Feb. 3–4, 2005, pp. 90–110.

[93] C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell, "A modular correctness proof of IEEE 802.11i and TLS," in *Proc. 12th ACM Conf. Computer and Comm. Security (CCS'05)*, Alexandria, VA, USA, Nov. 7–11, 2005, pp. 2–15.

[94] J. Heo and C. S. Hong, "An efficient and secured media access mechanism using the intelligent coordinator in low-rate WPAN environment," in *Proc. 9th*

Int. Conf. Knowledge-Based Intelligent Information and Engineering Systems (KES 2005), Melbourne, Australia Sept. 14–16, 2005, ser. Lecture Notes in Computer Science, R. Khosla, R. J. Howlett, and L. C. Jain, Eds., vol. 3682. Springer-Verlag, 2005, pp. 470–476.

[95] M. Hermelin and K. Nyberg, "Correlation properties of the Bluetooth combiner," in *Proc. Int. Conf. Information Security and Cryptology (ICISC'99)*, Seoul, Korea, Dec. 9–10, 2000, pp. 17–29.

[96] P. Hämäläinen, T. Alho, M. Hännikäinen, and T. D. Hämäläinen, "Design and implementation of low-area and low-power AES encryption hardware core," in *Proc. 9th Euromicro Conf. Digital System Design (DSD 2006)*, Cavtat, Croatia, Aug. 30–Sept. 1, 2006, pp. 577–583.

[97] P. Hämäläinen, M. Hännikäinen, and T. D. Hämäläinen, "Method and arrangement for real-time betting with an off-line terminal (Menetelmä ja järjestelmä reaaliaikaiseksi veikkaamiseksi off-linepäätteen avulla)," International Patent FI20 031 909, June 30, 2005.

[98] ——, "Security enhancement layer for Bluetooth," in *Wireless Security and Cryptography: Specifications and Implementations*, N. Sklavos and X. Zhang, Eds. CRC Press, 2007, to appear.

[99] P. Hämäläinen, M. Hännikäinen, T. D. Hämäläinen, H. Corporaal, and J. Saarinen, "Implementation of encryption algorithms on transport triggered architectures," in *Proc. 2001 IEEE Int. Symp. Circuits and Systems (ISCAS 2001)*, vol. 4, Sydney, Australia, May 6–9, 2001, pp. 726–729.

[100] P. Hämäläinen, M. Hännikäinen, T. D. Hämäläinen, and J. Saarinen, "Configurable hardware implementation of Triple-DES encryption algorithm for wireless local area network," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP 2001)*, vol. 2, Salt Lake City, UT, USA, May 7–11, 2001, pp. 1221–1224.

[101] P. Hämäläinen, M. Hännikäinen, T. D. Hämäläinen, and R. Soininen, "Offline architecture for real-time betting," in *Proc. 2003 IEEE Int. Conf. Multimedia & Expo (ICME 2003)*, vol. 1, Baltimore, MD, USA, July 6–9, 2003, pp. 709–712.

[102] P. Hämäläinen, M. Hännikäinen, M. Niemi, and T. D. Hämäläinen, "Performance evaluation of secure remote password protocol," in *Proc. 2002 IEEE*

Int. Symp. Circuits and Systems (ISCAS 2002), vol. 3, Scottsdale, AZ, USA, May 26–29, 2002, pp. 29–32.

[103] P. Hämäläinen, M. Kuorilehto, T. Alho, M. Hännikäinen, and T. D. Hämäläinen, "Security in wireless sensor networks: Considerations and experiments," in *Proc. Embedded Computer Systems: Architectures, Modelling, and Simulation (SAMOS VI) Workshop - Special Session on Wireless Sensor Networks*, Samos, Greece, July 17–20, 2006, pp. 167–177.

[104] T. D. Hämäläinen, R. Rautee, M. Hännikäinen, and J. Rekonius, "Methods and arrangements for implementing betting with off-line terminals (Menetelmiä ja järjestelyjä vedonlyönnin toteuttamiseksi off-line-päätteillä)," International Patent FI113 713 B, May 31, 2004.

[105] M. Hännikäinen, "Design of quality of service support for wireless local area networks," Ph.D. dissertation, Tampere Univ. of Tech., Tampere, Finland, Nov. 2002.

[106] M. Hännikäinen, T. D. Hämäläinen, M. Niemi, and J. Saarinen, "Trends in personal wireless data communications," *Elsevier Computer Communications*, vol. 25, no. 1, pp. 84–99, 2002.

[107] A. Hodjat, P. Schaumont, and I. Verbauwhede, "Architectural design features of a programmable high throughput AES coprocessor," in *Proc. IEEE Int. Conf. Information Technology: Coding and Computing (ITCC 2004)*, vol. 2, Las Vegas, NV, USA, Apr. 4–6, 2004, pp. 498–502.

[108] A. Hodjat and I. Verbauwhede, "Speed-area trade-off for 10 to 100 Gbits/s throughput AES processor," in *Proc. 37th IEEE Asilomar Conf. Signals, Systems and Computers*, vol. 2, Asilomar, CA, USA,, Nov. 9–12, 2003, pp. 2147–2150.

[109] R. Housley and W. Arbaugh, "Security problems in 802.11-based networks," *Communications of the ACM*, vol. 46, no. 5, pp. 31–34, 2003.

[110] M.-S. Hwang, C.-C. Lee, J.-Z. Lee, and C.-C. Yang, "A secure protocol for Bluetooth piconets using elliptic curve cryptography," *Telecommunication Systems*, vol. 29, no. 3, pp. 165–180, 2005.

[111] T. Ichikawa, T. Kasuya, and M. Matsui, "Hardware evaluation of the AES finalists," in *Proc. 3rd AES Candidate Conference*

(AES3), New York, NY, USA, Apr. 13–14, 2000. [Online]. Available: http://csrc.nist.gov/encryption/aes/round2/conf3/papers/15-tichikawa.pdf

[112] *IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std. 802.11, 1999.

[113] *Supplement to IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications – Higher-Speed Physical Layer Extension in the 2.4 GHz Band*, IEEE Std. 802.11b-1999, 1999.

[114] *Supplement to IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications—High-Speed Physical Layer in the 5 GHz Band*, IEEE Std. 802.11a-1999, 1999.

[115] *IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 15.3: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wireless Personal Area Networks (WPANs)*, IEEE Std. 802.15.3, 2003.

[116] *IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPAN)*, IEEE Std. 802.15.4, 2003.

[117] *Supplement to IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications—Amendment 4: Further Higher Data Rate Extension in the 2.4 GHz Band*, IEEE Std. 802.11g-2003, 2003.

[118] *IEEE Standard for Information Technology— Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications—Amendment 6: Medium Access Control (MAC) Security Enhancements*, IEEE Std. 802.11i, 2004.

[119] *IEEE Standard for Local and Metropolitan Area Networks – Port-Based Network Access Control*, IEEE Std. 802.1X-2004, 2004.

[120] *IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)*, IEEE Std. 802.15.1, 2005.

[121] (2005) IEEE 802.11 working group website. [Online]. Available: http://grouper.ieee.org/groups/802/11

[122] (2005) IEEE 802.15 WPAN task group 3 website. [Online]. Available: http://www.ieee802.org/15/pub/TG3.html

[123] (2006) IEEE standards wireless zone website. [Online]. Available: http://standards.ieee.org/wireless

[124] (2005) IEEE 802.16 working group website. [Online]. Available: http://www.ieee802.org/16

[125] (2005, Feb.) 4Q 2004 WLAN market share report (abstract). Report Number IN0501945WL. [Online]. Available: http://www.instat.com

[126] "Information technology—open systems interconnections—basic reference model: The basic model," ISO/IEC 7498-1:1994(E), Nov. 1994.

[127] M. Jakobsson and S. Wetzel, "Security weaknesses in Bluetooth," in *Proc. Cryptographer's Track at RSA Conf. 2001 (CT-RSA 2001)*, San Francisco, CA, USA, Apr. 8–12, 2001, pp. 176–191.

[128] H. Y. Jang, J. H. Shim, J. H. Suk, I. C. Hwang, and J. R. Choi, "Compatible design of CCMP and OCB AES cipher using separated encryptor and decryptor for IEEE 802.11i," in *Proc. 2004 IEEE Int. Symp. Circuits and Systems (ISCAS 2004)*, vol. 3, Vancouver, Canada, May 23–26, 2004, pp. 645–648.

[129] J. Jonsson, "On the security of CTR + CBC-MAC," in *Proc. 9th Annu. Int. Workshop on Selected Areas in Cryptography (SAC 2002)*, St. John's, Newfoundland, Canada, Aug. 15–16, 2002, pp. 76–93.

[130] K. Järvinen, M. Tommiska, and J. Skyttä, "Comparative survey of high-performance cryptographic algorithm implementations on FPGAs," *IEE Proc. Information Security*, vol. 152, no. 1, pp. 3–12, 2005.

[131] J.-P. Kaps and C. Paar, "Fast DES implementation for FPGAs and its application to a universal key-search machine," in *Proc. 5th Annu. Int. Workshop Selected Areas in Cryptography (SAC'98), Kingston, Ontario, Canada, Aug. 17–18*, ser. Lecture Notes in Computer Science, S. E. Tavares and H. Meijer, Eds., vol. 1556. Springer-Verlag, 1999, pp. 234–247.

[132] M. Kari, M. Hännikäinen, T. D. Hämäläinen, J. Knuuttila, and J. Saarinen, "Configurable platform for a wireless multimedia local area network," in *Proc. 5th Int. Workshop on Mobile Multimedia Comm. (MoMuC'98)*, Berlin, Germany, Oct. 12–14, 1998, pp. 301–306.

[133] R. Karri and P. Mishr, "Minimizing energy consumption of secure wireless session with QoS constraints," in *Proc. IEEE Int. Conf. Comm. (ICC 2002)*, vol. 4, New York, NY, USA, Apr. 28–May 2, 2002, pp. 2053–2057.

[134] T. Karygiannis and L. Owens, "Wireless network security: 802.11, Bluetooth and handheld devices," National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, Special Publication 800-48, Nov. 2002.

[135] T. Kean and A. Duncan, "DES key breaking, encryption and decryption on the XC6216," in *Proc. 6th IEEE Symp. FPGAs and Custom Computing Machines (FCCM'98)*, Napa Valley, CA, USA, Apr. 15–17, 1998, pp. 310–311.

[136] S. Kent and R. Atkinson, "Security architecture for the Internet Protocol," RFC 2401, Nov. 1998.

[137] D. Kügler, "Man in the middle attacks on Bluetooth," in *Proc. 7th Int. Conf. Financial Cryptography (FC'03)*, Gosier, Guadeloupe, French West Indies, Jan. 27–30, 2003, pp. 149–161.

[138] H. Kim and S. Lee, "Design and implementation of a private and public key crypto processor and its application to a security system," *IEEE Trans. Consumer Electronics*, vol. 50, no. 1, pp. 214–224, 2004.

[139] I. Kim, S. Steele, and J. G. Koller, "A fully pipelined, 700MBytes/s DES encryption core," in *Proc. 9th Great Lakes Symp. VLSI (GLSVLSI 2004)*, Ann Arbor, MI, USA, Mar. 4–6, 1999, pp. 39–44.

[140] P. Kitsos, S. Goudevenos, and O. Koufopavlou, "VLSI implementations of the Triple-DES block cipher," in *Proc. 10th IEEE Int. Conf. Electronics, Circuits and Systems (ICECS'03)*, Shadah, United Arab Emirates, Dec. 14–17, 2003, pp. 76–79.

[141] P. Kitsos, G. Kostopoulos, N. Sklavos, and O. Koufopavlou, "Hardware implementation of the RC4 stream cipher," in *Proc. 46th IEEE Midwest Symp. Circuits and Systems (MWSCAS 2003)*, vol. 3, Cairo, Egypt, Dec. 27–30, 2003, pp. 1363–1366.

[142] P. Kitsos, N. Sklavos, K. Papadomanolakis, and O. Koufopavlou, "Hardware implementation of Bluetooth security," *IEEE Pervasive Computing*, vol. 2, no. 1, pp. 21–29, 2003.

[143] Korek, Submission to the Netstumbler.org forum, July 2004. [Online]. Available: http://www.netstumbler.org/showthread.php?t=12277

[144] D. Kotturi, S.-M. Yoo, and J. Blizzard, "AES crypto chip utilizing high-speed parallel pipelined architecture," in *Proc. 2005 IEEE Int. Symp. Circuits and Systems (ISCAS 2005)*, vol. 5, Kobe, Japan, May 23–26, 2005, pp. 4653–4656.

[145] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-hashing for message authentication," RFC 2104, Feb. 1997.

[146] K. Kreutzer, S. Malik, and A. R. Newton, "From ASIC to ASIP: The next design discontinuity," in *Proc. 2002 IEEE Int. Conf. Computer Design: VLSI in Computers and Processors (ICCD'02)*, Freiburg, Germany, Sept. 16–18, 2002, pp. 84–90.

[147] T. Krovets and P. Rogaway, "The OCB authenticated-encryption algorithm," CFRG Internet Draft, Mar. 2005, work in progress. [Online]. Available: http://www.cs.ucdavis.edu/∼rogaway/papers/draft-krovetz-ocb-00.txt

[148] P. Kukkala, V. Helminen, M. Hännikäinen, and T. D. Hämäläinen, "UML 2.0 implementation of an embedded WLAN protocol," in *Proc. 15th IEEE Int. Symp. Personal, Indoor and Mobile Radio Comm. (PIMRC 2004)*, vol. 2, Barcelona, Spain, Sept. 5–8, 2004, pp. 1158–1162.

[149] P. D. Kundarewich, S. J. E. Wilton, and A. J. Hu, "A CPLD-based RC4 cracking system," in *Proc. 1999 IEEE Canadian Conf. Electrical and Computer Engineering (CCECE'99)*, vol. 1, Edmonton, Canada, May 9–12, 1999, pp. 397–402.

[150] H. Kuo and I. Verbauwhede, "Architectural optimization for a 1.82Gbits/sec VLSI implementation of the AES Rijndael algorithm," in *Proc. 3rd Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), Paris, France, May 14–16, 2001*, ser. Lecture Notes in Computer Science, C. K. Koc, D. Naccache, and C. Paar, Eds., vol. 2162. Springer-Verlag, 2001, pp. 51–64.

[151] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," in *Proc. ACM/SIGDA 14th Int. Symp. Field Programmable Gate Arrays (FPGA 2006)*, Monterey, CA, USA, Feb. 22–24, 2006, pp. 21–30.

[152] M. Kuorilehto, M. Hännikäinen, and T. D. Hämäläinen, "A survey of application distribution in wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking—Special Issue on Ad Hoc Networks: Cross-Layer Issues*, no. 5, pp. 774–778, 2005.

[153] A. Laurie and B. Laurie. (2004, Oct.) Serious flaws in Bluetooth security lead to disclosure of personal data. The Bunker. [Online]. Available: http://www.thebunker.net/security/bluetooth.htm

[154] H. Leitold, W. Mayerwieser, U. Payer, K. C. Posch, R. Posch, and J. Wolkerstorfer, "A 155 Mbps Triple-DES network encryptor," in *Proc. 2nd Int. Workshop Cryptographic Hardware and Embedded Systems (CHES 2000), Worcester, MA, USA, Aug. 17–18, 2000*, ser. Lecture Notes in Computer Science, C. K. Koc and C. Paar, Eds., vol. 1965. Springer-Verlag, 2000, pp. 164–174.

[155] A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," *Journal of Cryptology*, vol. 14, no. 4, pp. 255–293, 2001.

[156] J. Leonard and W. H. Mangione-Smith, "A case study of partially evaluated hardware circuits: Key-specific DES," in *Proc. 7th Int. Workshop Field Programmable Logic and Applications (FPL'97), London, UK, Sept. 1–3, 1997*, ser. Lecture Notes in Computer Science, W. Luk, P. Y. K. Cheung, and M. Glesner, Eds., vol. 1304. Springer-Verlag, 1997, pp. 151–160.

[157] A. Levi, E. Cetintas, M. Aydos, C. K. Koc, and M. U. Caglayan, "Relay attacks on Bluetooth authentication and solutions," in *Proc. 19th Int. Symp. Computer*

and Information Science (ISCIS 2004), Antalya, Turkey, Oct. 27–29, 2004, pp. 278–288.

[158] A. Levi and E. Savas, "Performance evaluation of public-key cryptosystem operations in WTLS protocol," in *Proc. 8th IEEE Int. Symp. Computers and Communications (ISCC 2003))*, vol. 2, Antalya, Turkey, June 30–July 3, 2004, pp. 1245–1250.

[159] O. Levy and A. Wool. (2005) A uniform framework for cryptanalysis of the Bluetooth E_0 cipher. Cryptology ePrint Archive, Report 2005/107. [Online]. Available: http://eprint.iacr.org/2005/107

[160] M. Lewis and S. Simmons, "A VLSI implementation of a cryptographic processor," in *Proc. Canadian Conf. Electrical and Computer Engineering (CCECE 2003)*, Montreal, Canada, May 4–7, 2003, pp. 821–826.

[161] H. Li and M. Singhal, "A key establishment protocol for Bluetooth scatternets," in *Proc. 25th IEEE Int. Conf. Distributed Computing Systems Workshops (ICDCS 2005 Workshops)*, Columbus, OH, USA, June 6–10, 2005, pp. 610–616.

[162] E. Lopez-Trejo, F. Rodriguez-Henriquez, and A. Diaz-Perez, "An FPGA implementation of CCM mode using AES," in *Revised selected papers from 8th Int. Conf. Information Security and Cryptology (ICISC 2005), Seoul, Korea, Dec. 1–2, 2005*, ser. Lecture Notes in Computer Science, D. Won and S. Kim, Eds., vol. 3925. Springer-Verlag, 2005, pp. 322–334.

[163] G. Lowe, "An attack on the Needham-Schroeder public-key authentication protocol," *Information Processing Letters*, vol. 56, no. 3, pp. 131–133, 1995.

[164] Y. Lu, W. Meier, and S. Vaudenay, "The conditional correlation attack: A practical attack on Bluetooth encryption," in *Proc. 25th Annu. Int. Cryptology Conf., Advances in Cryptology (CRYPTO 2005), Santa Barbara, California, USA, Aug. 14–18, 2005*, ser. Lecture Notes in Computer Science, V. Shoup, Ed., vol. 3621. Springer-Verlag, 2005, pp. 97–117.

[165] Y. Lu and S. Vaudenay, "Faster correlation attack on Bluetooth keystream generator E0," in *Proc. 24th Annu. Int. Cryptology Conf., Advances in Cryptology (CRYPTO 2004), Santa Barbara, California, USA, Aug. 15–19, 2004*, ser. Lecture Notes in Computer Science, M. K. Franklin, Ed., vol. 3152. Springer-Verlag, 2004, pp. 407–425.

[166] I. Mantin, "A practical attack on the fixed RC4 in the WEP mode," in *Proc. 11th Int. Conf. Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2005), Chennai, India, Dec. 4–8, 2005*, ser. Lecture Notes in Computer Science, B. Roy, Ed., vol. 3788. Springer-Verlag, 2005, pp. 395–411.

[167] D. A. McGrew. (2002, Nov.) Counter mode security: Analysis and recommendations. [Online]. Available: http://www.mindspring.com/∼dmcgrew/ctr-security.pdf

[168] M. McLoone and J. V. McCanny, "A high performance FGPA implementation of DES," in *Proc. 2000 IEEE Workshop Signal Processing Systems (SiPS 2000)*, Lafayette, LA, USA, Apr. 11–13, 2000, pp. 374–383.

[169] ——, "High performance single-chip FPGA Rijndael algorithm implementations," in *Proc. 3rd Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), Paris, France, May 14–16, 2001*, ser. Lecture Notes in Computer Science, C. K. Koc, D. Naccache, and C. Paar, Eds., vol. 2162. Springer-Verlag, 2001, pp. 65–76.

[170] ——, "Generic architecture and semiconductor intellectual property cores for advanced encryption standard cryptography," *IEE Proc. Computers and Digital Techniques*, vol. 150, no. 4, pp. 239–244, 2003.

[171] A. J. Menezes, P. C. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[172] (2003, Feb.) Wireless industry: Up-coming market strategies and technology trends (abstract). Report R459-0013. MindBranch. [Online]. Available: http://www.mindbranch.com

[173] (2004, Apr.) U.S. wireless service provider market opportunities, market forecasts, and market strategies, 2004–2009 (abstract). Report R49-186. MindBranch. [Online]. Available: http://www.mindbranch.com

[174] A. Mishra and W. A. Arbaugh, "Initial security analysis of the IEEE 802.1X standard," University of Maryland, College Park, MD, USA, Tech. Rep. CS-TR-4328, Feb. 2002.

[175] V. Moen, H. Raddum, and K. J. Hole, "Weaknesses in the temporal key hash of WPA," *ACM SIGMOBILE Mobile Computing and Communications Review (MC^2R)*, vol. 8, no. 2, pp. 76–83, 2004.

[176] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[177] S. Morioka and A. Satoh, "A 10-Gbps full-AES crypto design with a twisted BDD S-box architecture," *IEEE Trans. VLSI Systems*, vol. 12, no. 7, pp. 686–691, 2004.

[178] S. Morris, "Recommendations to early implementers: Encrypting broadcast transmissions in Bluetooth piconets," Bluetooth SIG Security Expert Group, White Paper 1.0, May 2002.

[179] P. Mroczkowski. (2000, May) Implementation of the block cipher Rijndael using Altera FPGA. [Online]. Available: http://csrc.nist.gov/CryptoToolkit/aes/round2/comments/20000510-pmroczkowski.pdf

[180] T. Muller, "Bluetooth security architecture," Bluetooth SIG, White Paper 1.C.116/1.0, July 1999.

[181] (2006) National institute of standards and technology (NIST) website. [Online]. Available: http://www.nist.gov

[182] Y. Neuvo, "Future wireless technologies," in *Proc. 6th Circuits and Systems Symp. Emerging Technologies: Frontiers of Mobile and Wireless Communication*, vol. 1, Shanghai, China, May 31–June 2, 2004, pp. 1–3.

[183] D. Niculescu, "Communication paradigms for sensor networks," *IEEE Communications Magazine*, vol. 43, no. 3, pp. 116–122, 2005.

[184] J. Oates. (2004, June) Virus attacks mobiles via Bluetooth. The Register. [Online]. Available: http://www.theregister.co.uk/2004/06/15/symbian_virus

[185] D. Oliva, R. Buchty, and N. Heintze, "AES and the Cryptonite crypto processor," in *Proc. Int. Conf. Compilers, Architectures, and Synthesis for Embedded Systems (CASES 2003)*, San Jose, CA, USA, Oct. 30–Nov. 1, 2003, pp. 198–209.

[186] M. Ossmann. (2004, Dec.) WEP: Dead again, part 1. Security Focus. [Online]. Available: http://www.securityfocus.com/infocus/1814

[187] I. Papaefstathiou, V. Papaefstathiou, and C. Sotiriou, "Design-space exploration of the most widely used cryptography algorithms," *Microprocessors and Microsystems*, no. 28, pp. 561–571, 2004.

[188] C. Patterson, "High performance DES encryption in Virtex FPGAs using JBits," in *Proc. 8th IEEE Symp. Field-Programmable Custom Computing Machines (FCCM 2000)*, Napa Valley, CA, USA, Apr. 17–19, 2000, pp. 113–121.

[189] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," *Communications of the ACM*, vol. 47, no. 6, pp. 53–57, 2004.

[190] K. E. Persson and D. Manivannan, "Secure connections in Bluetooth scatternets," in *Proc. 36th Hawaii Int. Conf. System Sciences (HICSS-36)*, Hawaii, USA, Jan. 6–9, 2003.

[191] H. Peters, R. Sethuraman, A. Beric, P. Meuwissen, S. Balakrishnan, C. A. A. Pinto, W. Kruijtzer, F. Ernst, G. Alkadi, J. van Meerbergen, and G. de Haan, "Application specific instruction-set processor template for motion estimation in video applications," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 15, no. 4, pp. 508–527, 2005.

[192] N. L. Petroni and W. A. Arbaugh, "The dangers of mitigating security design flaws: A wireless case study," *IEEE Security & Privacy*, vol. 1, no. 1, pp. 28–36, 2003.

[193] T. Pionteck, T. Staake, T. Stiefmeier, L. D. Kabulepa, and M. Glesner, "Design of a reconfigurable AES encryption/decryption engine for mobile terminals," in *Proc. 2004 IEEE Int. Symp. Circuits and Systems (ISCAS 2004)*, vol. 2, Vancouver, Canada, May 23–26, 2004, pp. 545–548.

[194] C. Politis, T. Oda, S. Dixit, A. Schieder, H.-Y. Lach, M. I. Smirnov, S. Uskela, and R. Tafazolli, "Cooperative networks for the future wireless world," *IEEE Communications Magazine*, vol. 42, no. 9, pp. 70–79, 2004.

[195] R. Poor and R. Struik. (2004, Nov.) Security resolutions 802.15.4. IEEE 802.15-04-0540-07. [Online]. Available: ftp://ieee:wireless@ftp.802wirelessworld.com/15/04/15-04-0540-01-004b-security-motions.ppt

[196] B. Potter, "Wireless security's future," *IEEE Security & Privacy*, vol. 1, no. 4, pp. 68–72, 2003.

[197] N. Pramstaller, S. Mangard, S. Dominikus, and J. Wolkerstorfer, "Efficient AES implementations on ASICs and FPGAs," in *Revised Selected and Invited Papers from 4th Conf. on the Advanced Encryption Standard (AES 2004), Bonn, Germany, May 10–12, 2004*, ser. Lecture Notes in Computer Science,

H. Dobbertin, V. Rijmen, and A. Sowa, Eds., vol. 3373. Springer-Verlag, 2005, pp. 98–112.

[198] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in embedded systems: Design challenges," *ACM Trans. Embedded Computing Systems*, vol. 3, no. 3, pp. 461–491, 2004.

[199] S. Ravi, A. Raghunathan, and N. Potlapally, "Securing wireless data: System architecture challenges," in *Proc. ACM Int. Symp. System Synthesis (ISSS'02)*, vol. 1, Kyoto, Japan, Oct. 2–4, 2002, pp. 195–200.

[200] S. Ravi, A. Raghunathan, N. Potlapally, and M. Sankaradass, "System design methodologies for a wireless security processing platform," in *Proc. 39th Design Automation Conf.*, New Orleans, LA, USA, June 10–14, 2002, pp. 777–782.

[201] G. Rehm. (2003) 802.11b homebrew WiFi antenna shootout. [Online]. Available: http://www.turnpoint.net/wireless/has.html

[202] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote authentication dial in user service (RADIUS)," RFC 2865, June 2000.

[203] K. Ritvanen, "Protection of data confidentiality and integrity in radio communications systems," Master's thesis, Helsinki University of Technology, Espoo, Finland, 2004.

[204] K. Ritvanen and K. Nyberg, "Upgrade of Bluetooth encryption and key replay attack," in *Proc. 9th Nordic Workshop on Secure IT Systems (Nordsec2004)*, Espoo, Finland, Nov. 4–5, 2004.

[205] R. Rivest, "The MD5 message-digest algorithm," RFC 1321, Apr. 1992.

[206] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 2, no. 21, pp. 120–126, 1978.

[207] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat, "Design strategies and modified descriptions to optimize cipher FPGA implementations: Fast and compact results for DES and Triple-DES," in *Proc. 13th Int. Workshop Field Programmable Logic and Applications (FPL 2003), Lisbon, Portugal, Sept. 1–3, 2003*, ser. Lecture Notes in Computer Science, P. Y. K. Cheung, G. A. Constantinides, and J. T. de Sousa, Eds., vol. 2778. Springer-Verlag, 2003, pp. 181–193.

[208] ——, "Efficient uses of FPGAs for implementations of DES and its experimental linear cryptanalysis," *IEEE Trans. Computers*, vol. 52, no. 4, pp. 473–482, 2003.

[209] ——, "Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications," in *Proc. IEEE Int. Conf. Information Technology: Coding and Computing (ITCC 2004)*, vol. 2, Las Vegas, NV, USA, Apr. 4–6, 2004, pp. 583–587.

[210] G. P. Saggese, A. Mazzeo, N. Mazzocca, and A. G. M. Strollo, "An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm," in *Proc. 13th Int. Workshop on Field Programmable Logic and Applications (FPL 2003), Lisbon, Portugal, Sept. 1–3, 2003*, ser. Lecture Notes in Computer Science, vol. 2778. Springer-Verlag, 2003, pp. 292–302.

[211] K.-T. Salli, "Security design for a wireless local area network," Master's thesis, Tampere Univ. of Tech., Tampere, Finland, 1998.

[212] K.-T. Salli, T. D. Hämäläinen, J. Knuuttila, and J. Saarinen, "Security design for a wireless local area network," in *Proc. 9th IEEE Int. Conf. Personal, Indoor and Mobile Radio Comm. (PIMRC'98)*, Boston, MA, USA, Sept. 8–11, 1998, pp. 1540–1544.

[213] N. Sastry and D. Wagner, "Security considerations for IEEE 802.15.4 networks," in *Proc. 2004 ACM Workshop on Wireless Security (WiSE'04)*, Philadelphia, PA, USA, Oct. 1, 2004, pp. 32–42.

[214] A. Satoh and S. Morioka, "Hardware-focused performance comparison for the standard block ciphers AES, Camellia, and Triple-DES," in *Proc. 6th Int. Conf. Information Security (ISC 2003), Bristol, UK, Oct. 1–3, 2003*, ser. Lecture Notes in Computer Science, C. Boyd and W. Mao, Eds., vol. 2851. Springer-Verlag, 2003, pp. 252–266.

[215] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact Rijndael hardware architecture with S-box optimization," in *Proc. 7th Int. Conf. Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2001), Gold Coast, Australia, Dec. 9–13, 2001*, ser. Lecture Notes in Computer Science, C. Boyd, Ed., vol. 2248. Springer-Verlag, 2001, pp. 239–254.

[216] T. Schaffer, A. Glaser, and P. D. Franzon, "Chip-package co-implementation of a Triple DES processor," *IEEE Trans. Advanced Packaging*, vol. 27, no. 1, pp. 194–202, 2004.

[217] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C*, 2nd ed. New York, NY, USA: John Wiley & Sons, Inc., 1996.

[218] Y. Shaked and A. Wool, "Cracking the Bluetooth PIN," in *Proc. 3rd Int. Conf. Mobile Systems, Applications, and Services (MobiSys 2005)*, Seattle, WA, USA, June 6–8, 2005, pp. 39–50.

[219] E. Shi and A. Perrig, "Designing secure sensor networks," *IEEE Wireless Communications Magazine*, vol. 11, no. 6, pp. 38–43, 2004.

[220] A. Sikora, "Design challenges for short-range wireless networks," *IEEE Proc. Communications*, vol. 151, no. 5, pp. 473–479, 2004.

[221] D. Singelee and B. Preneel, "Improved pairing protocol for Bluetooth," in *Proc. 5th Int. Conf. on AD-HOC Networks & Wireless (ADHOC-NOW 2006), Ottawa, Canada, Aug. 17–19, 2006*, ser. Lecture Notes in Computer Science, T. Kunz and S. S. Ravi, Eds., vol. 4104. Springer-Verlag, 2006, pp. 252–265.

[222] N. Smyth, M. McLoone, and J. V. McCanny, "Reconfigurable hardware acceleration of WLAN security," in *Proc. IEEE Workshop on Signal Processing Systems (SiPS'02)*, Austin, TX, USA, Oct. 13–15, 2004, pp. 194–199.

[223] W. Stallings, *Network and Internetwork Security: Principles and Practice*. USA: Prentice-Hall, 1995.

[224] F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat, "Efficient implementation of Rijndael encryption in reconfigurable hardware: Improvements and design tradeoffs," in *Proc. 5th Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003), Cologne, Germany, Sept. 8–10, 2003*, ser. Lecture Notes in Computer Science, C. D. Walter, C. K. Koc, and C. Paar, Eds., vol. 2779. Springer-Verlag, 2003, pp. 334–350.

[225] J. A. Stankovic, T. F. Abdelzaher, C. Lu, L. Sha, and J. C. Hou, "Real-time communication and coordination in embedded sensor networks," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1002–1022, 2003.

[226] D. Stanley, J. Walker, and B. Aboba, "Extensible authentication protocol (EAP) method requirements for wireless LANs," RFC 4017, Mar. 2005.

[227] A. Stubblefield, J. Ioannidis, and A. D. Rubin, "A key recovery attack on the 802.11b Wired Equivalent Privacy protocol (WEP)," *ACM Trans. Information and Systems Security*, vol. 7, no. 2, pp. 319–332, 2004.

[228] C.-P. Su, T.-F. Lin, C.-T. Huang, and C.-W. Wu, "A high-throughput low-cost AES processor," *IEEE Comm. Magazine*, vol. 41, no. 12, pp. 86–91, 2003.

[229] Z. Tao, Z. Guo, and R. Yao, "Piconet security in IEEE 802.15.3 WPAN," in *Proc. IEEE Wireless Communications and Networking Conf. (WCNC 2005)*, vol. 4, New Orleans, LA, USA, Mar. 13–17, 2005, pp. 2125–2130.

[230] D. Taylor, T. Wu, and T. Perrin, "Using SRP for TLS authentication," IETF Internet Draft, Oct. 2005, work in progress.

[231] K. Tikkanen, M. Hännikäinen, T. D. Hämäläinen, and J. Saarinen, "Advanced prototype platform for a wireless multimedia local area network," in *Proc. 10th European Signal Processing Conf. (EUSIPCO 2000)*, Tampere, Finland, Sept. 5–8, 2000, pp. 2309–2312.

[232] S. Trimberger, R. Pang, and A. Singh, "A 12 Gbps DES encryptor/decryptor core in an FPGA," in *Proc. 2nd Int. Workshop Cryptographic Hardware and Embedded Systems (CHES 2000), Worcester, MA, USA, Aug. 17–18, 2000*, ser. Lecture Notes in Computer Science, C. K. Koc and C. Paar, Eds., vol. 1965. Springer-Verlag, 2000, pp. 156–163.

[233] K. H. Tsoi, K. H. Lee, and P. H. W. Leong, "A massively parallel RC4 key search engine," in *Proc. 10th IEEE Symp. Field-Programmable Custom Computing Machines (FCCM 2002)*, Napa Valley, CA, USA, Apr. 22–24, 2002, pp. 13–21.

[234] J. T. Vainio. (2000, May) Bluetooth security. [Online]. Available: http://www.niksula.cs.hut.fi/∼jiitv/bluesec.html

[235] S. Vaudenay, "On Bluetooth repairing: Key agreement based on symmetric-key cryptography," in *Proc. 1st SKLOIS Conf. Information Security and Cryptology (CISC 2005), Beijing, China, Dec. 15–17, 2005*, ser. Lecture Notes in Computer Science, D. Feng, D. Lin, and M. Yung, Eds., vol. 3822. Springer-Verlag, 2005, pp. 1–9.

[236] J. R. Walker, "Unsafe at any key size: An analysis of the WEP encapsulation," IEEE, Tech. Rep. 802.11-00/362, Oct. 2000.

[237] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," in *Proc. 25th Annu. Int. Cryptology Conf., Advances in Cryptology (CRYPTO 2005)*, Santa Barbara, CA, USA, Aug. 14–18, 2005.

[238] X. Wang and H. Yu, "How to break MD5 and other hash functions," in *Proc. 24th Int. Conf. Theory and Application of Cryptographic Techniques, Advances in Cryptology EUROCRYPT 2005*, Aarhus, Denmark, May 22-26, 2005, pp. 19–35.

[239] (2005) Wardriving tools, wardriving software, wardriving utilities. [Online]. Available: http://www.wardrive.net/wardriving/tools

[240] O. Whitehouse. (2003, Oct.) War nibbling: Bluetooth insecurity. [Online]. Available: http://www.rootsecure.net/content/downloads/pdf/atstake_war_nibbling.pdf

[241] ——. (2004, Apr.) Bluetooth: Red Fang, Blue Fang. Presentation at CanSecWest/core04. [Online]. Available: http://cansecwest.com/csw04/csw04-Whitehouse.pdf

[242] D. Whiting, R. Housley, and N. Ferguson, "Counter with CBC-MAC (CCM)," RFC 3610, Sept. 2003.

[243] (2005) Wi-Fi Alliance website. [Online]. Available: http://www.wi-fi.org

[244] (2003, Apr.) Wi-Fi protected access: Strong, standard-based, interoperable security for today's Wi-Fi networks. Wi-Fi Alliance. [Online]. Available: http://www.wi-fi.org

[245] M. J. Wiener, "Efficient DES key search," in *Practical Cryptography for Data Internetworks*, W. Stallings, Ed. IEEE Computer Society Press, 1996. [Online]. Available: http://citeseer.ist.psu.edu/wiener93efficient.html

[246] D. C. Wilcox, L. G. Pierson, P. J. Robertson, E. L. Witzke, and K. Gass, "A DES ASIC suitable for network encryption at 10 Gbps and beyond," in *Proc. 1st Int. Workshop Cryptographic Hardware and Embedded Systems (CHES 1999), Worcester, MA, USA, Aug. 12–13, 1999*, ser. Lecture Notes in Computer Science, C. K. Koc and C. Paar, Eds., vol. 1717. Springer-Verlag, 1999, pp. 37–48.

[247] J. Williams, "The IEEE 802.11b security problem, part 1," *IEEE IT Professional*, vol. 3, no. 6, pp. 91–96, 2001.

[248] T. Wollinger, J. Guajardo, and C. Paar, "Security on FPGAs: State-of-the-art implementations and attacks," *ACM Trans. Embedded Computing Systems*, vol. 3, no. 3, pp. 534–574, 2004.

[249] K. Wong, M. Wark, and E. Dawson, "A single-chip FPGA implementation of the data encryption standard (DES) algorithm," in *Proc. Global Telecommunications Conf. (GLOBECOM'98)*, Sydney, Australia, Nov. 8–12, 1998, pp. 827–832.

[250] A. Wool, "Why security standards sometimes fail," *Communications of the ACM*, vol. 45, no. 12, p. 144, 2002.

[251] ——, "A note on the fragility of the 'Michael' message integrity code," *IEEE Trans. Wireless Communications*, vol. 3, no. 5, pp. 1459–1462, 2004.

[252] (2005) WPA Cracker website. [Online]. Available: http://www.tinypeap.com/html/wpa_cracker.html

[253] L. Wu, C. Weaver, and T. Austin, "CryptoManiac: A fast flexible architecture for secure communication," in *Proc. 28th Annu. Int. Symp. Computer Architecture (ISCA 2001)*, Göteborg, Sweden, June 30–July 4, 2001, pp. 110–119.

[254] T. Wu, "Secure remote password protocol," in *Proc. 1998 Internet Society Network and Distributed System Security Symp. (NDSS'98)*, San Diego, CA, USA, Mar. 11–13, 1998, pp. 97–111.

[255] (2005) Xilinx: EasyPath series. Xilinx. [Online]. Available: http://www.xilinx.com/products/silicon_solutions/fpgas/easypath

[256] (2006) Xilinx website. [Online]. Available: http://www.xilinx.com

[257] T. G. Xydis and S. Blake-Wilson, "Security comparison: Bluetooth communications vs. 802.11," White Paper, Bluetooth SIG Security Expert Group, May 2002.

[258] J. Zambreno, D. Nguyen, , and A. Choudhary, "Exploring area/delay tradeoffs in an AES FPGA implementation," in *Proc. 14th Int. Conf. Field Programmable Logic and Applications (FPL 2004), Leuven, Belgium, Aug. 30–Sept. 1, 2004*, ser. Lecture Notes in Computer Science, J. Becker, M. Platzner, and S. Vernalde, Eds., vol. 3203. Springer-Verlag, 2004, pp. 575–585.

[259] X. Zhang and K. K. Parhi, "Implementation approaches for the Advanced Encryption Standard algorithm," *IEEE Circuits and Systems Magazine*, vol. 2, no. 4, pp. 24–46, 2002.

[260] ——, "High-speed VLSI architectures for the AES algorithm," *IEEE Trans. VLSI Systems*, vol. 12, no. 9, pp. 957–967, 2004.

[261] L. Zhao, S. Makineni, and L. Bhuyan, "Anatomy and performance of SSL processing," in *Proc. 2005 IEEE Int. Symp. Performance Analysis of Systems and Software (ISPASS 2005)*, Austin, TX, USA, Mar. 20–22, 2005, pp. 197–206.

[262] "ZigBee specification version 1.0," ZigBee Alliance, Dec. 2004.

[263] (2005) ZigBee Alliance website. [Online]. Available: http://www.zigbee.org

APPENDIX: NOTE ON IWEP

The original security design of TUTWLAN [211, 212] includes a simple algorithm named IWEP, which has been implemented in hardware in [P4]. Despite that IWEP is referred to as a block encryption algorithm, it provides very poor cryptographic protection—if any. The only operations it uses are XOR and byte-wise permutation. IWEP can be expressed as a system of linear equations. When only one of the three variables (plaintext, key, ciphertext) is unknown, the equations can be solved. Hence, IWEP immediately leaks the key when a single pair of plaintext-ciphertext blocks is available to an attacker. This corresponds to the incorrect usage of One Time Pad (OTP) [171] in which the pad is reused. It is also very likely that IWEP can effectively be attacked when only the ciphertext is known.

PUBLICATIONS

PUBLICATION 1

P. Hämäläinen, M. Hännikäinen, M. Niemi, T. D. Hämäläinen, and J. Saarinen “Implementation of Link Security for Wireless Local Area Networks,” in *Proceedings of the 2001 IEEE International Conference on Telecommunications (ICT 2001)*, vol. 1, Bucharest, Romania, June 4–7, 2001, pp. 299–305.

Implementation of Link Security for Wireless Local Area Networks

Panu Hämäläinen[1], Marko Hännikäinen[1], Markku Niemi[2], Timo Hämäläinen[1], and Jukka Saarinen[1]

[1] Tampere University of Technology
Digital and Computer Systems Laboratory
P. O. Box 553, FIN-33101 Tampere, Finland
Tel. +358 3 365 2111, Fax +358 3 365 4575
panu.hamalainen@tut.fi, marko.hannikainen@tut.fi

[2] Nokia Mobile Phones
Sinitaival 5, FIN-33721 Tampere, Finland
Tel. +358 50 511 7341, Fax +358 3 318 3690
markku.t.niemi@nokia.com

ABSTRACT

This paper evaluates the implementation approaches of data transfer security for wireless terminals. Wireless Local Area Networks (WLAN) are examined and security implementations are discussed in the various protocol layers. Consequently, the Medium Access Control (MAC) protocol of the data link layer and configurable hardware implementations are argued to be the most prominent data security solution for wireless terminals. Two strong encryption algorithms, Triple Data Encryption Standard (3DES) and Advanced Encryption Standard (AES), are taken under research. The algorithms are implemented on a Field Programmable Gate Array (FPGA) for integration to a proprietary multimedia WLAN called TUTWLAN. Various implementation approaches are presented, resulting into different trade-offs between size and performance. The results indicate that by accurate design also strong encryption algorithms suitable for e-commerce applications can be efficiently embedded into small wireless terminals.

I. INTRODUCTION

Traditionally computer networks have provided services and solutions for simple data transmission, while video and voice have always had their own channels, e.g. TV broadcasts and telephone networks. However, large-scale convergence of these services with data networks through content digitalization is expected to happen in the near future. Multimedia services, consisting of voice, graphics, video, and data, will be accessible to users through a single network connection.

Along with the multimedia services, another strong trend is the movement towards mobility. This has already been realized for voice in most industrialized countries. Gradually, the developing wireless access networks are also enabling multimedia services for mobile users. The alternatives for personal wireless data services in wide area coverage are the 2nd generation data networks and the emerging 3rd generation services. On the other hand, Wireless Local Area Network (WLAN) technology is a potential alternative for wireless access to local intranets, for example. In addition to person-to-person communications, expected applications for these access networks include entertainment: games, videos, music, and e-commerce in its various forms. For localized services, accounting, stockpiling, education, and healthcare are areas that can easily take advantage of small, embedded, wireless terminals.

As new multimedia features and wireless access make networks more convenient and appealing, they also set new requirements for real-time processing and data security. For example, WLANs are easier to eavesdrop compared to the traditional cable LANs because data are available to anyone within the range of a transmitting device. Therefore, these networks require strong security implementations. On the other hand, there will exist a large variety of terminals targeted to different applications, each terminal containing different processing capabilities. By embedding the security implementations in the lowest protocol layers, i.e. in dedicated hardware, terminals with limited capacity for general processing do not have to compromise the security.

This paper studies the implementation of service security in wireless terminals, particularly the implementation of encryption. A proprietary, multimedia WLAN called TUTWLAN is taken into study. Security implementations purposed for the Medium Access Control (MAC) protocol layer of the TUTWLAN system are evaluated. The paper presents the hardware implementations of Triple Data Encryption Standard (3DES) and Rijndael encryption algorithms. In fall 2000 Rijndael was selected as Advanced Encryption Standard (AES) algorithm [8], but the standardization is still in progress. However, in this paper the Rijndael algorithm is referred to as AES.

The paper is organized as follows. First, motivation for link layer security implementations is given. Then, TUTWLAN and the current demonstrator platform are briefly presented. Next, the 3DES and AES encryption algorithms are explained. 3DES hardware design alternatives are discussed and implementation trade-offs are made. Next, AES implementations are presented and compared to the 3DES implementations. Finally, conclusions are drawn in the last section.

II. MOTIVATION FOR MAC LAYER SECURITY

Data security for an insecure wireless medium can be implemented using a number of approaches. For example, encryption in the current Internet is done mostly using special application layer encryption software, such as Pretty Good Privacy (PGP) with email applications. This is suitable for non-real-time applications and for terminals that have high processing power. However, a WLAN terminal, for example, can be anything between a small stand-alone network camera

with restricted processing capabilities and a large, state-of-the-art workstation. Therefore, in order to maintain enough capacity for user applications and to meet the constantly increasing data rate requirements also with small terminals, it is advantageous to take the implementation of security as low as possible – to the hardware layers. Consequently, compromising the security because of the capacity limitations of the main processor can be avoided as the implementations are carried out using dedicated and optimized hardware. In addition, hardware reconfiguration has to be maintained for enabling easy and low-cost security updating.

Another advantage of embedding the security into the terminal hardware is that encryption is always available and used, without any user interfering or configuration. Moreover, as the encryption implementation is placed low on the protocol stack, more information is protected and the level of privacy thus increased. Instead of only encrypting packet payloads on the application layer, the headers of all the upper protocols, containing for example IP addresses and information of the used applications and accessed Internet servers, are also hidden. This effectively impairs the possibility of malicious traffic analysis.

Another important motivation for the link layer security implementations is the required interoperability. No separate encryption applications or system drivers are required as the security functionality and interoperability are confirmed by layer-to-layer communication standards.

The trend towards lower layer security implementations can be seen in the design of the latest network technologies. For example, the next generation Internet Protocol version 6 (IPv6) [21] provides build-in security features in the network layer, while previously the security has been implemented by applications. In the WLAN field, the IEEE 802.11 WLAN standard [12] and Bluetooth technology [9] place the security to the MAC layer[1]. In addition, authentication and user data security have been taken as a central design requirement for most of the new wireless network technologies being developed. For example, the IEEE 802.16 working group is specifying new wireless access systems in the 5-GHz frequency band [11] and ETSI is preparing the new HIPERLAN/2 standard for broadband WLAN systems [10].

There are also security alternatives on the higher layers in the protocol stack. PGP is an application package providing secured electronic mail transfer, Secure Shell (SSH) can be used for securing remote login, and Transport Layer Security (TLS) [21] provides secured Transmission Control Protocol (TCP) connections. However, it is up to the user whether these high level services are applied. If the application does not use TCP, the availability of TLS is also useless. In addition, currently many network applications do not include any security features. By adding security implementations to the MAC/link layers increases also the security of these legacy applications.

[1] There exist some critique on the security designs of IEEE 802.11 and Bluetooth (http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html, http://www.bell-labs.com/user/markusj/bt.html, 4.4.2001).

A single layer security is efficient only as long as the encryption keys are not available for intruders (supposing that the used algorithm is strong, of course). If an intruder manages to crack one packet of data, access to all traffic in that link is available. A protective solution is to enable dynamic reinforcement of the security by constant exchange of encryption keys. In terminals that enable reconfigurability, the encryption algorithm itself can also be switched according to the application requirements. Another solution is to use additional encryption in some higher layer, which secures at least the user data in case the link encryption is compromised.

III. TUTWLAN DEMONSTRATOR

TUTWLAN is a proprietary, multimedia-capable WLAN implemented at Tampere University of Technology. One of the novelties in TUTWLAN is to tightly integrate the data security into the MAC protocol (TUTMAC) [18] and thus provide a transparently secured transmission channel to upper layer protocols and applications. In addition, TUTWLAN utilizes reconfigurable hardware for the MAC implementation, which enables the implementation of a large range of different protocol features on the same platform. It is possible to change the encryption algorithm, and thereby the level of security independently of the user application. In addition, compared to software, the reconfigurable hardware provides also physical security for encryption keys. As opposed to Application Specific Integrated Circuit (ASIC) designs, reconfigurability enables the switching of algorithms during operation as well as their upgrading and modification in fielded devices [1].

The current TUTWLAN demonstrator platform [16] is depicted in Figure 1. The radio and host interfaces are implemented in a Field Programmable Gate Array (FPGA). The FPGA is a Virtex chip [22]. Virtex is a Xilinx FPGA family that has been designed especially for large and time-critical implementations. It was adopted as a part of the TUTWLAN platform to ensure sufficient area and performance for the interfaces and other functions, including encryption. An Analog Devices ADSP21060L [3] Digital Signal Processor (DSP) is utilized to fulfill the TUTMAC requirements. However, the functions with most demanding real-time requirements, such as synchronization to the medium, are implemented in the FPGA. The radio of the TUTWLAN platform is Intersil PRISM1RC-EVAL, which provides 2 Mbit/s link speed [20]. It is currently being updated to 11 Mbit/s rate.

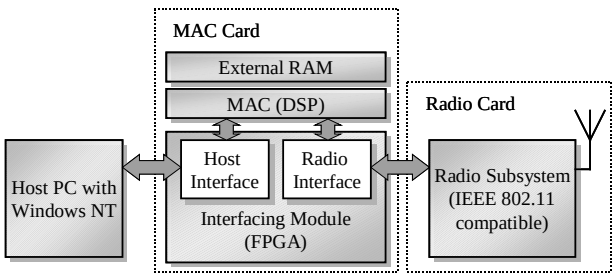


Fig. 1. *TUTWLAN demonstrator platform.*

Originally TUTMAC includes two different encryption algorithms: Improved Wired Equivalent Privacy (IWEP) and International Data Encryption Algorithm (IDEA) [17]. IWEP is the default level of security and it has been implemented in the FPGA. IDEA was included for applications requiring strong data encryption and it has been implemented as a task in the DSP. Because of the further study of encryption algorithms and the ease of reconfigurability, RC4 was also evaluated as a possible algorithm to be integrated into the system [19].

IV. 3DES AND AES ALGORITHMS

Until recently, the most utilized and trusted encryption algorithm has been 3DES. It is based on DES developed in the 1970's. Commonly, 3DES is regarded as a rather weighty cipher, especially for software implementations. 3DES consists of 48 encryption rounds, each of which contains lots of bit level operations. In software even a simple bitwise permutation is relatively tricky and implies several lines of code. On the contrary, in hardware permutations are simple hard-wired connections that are easy to accomplish and do not produce any additional delay. In addition, the algorithm's most essential parts are the substitution boxes (S-box) that can be effectively implemented in hardware. A DES S-box is a non-linear array of 4-bit entries, which are used for replacing parts of the input data.

3DES was designed to encrypt 64-bit blocks of data under the control of three unrelated 56-bit keys. Each key is used as an input to a DES block. The cipher is symmetric. Thus, decryption is accomplished by using the same keys and the same algorithm as in encryption. The only difference is that the key schedule is reversed. Figure 2 presents the high level structure of 3DES. More detailed information can be found from [5].

In 1997 the National Institute of Standards and Technology (NIST) initiated an encryption algorithm development process [8] for replacing over twenty years served DES and eventually 3DES. As a result, NIST received several proposal algorithms and finally, after great amount of debate and analysis, a Dutch algorithm called Rijndael was selected as AES. The design requirements for AES were that it should support key sizes of 128, 192, and 256 bits. In addition, the chosen algorithm was required to be highly resistant against all known cryptographic attacks. Figure 3 depicts the high level structure of AES encryption.

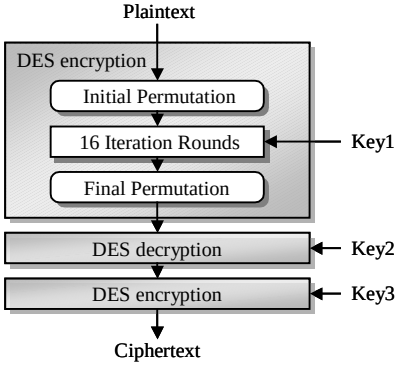


Fig. 2. *High-level structure of 3DES.*

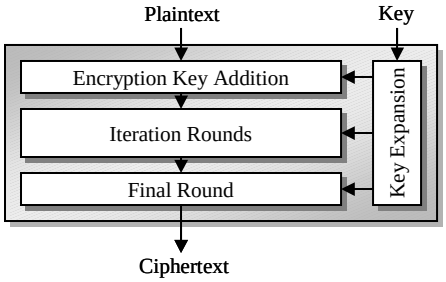


Fig. 3. *High-level structure of AES.*

As most of the block ciphers, AES consists of successive, similar iteration rounds. In addition to the variable-sized key, AES also supports data block sizes of 128, 192, and 256 bits. Depending on the sizes of the key and data, the number of the iteration rounds is 10, 12, or 14. Each round contains a byte substitution, data shifts and mixing, and an encryption key addition. As in DES, the byte substitutions are performed by non-linear S-boxes. Data mixing is done by applying modular multiplication. The encryption key addition is a simple exclusive-or (XOR) operation. More information about the AES development effort, the algorithm's specification, and several analysis reports can be found via [8].

V. ENCRYPTION IMPLEMENTATIONS

This section introduces the realized implementation alternatives, optimizations, and the achieved results of the 3DES and AES algorithms. They only include Electronic Code Book (ECB) mode. ECB is a block encryption mode in which a cipher encrypts plaintext blocks independently of each other. There also exist modes which contain feedback loops from the cipher's output to its input. The advantage of them is that similar plaintext blocks within a message are encrypted into different ciphertext blocks. It is possible to modify the presented implementations quite simply into feedback versions. The implementations were made in Very High-speed integrated circuit Hardware Description Language (VHDL) and the used development software was a PC-version of Xilinx Foundation F2.1i [6]. The designs were targeted to the TUTWLAN board FPGA chip.

A. 3DES Implementations

Since 3DES consists of 48 consecutive similar iterations, it was possible to carry out the basic implementation using only one iteration block, a subkey generator, and a state machine. The iteration block performs enciphering/deciphering, the key generator creates the needed subkeys, and the state machine takes care of feeding the right inputs and subkeys to the iteration block. The same approach was also used in [7] and [14] for DES.

Figure 4 illustrates the high level structure of the basic implementation. The state machine controls the configuration of the multiplexors. The key-generation unit has been excluded from the figure. This implementation supports both encryption and decryption. As stated, the cipher is symmetric. If the decryption mode is chosen, the subkey order is only reversed.

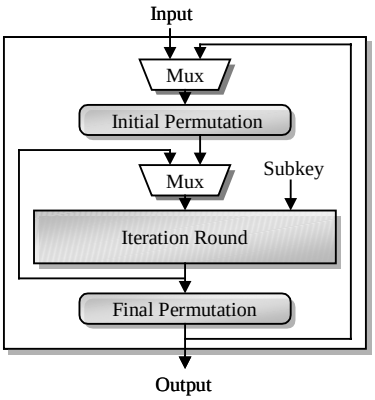


Fig. 4. *Basic 3DES implementation.*

The execution of the entire flow for a 64-bit block of data takes 55 clock cycles. In order to find the best alternative for S-box implementation, two different methods were tested: one that only uses combinatorial logic to perform the substitutions, and another that takes advantage of the FPGA's internal Read Only Memory (ROM). In the combinatorial implementation, the substitutions were made using multiplexors. In the ROM implementation each memory slot contained an output value corresponding to the input. As it was expected, the ROM design proved to be more efficient, and it was used in the other implementations.

Because the initial and final permutations in DES are each other's inverses, performing the final permutation to a message permuted with the initial permutation leads to the original message [4]. Therefore, it is possible to leave out the final permutations of the first and second DES and the initial permutations of the second and third DES. This way four clock cycles per message block are saved (resulting execution time is 51 clock cycles). In addition, excluding the intermediate initial and final permutations makes the outline of the cipher simpler as the data path from final permutation to initial permutation can be removed.

Next it was tested whether the execution time of the cipher could be shortened. The implementation executes two iterations per clock cycle instead of one. As a result, the loop from the iteration round output to its input needs to be executed only 24 times instead of 48. The idea of loop unrolling for DES was originally presented in [2]. However, in [7] it was never taken advantage of.

In order to maximize the throughput, full pipelining was utilized in the last version of the cipher. According to the results of the previous implementations, the most effective way was to execute two iteration rounds per clock cycle. Therefore, the pipelined design consists of 24 consecutive double-iteration rounds with registers in between. Since every iteration round is followed by a register, the output latency for the first block is 24 clock cycles. However, with this design the throughput increases remarkably, because after the first 24 clock cycles a new 64-bit ciphertext block is received on every cycle.

B. AES Implementations

As opposed to 3DES, the AES encryption and decryption are asymmetric. While in 3DES decryption is simply realized by changing the subkey schedule, in AES also the data path has to be changed. This highly increases the required chip size for implementations. However, there exist functional parts that can be shared between encryption and decryption in the design. According to [15], adding decryption increases the amount of needed resources at least by 55%. In order to study this, two version of AES are presented, one including only encryption and the other one with both encryption and decryption. To minimize the area, the combined implementation tries to maximize the resource sharing. This is limited so that the performance (clock cycles per block) is not impaired compared to the plain encryption.

Because of the restricted amount of the user I/O in the TUTWLAN Virtex chip, AES was implemented only as 128-bit-key-and-data versions. However, unlike in [1] and [15], the implementations also include the round key generation unit. In addition, as opposed to [15] the round keys are calculated on the fly in parallel with the encryption core. Therefore, no extra memory for storing pre-calculated round keys or an initialization phase before encryption are needed. The same applies also for the decryption of the combined implementation, providing only that one block is always encrypted before decryption after the key has been changed. Otherwise the first decryption with a new key requires a short setup period. Figure 5 depicts the outline of the implemented AES encryption core.

The high level structures of the both implementations are the same. The core consists of one iteration round block. Similarly to the 3DES basic design, the AES implementations are also iterative. The core logic is controlled by a state machine that sets the iteration round block into the required configuration and feeds it with the right inputs, one from the block input or the iteration round output and the other one from the round key generator. Figure 6 presents the internals of the iteration round block for the encryption-only implementation. The names of the sub-blocks in the figures are taken directly from the Rijndael specification [13].

Most of the AES algorithm's iteration rounds, except the first and the last one, are exactly similar (Figure 3). These two rounds required extra control logic inside the round block in order to utilize resource sharing. This way creating wholly separate components for the first and last round was avoided. That would have been very costly, since, e.g. *ByteSub*, which is needed in the final round, includes 16 substitution boxes, each of which contains 256 eight-bit entries. Similarly to the 3DES designs, the S-boxes were implemented as memory elements resulting in total amount of 4 kilobytes of ROM. Duplicating this would have consumed large amount of additional FPGA area. The rest of the operations inside the round block are quite simple and do not require much resources.

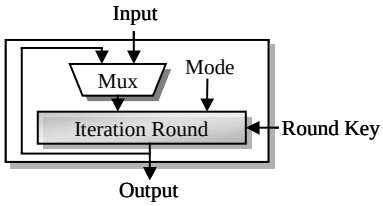


Fig. 5. *AES encryption core.*

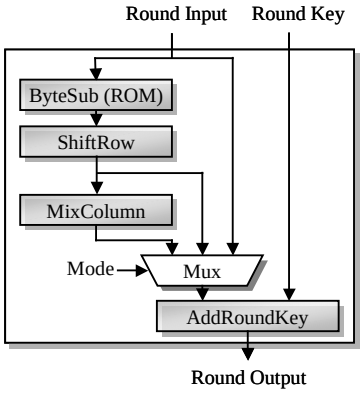


Fig. 6. *AES encryption iteration round.*

Due to the asymmetry and the maximal effort for resource sharing between encryption and decryption, the implementation of the combined iteration round was not as straightforward as the encryption. In decryption every operation performed in encryption needs to be inverted and the round key schedule must be reversed. The only parts that can be shared between encryption and decryption are the byte substitution and the round key addition. However, the *ByteSub* block requires some modifications. Figure 7 depicts the structure of the combined iteration round. As can be seen, in addition to the round number selection, the *mode* signal is now also deciding whether the block is performing encryption or decryption. The reason for placing the inverse of the *MixColumn* after the round key addition is to make the key generation logic simpler (the sub-block could also be placed in parallel with *MixColumn*).

While in the encryption-only version the whole *ByteSub* operation was implemented in ROM, in this case it was divided into three parts. The ROM still contains the same amount of entries, but the entries are different and some extra logic is required around it. As a result, the same ROM can be used in both encryption and decryption. The modified byte substitution logic is presented in Figure 8. Depending on the mode, the multiplexors direct the data through the affine transformation [13] or its inverse.

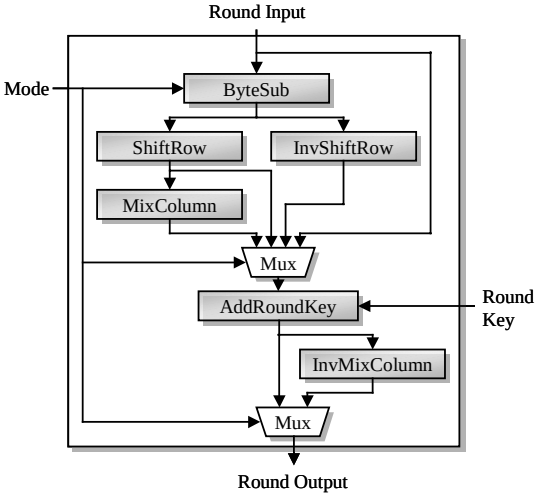


Fig. 7. *Combined AES iteration round.*

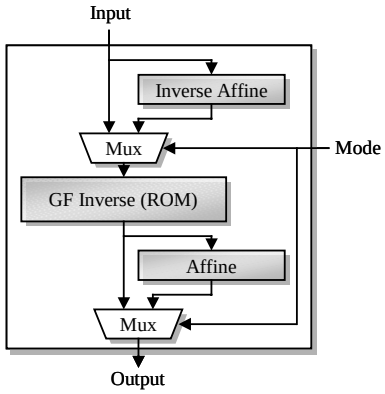


Fig. 8. *Combined ByteSub implementation.*

The key generator, depicted in Figure 9, is quite similar to the encryption core. It is also iterative. For the first time the input is the encryption key and thereafter the output of the generator itself. The *mode* signal is again used for choosing between encryption and decryption (not needed in the encryption-only implementation) and the round number is required for deciding which one of the algorithm's round-dependent constants is used. The block includes four substitution boxes, similar to the boxes in the encryption-only implementation. The last encryption round key is saved for decryption and the decryption round keys are generated from it by inverting the operations of the encryption key generation. This way no extra storage, i.e. RAM, for saving all the round keys was needed. This is profitable, since it is not possible to optimize the RAM size on the FPGA for the round keys using the design tool. Instead, the tool forces to create a too large RAM, which unnecessarily consumes extra resources on the chip. It was estimated that, in this case, adding the logic for inverting the encryption round key generation is more cost-effective than applying the RAM storage.

The encryption of a 128-bit block with the presented designs takes 11 clock cycles. As stated, the same applies for the decryption if at least one encryption with a new key is performed first and the last round key is saved. Otherwise the setup latency for the first decryption is also 11 clock cycles.

The overall performances can still be accelerated by loop unrolling and pipelining. However, according to [15], unrolling two rounds or adding two pipeline stages with decryption included would already exhaust the used FPGA. Therefore, in order to leave enough resources also for the TUTWLAN interfaces, loop unrolling or pipelining were not realized in this study.

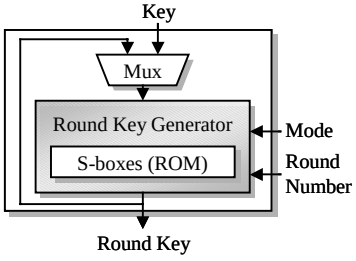


Fig. 9. *AES round key generator.*

VI. IMPLEMENTATION RESULTS

Table 1 shows the detailed implementation results for all the realized 3DES designs on Virtex-V800FG676-6. In the table, the *user I/O* refers to the number of the chip's input/output pins reserved by the designer, a *slice* is a Virtex configurable logic cell, and the *gate count* is the equivalent number of gates needed if the designs were implemented with logical gates only. A slice contains logical gates and Look Up Tables (LUTs), which can be configured to function as ROM. The table also shows the number of clock cycles needed for encrypting a byte of data. The throughputs (megabytes/s) are calculated for the designs' maximum clock frequencies given by the Xilinx Foundation tool.

Table 1. *3DES implementation results.*

	Basic	*Optim*	*2-Rnd*	*Pipeline*
User I/O	298	298	298	298
Slices (% of total)	825 (9%)	740 (8%)	833 (9%)	6,689 (71%)
Gate Count	15,370	13,876	18,930	271,472
Max. Clock (MHz)	40.5	35.8	29.0	45.6
Cycles/Byte	6.88	6.38	3.38	0.125
Throughput (MB/s)	5.90	5.61	8.61	364

The table shows that by removing the redundant permutations from the basic implementation had a significant effect on the area (drop of 10% in used slices). Surprisingly, despite of the achieved smaller area, the maximum clock frequency was decreased. The design tool managed to optimize the basic implementation markedly better than the optimized one (*Optim*).

The two-round implementation (*2-Rnd*) cut the execution time down to 27 clock cycles and proved to be very cost-effective as well. The results show that even though the maximum clock frequency on the chip was somewhat decreased compared to the optimized implementation, the data throughput was still increased when running the both implementations at the maximum clock. It is be possible to execute even more iterations on a single clock cycle. However, since the maximum clock frequency in the two-round design was already decreased, combining more rounds was not expected to increase the overall performance.

As can be seen, pipelining led to quite a large implementation. However, it proved to be very effective when examining the throughput, which is over 360 MB/s. Furthermore, even though the implementation requires 12 kilobytes of ROM, it only consumes 71% of the chip's resources. This shows that Virtex is suitable for very large implementations.

Table 2 presents the implementation results for the AES encryption (*Encryption*) and encryption-decryption (*Combined*) implementations on the same Virtex chip. The increase in area was 44%, which is considerably lower than the estimated 55% [15]. Compared to the 3DES implementations, AES turned out to be quite large and resource demanding. The encryption implementation requires three times and the combined implementation over four times more resources on the chip than the corresponding iterative 3DES implementation (*Optim*). However, even without any loop unrolling or pipelining the AES cipher is able to perform much faster than any of the presented non-pipelined 3DES implementations. It can be estimated that in order to achieve throughput close to the AES implementation, 3DES requires an iterative implementation with as much as 12 pipeline stages, assuming that each stage implements two iteration rounds and a new data block is fed without delays. This kind of pipeline requires a more complicated state machine, which inevitably leads to a decrease in maximum clock frequency and consumes more chip resources. In addition, pipelining can only accelerate the encryption modes without feedback loops.

Table 2. *AES implementation results.*

	Encryption	*Estimate*	*Combined*
User I/O	388	388	388
Slices (% of total)	2,272 (24%)	3,522 (37%)	3,267 (34%)
Gate Count	101,506	157,335	114,414
Max. Clock (MHz)	31.3	-	21.2
Cycles/Byte	0.688	0.688	0.688
Throughput (MB/s)	45.5	-	30.8

A drawback in AES compared to 3DES is that decryption requires an almost separate data path from encryption. This highly increases the needed chip area for AES implementations with decryption. However, the algorithm designers claim that the asymmetry is not a big problem: there exist encryption modes that only require the encryption function to perform also decryption. However, if these modes are not desired or if they cannot be used, decryption also needs to be implemented. For example, one of the most popular encryption modes is Cipher Block Chaining (CBC) which requires also the decryption functionality. Nevertheless, AES offers better security than 3DES and thus it may be worth paying the price of a larger chip.

A. TUTWLAN Implementation Evaluation

Within the presented 3DES implementations, the iterative design with two iterations per clock cycle was considered the best alternative for TUTWLAN. The maximum clock frequency was relatively high, the throughput meets the transmission requirements, and the design could also be fitted on smaller chips. With some re-timing and careful adjustment of the synthesis parameters, it should be possible to increase the maximum clock frequency closer to the target of 40 MHz of the TUTWLAN FPGA.

Like 3DES, AES is considered to be a suitable alternative for TUTWLAN as well. The maximum clock frequency of the design was somewhat lower than the target but it can be increased by adding registers inside the iteration round block and the key generation unit. The addition of one register stage is already likely to meet the timing requirements. Then the throughput stays above the required level as well.

AES gives much freedom to make trade-offs between execution time and area. For example, by reducing the parallelism inside the iteration round blocks it is possible to make the design smaller, but the reduction clearly affects the throughput. One platform dependent modification, which was also tested in the TUTWLAN

chip, is to apply dual-port RAM (DPRAM) instead of ROM. This was assumed to decrease the amount of needed resources for *ByteSub* into half. However, the amount was increased. The reason to this was that despite of the fixed DPRAM control signals, more complicated logic was still generated compared to ROM.

VII. CONCLUSIONS

Service security is an important requirement for enabling new services in wireless networks. The security is implemented by reliable encryption. The experiments with the TUTWLAN platform and the implementations of the 3DES and AES encryption showed that MAC/link layer implemented security can be efficiently realized. Altogether, both of the analyzed algorithms turned out to be large and resource demanding, but the used Virtex chip proved to be well suited for them. In fact, the target frequency was exceeded in 3DES designs and AES can easily be modified to meet the timing. The reached data rates are enough for most real-time applications and currently the strength of 3DES enables also sensitive applications in the field of e-commerce. As 3DES becomes gradually obsolete, it can be easily replaced by AES due to the reconfigurability of the TUTWLAN platform.

REFERENCES

[1] A. Elbirt, W. Yip, B. Chetwynd, C. Paar, "An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists," 3rd AES Conference, April 13-14, 2000, New York, USA, pp. 13-27.

[2] A. G. Broscius, J. M. Smith, "Exploiting Parallelism in Hardware Implementation of the DES," CRYPTO'91, 1991, Santa Barbara, California, USA, pp. 367-376.

[3] "ADSP-2160x SHARC DSP Microcomputer Family," Data Sheet, Analog Devices, Inc., 1999, USA.

[4] D. C. Feldmeier, P. R. Karn, "UNIX Password Security – Ten Years Later," CRYPTO'89, 1989, Santa Barbara, California, USA, pp. 44-63.

[5] "Data Encryption Standard," Federal Information Processing Standards (FIPS) Publication 46-3, National institute of Standards and Technology (NIST), 1999, USA.

[6] "Foundation Series 2.1i Install and Release Document," Xilinx, Inc., 1999, USA.

[7] H. Eberle, "A High-speed DES Implementation for Network Applications," CRYPTO'92, 1992, Santa Barbara, California, USA, pp. 521-539.

[8] Homepage of Advanced Encryption Standard (AES) Development Effort, http://csrc.nist.gov/encryption/aes, 14.2.2001.

[9] Homepage of Bluetooth Special Interest Group (SIG), http://www.bluetooth.com, 15.2.2001.

[10] Homepage of ETSI BRAN Project, http://www.etsi.org/bran, 14.2.2001.

[11] Homepage of IEEE 802.16 Working Group, http://grouper.ieee.org/groups/802/16, 14.2.2001.

[12] IEEE Std 802.11, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," LAN MAN Standards Committee of the IEEE Computer Society, 1999, USA.

[13] J. Daemen, V. Rijmen, "AES Proposal: Rijndael," http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf, 6.4.2001.

[14] J-P. Kaps, C. Paar, "Fast DES Implementations for FPGAs and its Application to a Universal Key-Search Machine," 5th Annual Workshop on Selected Areas in Cryptography (SAC '98), August 17-18, 1998, Kingston, Ontario, Canada, pp. 234-247.

[15] K. Gaj, P. Chodowiec, "Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware," 3rd AES Conference, April 13-14, 2000, New York, USA, pp. 40-54.

[16] K. Tikkanen, M. Hännikäinen, T. Hämäläinen, J. Saarinen, "Advanced Prototype Platform for a Wireless Multimedia Local Area Network," X European Signal Processing Conference (EUSIPCO 2000), September 5-8, 2000, Tampere, Finland, pp. 2309-2312.

[17] K.-T. Salli, T. Hämäläinen, J. Knuutila, J. Saarinen, "Security Design for a Wireless Local Area Network," The 9th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'98), September 8-11, 1998, Boston, USA, pp. 1540-1544.

[18] M. Hännikäinen, J. Knuutila, A. Letonsaari, T. Hämäläinen, J. Jokela, J. Ala-Laurila, J. Saarinen, "TUTMAC: a Medium Access Control Protocol for a New Multimedia Wireless Local Area Network," The 9th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'98), September 8-11, 1998, Boston, USA, pp. 592-596.

[19] P. Hämäläinen, M. Hännikäinen, T. Hämäläinen, J. Saarinen, "Hardware Implementation of the Improved WEP and RC4 Encryption Algorithms for Wireless Terminals," X European Signal Processing Conference (EUSIPCO 2000), September 5-8, 2000, Tampere, Finland, pp. 2289-2292.

[20] "PRISM1RC-EVAL – 2.4 GHz MACless DSSS Radio User's Guide," Intersil Corporation, 1999, USA.

[21] R. Molva, "Internet Security Architecture," Computer Networks 31, Elsevier Science B. V., 1999, pp. 787-804.

[22] "The Programmable Logic Data Book," Xilinx, Inc., 1999, USA.

PUBLICATION 2

P. Hämäläinen, M. Hännikäinen, and T. D. Hämäläinen, “Efficient Hardware Implementation of Security Processing for IEEE 802.15.4 Wireless Networks,” in *Proceedings of the 2005 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS 2005)*, Cincinnati, Ohio, USA, Aug. 7–10, 2005, pp. 484–487.

Efficient Hardware Implementation of Security Processing for IEEE 802.15.4 Wireless Networks

Panu Hämäläinen, Marko Hännikäinen, and Timo D. Hämäläinen
Tampere University of Technology
Institute of Digital and Computer Systems
P. O. Box 553, FI-33101 Tampere, Finland
Tel: +358 3 3115 2111, Fax: +358 3 3115 4561
Email: panu.hamalainen@tut.fi, marko.hannikainen@tut.fi, timo.d.hamalainen@tut.fi

Abstract— **The IEEE 802.15.4 standard defines the medium access control and physical layer for low-rate, low-power Wireless Personal Area Networks (WPAN). As a number of WPAN applications require protected communications, the standard defines security procedures. Since the procedures typically consume most processing capacity in the limited 802.15.4 devices, efficient implementations are needed. As a solution, this paper presents a compact and energy-efficient hardware design, supporting all the security suites of the standard. Compared to typical WPAN processors, the presented FPGA prototype and the estimated ASIC implementation offer significantly higher performance and lower energy consumption. The FPGA throughput at the highest security level is 90 Mb/s and the energy consumption is 1/190 of an 8-bit microcontroller and 1/5 of an ARM9. The estimated energy consumption for the equivalent ASIC implementation is 1/10 of the FPGA prototype. In addition to 802.15.4, the hardware design supports all wireless technologies derived from the IEEE 802.11i security specification.**

I. Introduction

Communications are steadily shifting from wired media to wireless networks. Along with high-speed technologies, a need for low-rate and low-power wireless networks for short-range monitoring and control has emerged. Automation, safety, healthcare, and entertainment at homes, public buildings, and industry are seen as the major application fields. Example applications include heating and alarms systems, remote controllers, and data transfers between personal portable devices.

IEEE 802.15.4 [1], ratified in the end of the year 2003, is one of the first standards defining the radio and the medium access control layer for a low-rate, low-power Wireless Personal Area Network (WPAN). The technology is also referred to as ZigBee, which is an industry alliance working on the 802.15.4 and upper protocol layers. The 802.15.4 standard supports three frequency bands (868 MHz, 915 MHz, and 2.45 GHz) and data rates up to 250 kb/s. The 802.15.4 devices either form a star topology network or communicate through peer-to-peer connections.

It is clear that, e.g., alarm systems must be protected from unauthorized exploits. Hence, the 802.15.4 standard specifies cryptographic procedures for protecting communications at the medium access control layer. Security procedures are generally among the tasks requiring most processing capacity in network devices. Thus, the overall processing limitations as well as energy consumption can be significantly alleviated with efficient security processing implementations, which is especially desirable in battery-powered devices. This paper presents a compact and energy-efficient hardware design for the 802.15.4 security processing. Since the standard utilizes the work of the IEEE 802.11i security group [2], the design supports other new wireless IEEE technologies as well.

The paper is organized as follows. Section II presents an overview of the 802.15.4 security processing. Section III discusses the different alternatives for the hardware implementation of the processing. In Section IV the implementation of the alternative argued to be best suited for 802.15.4 is presented. The implementation results are reported in Section V. The section also compares the results to processors of the same application domain. Section VI concludes the paper.

II. Overview of 802.15.4 Security Processing

The 802.15.4 standard [1] defines optional cryptographic security suites for providing either confidentiality, integrity, or both. Confidentiality is achieved through encryption using Advanced Encryption Algorithm (AES) [3] in Counter mode (CTR) and integrity through Message Integrity Codes (MIC) generated with AES in Cipher Block Chaining Message Authentication Code (CBC-MAC) mode. Hence, the integrity also includes authentication of origin. The combination is offered with AES in the CTR with CBC-MAC mode (CCM).

The 802.15.4 security suites and their services are summarized in Table I. The length of the 128-bit MIC can be truncated to 32 or 64 bits by choosing corresponding AES-MIC or AES-CCM suites. The AES-CTR and AES-CCM suites can optionally be configured to utilize freshness protection against replay attacks. Authentication and key exchange are not defined in the standard. They must be implemented on the higher protocol layers.

In the AES-CTR and AES-CCM suites the 802.15.4 payload data is en/decrypted by XORing a key stream produced from a secret key and a counter with the data. The counter is incremented for each data block. It consists of a nonce and a 16-bit block counter (in the standard there are also other fields but in this work they are all regarded as a part of the nonce). For integrity the AES-MIC and AES-CCM suites process each data block with AES after XORing the plaintext block with the previous ciphertext block. The last ciphertext

0-7803-9197-7/05/$20.00 © 2005 IEEE.

TABLE I
802.15.4 SECURITY SUITES AND THEIR SERVICES

Suite	Confidentiality	Integrity	Freshness
AES-CTR	√	-	√
AES-MIC-128	-	√	-
AES-MIC-64	-	√	-
AES-MIC-32	-	√	-
AES-CCM-128	√	√	√
AES-CCM-64	√	√	√
AES-CCM-32	√	√	√

block is truncated to the length defined by the selected suite and output as the MIC. The MIC protects selected portions of the 802.15.4 header and the payload. In the AES-CCM suites also the MIC is encrypted. The security suites only require the 128-bit-key forward functionality of AES itself.

III. HARDWARE DESIGN ALTERNATIVES

There are a number of alternatives for the hardware implementation of the 802.15.4 security processing. Since AES is the computational core, its design has a significant effect on the energy consumption and performance of the whole design.

Similarly to other block ciphers with rounds, the basic approaches for the AES implementation are the iterated, pipelined, and loop-unrolled architectures [4]. In the iterated architecture the processed block is circulated through a single-cycle round logic. The pipelined architecture consists of the AES rounds with registers in between. A loop-unrolled architecture performs two or more rounds at a clock cycle and the execution is either iterated or pipelined. More advanced methods include sub-pipelining and different combinations of the three basic approaches [4]. As the iterated architecture is the smallest, it is considered the most cost-effective choice for 802.15.4 in this work. Due to the low data rates of 802.15.4, high enough throughput is also achieved.

Each 32-bit piece of a data block can be processed independently during an AES round. Hence, the area of a round can be decreased with a 32-bit folded round [5], reducing the width of the data path by the factor of four. Even though the cycle count is increased with the same factor, in this work the area savings are regarded more significant and folding is utilized.

The AES substitution boxes (S-box) [3] are often implemented as memory-based look-up-tables (LUT). The LUT size can be decreased at the cost of latency when the S-box representation is transformed into another arithmetic domain [6]. On the other hand, the latency can be shortened by combining more operations into large LUTs, which require fewer accesses. Whereas memory-based S-boxes are the best choices for compact Field Programmable Gate Array (FPGA) designs, combinatorial logic is well-suited for Application Specific Integrated Circuits (ASIC). Also, in FPGAs combinatorial logic can be used for fine-grained pipelining in high-speed designs at the cost of resources [7]. In this paper the prototype implementation of the 802.15.4 security processing is targeted to a FPGA, and thus, memory-based S-boxes are utilized for a compact design. However, they can be easily substituted with another technique due to the modularity of the design.

The AES roundkeys can be precomputed or generated on-the-fly [4]. Precomputing requires setup time and memory but it also implies power savings as the key logic has to operate only once per AES key [5]. Opposed to on-the-fly computation, precomputation can share resources with the AES data path, supporting compact designs. Thus, precomputation is chosen for the 802.15.4 implementation.

In addition to the speed, power, and area requirements, the utilized encryption mode defines the reasonable architectural choices for the AES core. The throughputs of the CTR mode and the CTR encryption in the CCM mode can be unlimitedly increased by utilizing pipelining and parallel cores. However, due to the feedback loop of the MIC mode and the CCM MIC generation, already a single, iterated core achieves the maximum throughput for the MIC computation. Therefore, in 802.15.4 the most cost-efficient choice is to utilize only one, iterated AES core. In order to reduce buffering, in this paper the CCM mode interleaves the MIC and CTR processing.

IV. 802.15.4 SECURITY PROCESSING DESIGN

The prototype hardware of the 802.15.4 security processing is implemented in register transfer level VHDL and synthesized to an Altera Cyclone FPGA. The design consists of a compact AES core with pre-computed key schedule and control logic for supporting all the 802.15.4 security suites.

A. Implementation Platform

The FPGA utilized in this work is Altera Cyclone EP1C4F324C6 [8]. It is comprised of a two-dimensional logic array, consisting of Logic Array Blocks (LAB), each containing 10 Logic Elements (LE) and interconnects. A LE constitutes of a 4-input LUT and a register. The FPGA contains also embedded M4K memory blocks, which can be used for implementing a variety of memory functions, including shift registers, FIFO buffers, and single- and dual-port RAMs and ROMs. An M4K can store up to 4,608 bits. EP1C4F324C6 contains 4,000 LEs and 17 M4K blocks. The design tools utilized in this work are ModelSim SE PLUS 5.8d 2004.06, Precision RTL Synthesis 2003b.41, and Quartus II v4.1.

B. AES Design

The design of the AES core is based on the folded round design of [5]. In this work the decryption functionality is removed and the intermediate results (*state*) are stored in four 8-byte dual-port RAMs (DPRAM) as the M4Ks do not support shift registers with suitable tap distances. The data path architecture is presented in Fig. 1. In the following, the names SubBytes, ShiftRows, MixColumns, Rcon, and *state* are consistent with the AES specification [3].

In the figure all the connections are 32 bits wide. The clocked entities are marked with '>'. The core consists of the encryption and roundkey generation sharing the *data in* port and the 32-bit SubBytes implemented as two 256-byte

dual-port ROMs. The *load key* signal controls whether the roundkey generation or encryption is performed. The roundkeys are stored in a 32-bit RAM with 44 memory slots. Along with storing the intermediate results, the *state* entity contains addressing logic for performing the ShiftRows operation.

The implementation results of the AES core are presented in Table II. For comparison, the table also shows the results for the most common AES implementation method with an iterated, single-round, block-wide (128-bit) architecture and the on-the-fly roundkey computation in the same FPGA. The value *memory bits* reports the number of used bits in M4Ks. The cycle count and the throughput of the folded core do not include the roundkey generation latency.

The powers were estimated using combined ModelSim and Quartus power simulation at 50 MHz. The measurements were performed using a static encryption key and random input blocks. It was also found out that if the key is switched for every block, the power of the folded design slightly decreases since the *state* entity is used less frequently. The standby power is fixed and only depends on the chosen FPGA. The power-effectiveness of the designs can be evaluated by comparing the sum of the other three power values. The dynamic internal power of the folded design is only 34%, LE count 37%, and the memory bit count 28% of the block-wide design. Despite the higher frequency, the throughput is lower because of the increased cycle count.

TABLE II
COMPARISON OF AES IMPLEMENTATIONS IN EP1C4F324C6

Measure	Block-wide	Folded
Data path width, bits	128	32
Logic elements	1,368	512
Memory bits (M4Ks)	20,408 (10)	5,760 (7)
Max. clock, MHz	110	116
Key setup, cycles	0	46
Cycles per block	10	55
Throughput @max. clock, Mb/s	1,408	270
Total internal power @50 MHz, mW	138.34	86.40
- Standby power, mW	60.00	60.00
- Logic element power, mW	56.63	11.44
- M4K power, mW	7.64	7.62
- Clock tree power, mW	14.07	7.34

C. 802.15.4 Security Processing

The data path of the 802.15.4 security processing design is shown in Fig. 2. Input data encodings and MIC value comparisons and truncations are performed outside. The interface and internal signals are 32 bits wide, except the control signals, *pad length*, *nonce*, and the 16-bit block counter *ctr*.

Key and data blocks are fed through the shared *data in* port. The *load key* signal setups the module for reading new key and nonce values. The key is forwarded four bytes at a time through the multiplexers (Mux) 1, 3, and 5 to the AES core and the roundkeys are generated. It is also possible to update only the nonce and maintain the already computed roundkeys. The *mode* signal defines the CTR, MIC, or CCM mode and sets the module to encrypt or decrypt (affects only the CCM mode). The *pad length* input is used for informing the number of padding bytes in the final input block. This is required in the CCM decryption for ignoring (masking) the last bytes of the decrypted block before the MIC computation phase.

For each 802.15.4 mode the input data blocks are stored in the 128-bit data register. Whereas in the MIC and CTR modes the blocks could be processed without storing, in the CCM mode they are needed twice. The 128-bit chain register maintains the state of the MIC processing. In the MIC computations the AES core is fed with the data XORed with the chain register. The AES result is stored as a new value in the chain register and output after the last block. In the CTR computations the AES input consists of the concatenated *nonce* and *ctr*. The data blocks are XORed with the AES result and output. For the XORing the AES result is conveyed through the *mask* entity and Mux 2 to the XOR gate.

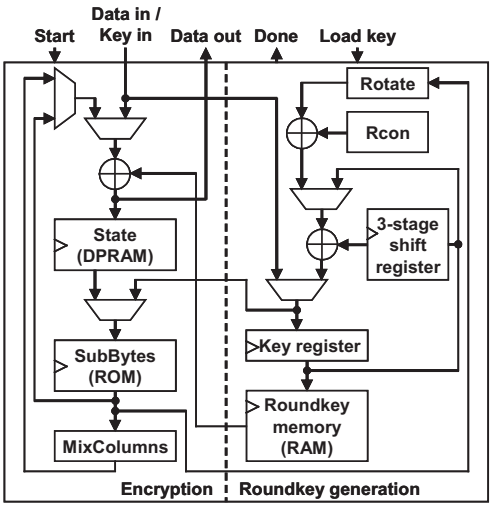


Fig. 1. Data path of the folded AES core.

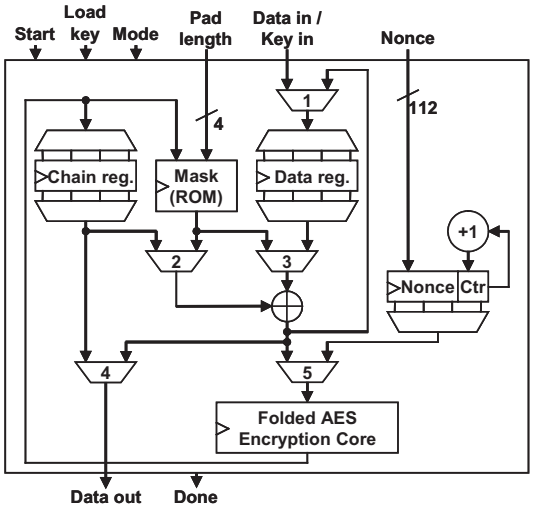


Fig. 2. Data path of the 802.15.4 security processing hardware.

TABLE III

SECURITY PROCESSING IMPLEMENTATION IN EP1C4F324C6

Measure	Value
Data path width, bits	32
Logic elements	1,434
Memory bits (M4Ks)	7,808 (11)
Max. clock, MHz	78
Internal power @50 MHz, mW	98.92
- Standby power, mW	60.00
- LE power, mW	14.91
- M4K power, mW	7.16
- Clock tree power, mW	16.85

TABLE IV

PERFORMANCE OF SECURITY PROCESSING IMPLEMENTATION

Measure	CTR	MIC	CCM
Key setup, cycles	48	48	48
Cycles per block	57	57	112
Throughput @78 MHz, Mb/s	176	176	90

In the CCM mode the processing starts with MIC computation for the 802.15.4 header and then constantly alternates between the CTR and MIC configurations for the payload. The order of CTR and MIC processing must be switched in decryption. As an opposite of the CTR mode, in CCM *ctr* is set to the value 1 for the first data block. When the MIC is computed or input, it is en/decrypted by setting *ctr* to zero [1]. In the decryption the decrypted MIC is output first, followed by the computed MIC.

V. RESULTS

The results for the 802.15.4 design in EP1C4F324C6 are presented in Table III. In addition to the AES core, memory bits are used by the masking logic (16×128 bits). The folded core consumes 36% of the reserved LEs and 74% of the reserved memory bits. The presented power figures are for the CCM mode with a static key. It was measured that the powers of the other modes are also very close to CCM. The performances for the 802.15.4 processing modes are presented in Table IV.

Table V compares the security processing design with implementations in typical processors of the 802.15.4 application domain. The throughputs (Tp) are for the CCM mode at the reported clock frequency. DS80C323 [9] is an 8-bit, 8051-compatible microcontroller and ARM966E-S [10] is a 32-bit processor aimed at embedded devices. The throughput for DS80C323 is derived from [11] and for ARM9 from [12]. CCM requires two AES passes for a data block. The processor powers do not include memories.

The table also presents the power and the throughput for an ASIC estimate of the 802.15.4 security processing design. Ref. [13] reports a 0.18 μm CMOS AES design comparable to the block-wide core of this paper. Equal throughputs are achieved with both the designs at the same clock frequency. The power of the 802.15.4 ASIC estimate is computed by assuming that the power differences in the ASIC technology are consistent with the dynamic power (standby power excluded) differences between the FPGA designs of this paper.

TABLE V

COMPARISON OF 802.15.4 SECURITY PROCESSING IMPLEMENTATIONS

Platform	Clock (MHz)	Power (mW)	Tp (Mb/s)
FPGA	50	99	57
ASIC (estimate)	125	28	140
DS80C323 [9] [11]	18	30	0.09
ARM966E-S [10] [12]	200	140	16

Significant energy savings are achieved with the 802.15.4 design compared to the processor implementations. With precomputed roundkeys the energy consumption of the CCM mode in the FPGA is 0.22 μJ per data block, which is $1/190$ of DS80C323 and $1/5$ of ARM9. The energy consumption of the ASIC estimate is $1/10$ of the FPGA.

VI. CONCLUSION

This paper presented a compact and energy-efficient hardware design for the 802.15.4 security processing. The design was argued to offer the best architectural area/power/performance ratios for the low-power wireless devices. Compared to typical WPAN processors, the implemented FPGA prototype and the estimated ASIC implementation provide significantly lower energy consumption and higher performance. The same design can also be utilized in WLAN terminals employing the IEEE 802.11i standard.

REFERENCES

[1] *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPAN)*, IEEE Std. 802.15.4, 2003.

[2] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Medium Acccess Control (MAC) Security Enhancements*, IEEE Std. 802.11i, 2004.

[3] *Advanced Encryption Standard (AES)*, FIPS Std. 197, 2001.

[4] X. Zhang and K. K. Parhi, "Implementation approaches for the Advanced Encryption Standard algorithm," *IEEE Circuits and Systems Magazine*, vol. 2, no. 4, pp. 24–46, 2002.

[5] P. Chodowiec and K. Gaj, "Very compact FPGA implementation of the AES algorithm," in *Proc. 5th Int. Workshop Cryptographic Hardware and Embedded Systems (CHES 2003), Cologne, Germany, Sept. 8–10, 2003*, ser. Lecture Notes in Computer Science, C. D. Walter, C. K. Koc, and C. Paar, Eds., vol. 2779. Springer-Verlag, 2003, pp. 319–333.

[6] C.-P. Su, T.-F. Lin, C.-T. Huang, and C.-W. Wu, "A high-throughput low-cost AES processor," *IEEE Communications Magazine*, vol. 41, no. 12, pp. 86–91, 2003.

[7] X. Zhang and K. K. Parhi, "High-speed VLSI architectures for the AES algorithm," *IEEE Trans. VLSI Systems*, vol. 12, no. 9, pp. 957–967, 2004.

[8] (2005) Altera website. [Online]. Available: http://www.altera.com

[9] *DS80C320/DS80C323 High-Speed/Low-Power Microcontrollers*, Maxim/Dallas Semiconductors, 2004.

[10] (2005) ARM website. [Online]. Available: http://www.arm.com

[11] J. Daemen and V. Rijmen, "AES proposal: Rijndael," Mar. 1999. [Online]. Available: http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf

[12] P. Hämäläinen, J. Heikkinen, M. Hännikäinen, and T. D. Hämäläinen, "Design of transport triggered architecture processors for wireless encryption," in *Proc. 8th Euromicro Conference on Digital System Design (DSD 2005)*, Porto, Portugal, Aug. 30–Sept. 3, 2005, (to appear).

[13] I. Verbauwhede, P. Schaumont, and H. Kuo, "Design and performance testing of a 2.29-GB/s Rijndael processor," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 3, pp. 569–572, 2003.

PUBLICATION 3

T. Järvinen, P. Salmela, P. Hämäläinen, and J. Takala, "Efficient Byte Permutation Realizations for Compact AES Implementations," in *Proceedings of the 13th European Signal Processing Conference (EUSIPCO 2005)*, Antalya, Turkey, Sept. 4–8, 2005, 4 pages.

EFFICIENT BYTE PERMUTATION REALIZATIONS FOR COMPACT AES IMPLEMENTATIONS

Tuomas Järvinen, Perttu Salmela, Panu Hämäläinen, and Jarmo Takala

Tampere University of Technology, P.O.Box 553, FIN-33101 Tampere, Finland
Email: forename.surname@tut.fi

ABSTRACT

Advanced Encryption Standard (AES) algorithm incorporates a byte permutation operation which reorders the bytes within a 128-bit data block. This permutation can be described by reading the input data bytes into a 4×4 matrix called state in column wise and shifting the rows by one, two, or three bytes to the left. In decryption, the shifting is reversed, i.e., the rows are shifted to the right. While such shifting operations are straightforward if the computation is done with 128-bit data blocks at a time, they become more complex in area-efficient folded implementations where smaller than 128-bit data blocks are used. In such cases, a storage of data is required, either in the form of registers or memories. In this paper, efficient realizations of the byte permutations in AES algorithm, where the size of simultaneously computed data can be 1, 2, 4, or 8 bytes, are presented. All the realizations use the minimum number of storage elements implying area-efficiency.

1. INTRODUCTION

In various embedded applications, there is a need for a simple low-power implementation of AES algorithm [1] to guarantee a secure data transmission. However, current AES implementations are targeted mainly to high-speed data thus they exploit the parallelism in the algorithm for increasing the throughput. On the other hand, low-rate networks exist, e.g., for short range monitoring and control in automation, safety, healthcare, and entertainment at homes, public buildings and industry. For these applications, IEEE 802.15.4 standard [2] was ratified in the end of 2003 defining the radio and medium access control layer of low-rate, low-power wireless networks. The core of 802.15.4 security is the AES algorithm, thus its implementation has a great impact on area- and power-efficiency of the overall security system.

The current AES implementations can be divided into iterated, pipelined, and loop unrolled structures [3]. In iterated structures, computation of a 128-bit data block requires multiple iterations. In pipelined implementations, additional register stages are placed in the data path for enhancing the throughput. In the loop unrolled implementations, two or more rounds are computed at a clock cycle.

An efficient method for decreasing area and power is obtained with the iterated and folded structures where computation is done with the data blocks of size less than 128-bits. An example of such structure is proposed in [4] where 128-bit data block is computed iteratively in 32-bit blocks. Similar folded structure has been proposed in [5, 6]. The key advantage in folded structures is that the area can be decreased in proportion of the folding factor. However, the byte permutations become more complex since they span over the whole 128-bit block and either registers or memories are needed for delaying certain data bytes. Reducing the complexity of byte permutation realizations is the principal problem considered in this paper.

The AES byte permutations are illustrated in Fig. 1(a,b) with a 128-bit data organized into a 4×4 matrix called state. Each entry in the matrix represents a byte index within a 128-bit data block. These indices are permuted by shifting the row i cyclically to left/right by i, $i = \{0,1,2,3\}$, thus these byte permutations are often called as ShiftRows operations. The permuted indices are $(0,5,10,15,1,6,11,12,2,7,8,13,3,4,9,14)$ for left shift and $(0,7,10,13,1,4,11,14,2,5,8,15,3,6,9,12)$ for right shift permutations.

The principal block diagram of the folded AES structure is depicted in Fig. 2. It consists of ShiftRows, SubBytes, MixColumns, and AddRoundKey operations. Various area-efficient implementations of SubBytes operation have been proposed, e.g., in [6, 7, 8], for MixColumns operation in [4, 6], and for AddRoundKey operation in folded AES implementation in [4]. None of the existing papers have considered efficient ShiftRows implementations for folded AES structures.

In current folded AES structures, the byte permutations are implemented with registers or FPGA-specific dual-port memories. For example, in [4], an FPGA implementation of a folded AES implementation is proposed where the byte permutations are suggested to be performed with separate dual-port memories for read and write operations. At the end of each round, the memories are swapped. This method uses twice the amount of memory that is actually needed. The authors also proposed an alternative realization, which uses shift registers. Also in [5, 6] registers were used for byte permutations. However, none of the previous register-based permutation unit resulted in the minimum number of registers.

In this paper, efficient realizations for the byte permutations in AES algorithm of any cipher key length are proposed. These realizations can be divided into register- and memory-based structures. The advantage of the given structures is that only the minimum amount of storage is needed implying area efficiency. The proposed structures are well suited for folded AES implementations where the computation is done with Q bytes at a time, $Q = \{1,2,4,8\}$. In addition, some application-specific processors may benefit from the proposed structures in AES implementations if they are included as a custom unit. This way some performance gain is obtained since no permutations need to be done with processor's own resources, which can be time time-consuming since the permutations are done with bytes while typical processors operate with 16- or 32-bit words.

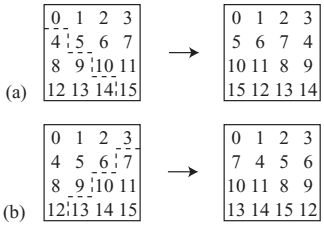


Figure 1: Byte permutations (ShiftRows operation) in AES algorithm: (a) shift rows left, (b) shift rows right.

2. MEMORY-BASED BYTE PERMUTATION UNITS

The principal idea in memory-based byte permutation unit (BPU) is to exploit an in-place storage method such that no conflicts are allowed and only the minimum amount of memory is used. In this way, there is no need to use separate write and read memories but a dual-port memory is needed due to simultaneous read and write operations. In general, the row address of a certain data byte is likely to be different in successive rounds but, however, the row address sequences are cyclic, i.e., they repeat after certain number of rounds.

Next, the address sequences are defined. The module addresses *ma* are constant determined by the following equation where Q is the number of memories and h is the index for 16 data bytes

$$ma(h) = h \mod Q. \quad (1)$$

In this case, there is no need for the multiplexing of data bytes between the memory modules and computation units thus the input and output ports of the memories can be straight connected to other computation units.

The row address sequences are slightly different for each parallel memory configuration. Let us assume that the number of memory modules Q is 1, 2, or 4. In this case, the row address *ra* of module *ma* for the left shift operation is given as

$$ra(j,k) = j+4j(k+1)+(4/Q)(k+1)ma \mod 16/Q \quad (2)$$

where j is the access index, $j = (0,1,\ldots,16/Q-1)$, and k is the round, $k = (0,1,2,\ldots)$.

For the right shift operation, the row address for module *ma* is given as

$$ra(j,k) = j-4j(k+1)-(4/Q)(k+1)ma \mod 16/Q. \quad (3)$$

Implementation of the row address generator is very simple: as the access index j and round index k need to be provided anyway, no separate counters are needed. It is also worth noting that the number of memory modules Q is constant in actual realization, thus no division need to be implemented. In addition, since all the computations are modulo $16/Q$, the word width for the generated row address is $\log_2(16/Q)$.

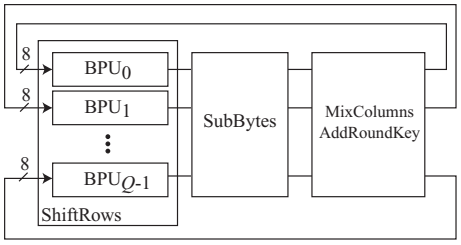


Figure 2: Principal block diagram of folded AES structure. BPU: byte permutation unit. Q: number of bytes processed at a time. $Q = \{1,2,4,8\}$.

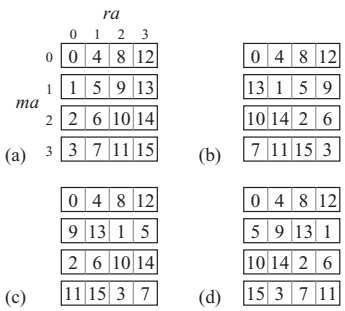


Figure 3: Evolution of memory contents: (a) initial state, (b) after 1st round ($k = 0$), (c) after 2nd round ($k = 1$), and (d) after 3rd round ($k = 2$).

Next, the operation of memory-based BPUs is illustrated with an example where the number of parallel memories is four, which is targeted to AES structures where four bytes are computed at a time, as in [4, 5]. First, the 128-bit data block is initialized into four dual-port memories as depicted in Fig. 3(a). As shown, the data bytes are written into consecutive memory locations. Consider the left shift operation where the first four bytes to be processed are $0,5,10,15$. These bytes are read out, processed, and the results with the new indices $0,1,2,3$ are stored into the same locations. The row addresses for that are obtained from (2). After all the bytes are processed, the contents of the memories is as depicted in Fig. 3(b). Continuing in a similar way, the contents of memories after 2nd and 3rd rounds are shown in Fig. 3(c) and Fig. 3(d), respectively. Thereafter, the memory contents return to its initial state and the cycle starts over.

3. REGISTER-BASED BYTE PERMUTATION UNITS

Register-based BPUs are constructed based on Parhi's design methodology for data format converters proposed in [9]. Such design methodology results in one-dimensional permutation units, where a shift register is used for delaying data. The data is reordered by circulating the bytes from the last register backwards with the aid of multiplexers. This backward allocation is possible since certain bytes are read out earlier, i.e., bypassed with multiplexers placed at the output. Since the backward allocation and bypassing are in balance, there exist no deadlocks, and each time a byte needs to be backward allocated there is an empty slot available.

Let us begin with a one-port byte permutation unit. By following the design methodology in [9], the structure depicted in Fig. 4(a) is obtained. The one-port BPU is capable of performing both left and right shift byte permutations with the given control signals c_i. Note that with c_i value of zero, the uppermost byte in the input is passed to the output. The latency of the network is 12 cycles, which is the maximum distance a data byte has to be moved in the permutation. Thus, 12 registers are needed in minimum, which is the lower bound for register complexity. The operation

of the network is continuous so there is no need for empty cycles and the next 128-bit sequence can be fed in after the last byte of the previous 128-bit sequence. The depicted control signals are given for permuting a single 128-bit sequence where clock cycle $t = 0$ is the time instant when the first byte is at the input of BPU.

The two-port BPU depicted in Fig. 4(b) takes in two bytes at a time and has a latency of six cycles. Also this BPU is capable of performing both the left and right shift permutations with the given control signals. Similarly, we obtain four- and eight-port BPUs, depicted in Fig. 4(c) and Fig. 4(d), respectively. Both the structures support left and right shift permutations.

The presented register-based byte permutation units were obtained based on the general design methodology for data format converters in [9]. This methodology is well suitable for ShiftRows permutations since there is no need to exchange bytes between parallel delay lines. All the resulting structures require only the minimum number of registers, as stated in [9]. As the number of registers in the proposed BPUs is less than required for storing all the 16 bytes, some pipeline stages need to be placed into final AES structures. Pipelining is however often used in AES implementations and is likely to provide some performance gain by shortening the critical path.

4. COMPARISON

In this section, a short comparison of the 4-port BPUs is made and the results are shown in Table 1. The Proposed 1 corresponds to 4-port memory-based BPU and the Proposed 2 to 4-port register-based BPU depicted in Fig. 4(c). None of the structures except the proposed ones are scalable to the other number of ports. In addition, the structures proposed in [6, 5] are not applicable to decryption since no right shift operation is supported. If such support were adopted, it would increase the complexity of multiplexing. In Table 1, the multiplexing complexity is estimated by converting all multiplexers to equivalent 8-bit 2-to-1 multiplexers, e.g., a 32-bit 4-to-1 multiplexer is equivalent to 12 8-bit 2-to-1 muxes. As seen, the number of registers in the Proposed 1 is less than in the other realization. Similarly, the size of memory in the Proposed 2 is half of the scheme in [4]. The register-based BPU in [4] has less multiplexers than the Proposed 1 because the bytes are not backward allocated. This requires more registers and implies also greater latency. Also in [6, 5] less multiplexers is needed than in Proposed 2, which is due to the fact that they do not support both left and right ShiftRows permutations. It is possible to configure the Proposed 2 BPU in Fig. 4(c) for left shift permutations which would reduce the number of multiplexers to 10. In that case the multiplexers corresponding the control signals c_0, c_1, c_2 are left out and c_3 is reduced to an 8-bit 2-to-1 multiplexer.

Table 1: Comparison of 4-port BPUs. # regs: number of 8-bit registers. mem: size of memory in bytes. # muxes: number of 8-bit 2-to-1 multiplexers.

	# regs	mem	# muxes	shift left/right	scalable
[4] (memory)	–	32	–	yes/yes	no
[4] (register)	28	–	9	yes/yes	no
[6]	28	–	12	yes/no	no
[5]	16	–	12	yes/no	no
Proposed 1	–	16	–	yes/yes	yes
Proposed 2	12	–	14	yes/yes	yes

5. CONCLUSIONS

In this paper, efficient byte permutation units for compact AES structures of any cipher key length were proposed. The units are divided into memory- and register-based structures based on the type of storage. All units are area-efficient since they use only the minimum amount of storage elements. They support folded AES structures where the number of parallel computed bytes Q can be $Q = \{1, 2, 4, 8\}$. It was also shown that compared to other existing approaches for byte permutations in compact AES implementations, the proposed units resulted in less number of registers and memory.

REFERENCES

[1] *Advanced Encryption Standard*, National Institute of Standards and Technology Std., Nov. 2001.

[2] *Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for low-rate wireless personal area networks*, IEEE Std., 2003.

[3] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 4, pp. 545–557, Aug. 2001.

[4] P. Chodowiec and K. Gaj, "Very compact FPGA implementation of the AES algorithm." in *CHES*, ser. Lecture Notes in Computer Science, C. D. Walter, Ç. K. Koç, and C. Paar, Eds., vol. 2779. Springer, 2003, pp. 319–333.

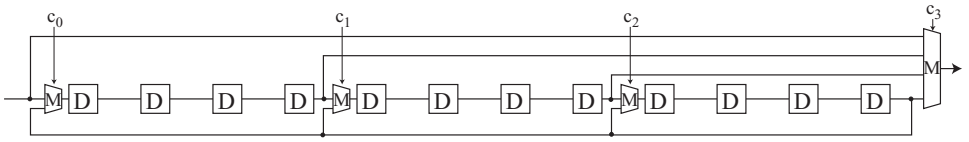
[5] S. McMillan and C. Patterson, "Jbits^{tm} implementations of the advanced encryption standard (Rijndael)." in *FPL*, ser. Lecture Notes in Computer Science, G. J. Brebner and R. Woods, Eds., vol. 2147. Springer, 2001, pp. 162–171.

[6] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact Rijndael hardware architecture with S-box optimization," in *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*. Springer-Verlag, 2001, pp. 239–254.

[7] V. Rijmen. Efficient implementation of the Rijndael S-box. [Online]. Available: www.esat.kuleuven.ac.be/∼rijmen/rijndael/sbox.pdf

[8] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi, "Efficient Rijndael encryption implementation with composite field arithmetic," in *CHES '01: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*. Springer-Verlag, 2001, pp. 171–184.

[9] K. K. Parhi, "Systematic synthesis of DSP data format converters using life-time analysis and forward-backward register allocation," *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, vol. 39, no. 7, pp. 423–440, Jul. 1992.



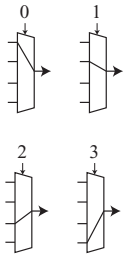
(a)

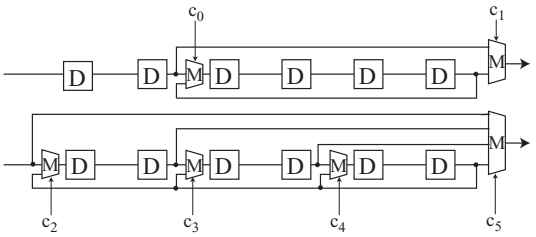
shift left

	t = 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
c_0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
c_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0
c_2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0
c_3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	1	0	3	2	1	1	3	2	3	2	3	3	3	3

shift right

	t = 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
c_0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
c_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0
c_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	0	0
c_3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	0	1	2	3	1	1	2	3	2	3	2	3	3	3





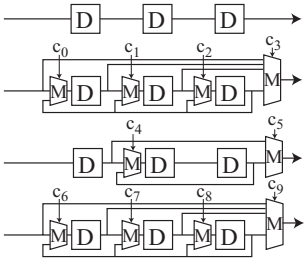
(b)

shift left

	t = 0	1	2	3	4	5	6	7	8	9	10	11	12
c_0	0	0	0	0	0	0	1	0	1	0	0	0	0
c_1	2	2	2	2	2	2	0	2	0	2	2	2	2
c_2	0	0	0	0	0	0	0	1	0	0	0	0	0
c_3	0	0	0	0	0	0	0	0	0	1	0	0	0
c_4	0	0	0	0	0	0	1	0	1	0	1	1	0
c_5	3	3	3	3	3	3	2	0	2	1	2	2	3

shift right

	t = 0	1	2	3	4	5	6	7	8	9	10	11	12
c_0	0	0	0	0	0	0	1	0	1	0	0	0	0
c_1	2	2	2	2	2	2	0	2	0	2	2	2	2
c_2	0	0	0	0	0	0	1	0	0	0	0	0	0
c_3	0	0	0	0	0	0	0	0	1	0	0	0	0
c_4	0	0	0	0	0	0	0	1	0	1	1	1	0
c_5	3	3	3	3	3	3	0	2	1	2	2	2	3



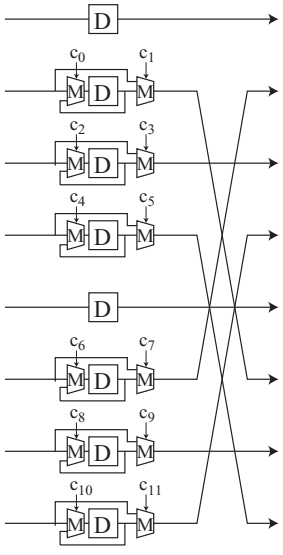
(c)

shift left

	t = 0	1	2	3	4	5	6
c_0	0	0	0	0	0	0	0
c_1	0	0	0	0	0	0	0
c_2	0	0	0	1	1	1	0
c_3	1	1	1	0	0	0	1
c_4	0	0	0	1	1	0	0
c_5	1	1	1	0	0	1	1
c_6	0	0	0	1	0	0	0
c_7	0	0	0	0	1	0	0
c_8	0	0	0	0	0	1	0
c_9	3	3	3	0	1	2	3

shift right

	t = 0	1	2	3	4	5	6
c_0	0	0	0	1	0	0	0
c_1	0	0	0	0	1	0	0
c_2	0	0	0	0	0	1	0
c_3	3	3	3	0	1	2	3
c_4	0	0	0	1	1	0	0
c_5	1	1	1	0	0	0	1
c_6	0	0	0	0	0	1	0
c_7	0	0	0	0	0	0	0
c_8	0	0	0	0	0	0	0
c_9	3	3	3	0	1	2	3



(d)

shift left

	t = 0	1	2
c_0	0	1	0
c_1	1	0	1
c_2	0	1	0
c_3	1	0	1
c_4	0	0	0
c_5	1	1	1
c_6	0	0	0
c_7	1	1	1
c_8	0	1	0
c_9	1	0	1
c_{10}	0	1	0
c_{11}	1	0	1

shift right

	t = 0	1	2
c_0	0	0	0
c_1	1	1	1
c_2	0	1	0
c_3	1	0	1
c_4	0	1	0
c_5	1	0	1
c_6	0	1	0
c_7	1	0	1
c_8	0	1	0
c_9	1	0	1
c_{10}	0	0	0
c_{11}	1	1	1

Figure 4: Block diagrams of register-based byte permutation units (BPU) with multiplexer control. (a) 1-port BPU, (b) 2-port BPU, (c) 4-port BPU, and (d) 8-port BPU. D: 8-bit register. M: 8-bit multiplexer. c_i: control signal. t: clock cycle.

PUBLICATION 4

P. Hämäläinen, M. Hännikäinen, T. D. Hämäläinen, and J. Saarinen "Hardware Implementation of the Improved WEP and RC4 Encryption Algorithms for Wireless Terminals," in *Proceedings of the 10th European Signal Processing Conference (EUSIPCO 2000)*, vol. 4, Tampere, Finland, Sept. 5–8, 2000, pp. 2289–2292.

HARDWARE IMPLEMENTATION OF THE IMPROVED WEP AND RC4 ENCRYPTION ALGORITHMS FOR WIRELESS TERMINALS

Panu Hämäläinen, Marko Hännikäinen, Timo Hämäläinen, and Jukka Saarinen
Digital and Computer Systems Laboratory, Tampere University of Technology
Hermiankatu 3 A, FIN-33720 Tampere, FINLAND
Tel: +358 3 365 2111, Fax: +358 3 365 4575
e-mail: panu.hamalainen@tut.fi, marko.hannikainen@tut.fi, timo.d.hamalainen@tut.fi, jukka.saarinen@tut.fi

ABSTRACT

This paper presents hardware implementations for Improved Wired Equivalent Privacy (IWEP) and RC4 ("Ron's Cipher #4") encryption algorithms. IWEP is a block algorithm providing light-strength encryption. The algorithm has been designed for a new Wireless Local Area Network (WLAN), called TUTWLAN (Tampere University of Technology Wireless Local Area Network). On the contrary RC4, developed by RSA Data Security, Inc., is a powerful stream algorithm used in many commercial products. It is also utilized in the Wired Equivalent Privacy (WEP) standard algorithm for WLANs. The objective of this work has been to study the suitability of hardware implementation for these previously software-implemented ciphers. Hardware is needed to replace software especially in wireless multimedia terminals, in which real-time data processing and limited on-chip memory sizes are key elements. The implementations are made in Very high-speed integrated circuit Hardware Description Language (VHDL) on Xilinx Field Programmable Gate Array (FPGA) chips.

1 INTRODUCTION

Wireless local area network (WLAN) technology is seen very promising for various types of wireless indoor and limited outdoor communications, such as ad hoc networking and for continuous access to company network servers. WLANs provide network solutions when wired networks become impossible or inconvenient. Typical examples are networks that are set up on temporary basis and fixed networks in buildings in which cabling is not reasonable.

Even though wireless connections make networks very flexible and more convenient, they also bring an important issue into closer consideration: data security. The reason to this is that WLANs are far easier to eavesdrop compared to the traditional cable networks. At its simplest level, data is available to anyone within the range of a transmitting device. Therefore, these networks are very sensitive to security violations and need powerful encryption. On the other hand, because of the real-time requirements and limited processing capabilities of small terminals (e.g. on-chip memory), encryption methods used should also be efficient and simple. To avoid exceeding processing limits without losing the reliability of encryption, it would be profitable to implement the algorithms in low-cost hardware, and thereby more capacity would be left to user applications.

In this paper the Xilinx FPGA (Field Programmable Gate Array) implementations of two encryption algorithms, Improved Wired Equivalent Privacy (IWEP) and RC4 ("Ron's Cipher #4") are presented. The first algorithm (i.e. cipher) was developed at Tampere University of Technology (TUT) [1]. It was designed considering hardware implementations for the light encryption of time-critical data in TUTWLAN system [2]. Therefore, the results shown here are very promising. On the contrary, RC4's software-oriented design leads to results nearly equal to its software implementations. This cipher was developed by RSA Data Security, Inc. and is used in several commercial products.

The paper is organised as follows. At first, the IWEP algorithm and implementation results are introduced, followed by RC4 implementation. Discussion of the results is aroused in the concluding section.

2 IMPROVED WIRED EQUIVALENT PRIVACY (IWEP) IMPLEMENTATION

High level illustration of IWEP is presented in Figure 1. Despite its name the cipher is considerably different from Wired Equivalent Privacy (WEP) algorithm included into IEEE 802.11 standard for WLANs [3]. The WEP standard utilizes RC4 stream cipher. However, IWEP is a block cipher.

IWEP encrypts data in 64-bit blocks and uses a 64-bit key. The data ciphering is firmly based on permutation and exclusive-or (XOR) operation. The encryption consists of three iteration rounds, every round including a XORing and a permutation round. In the encryption and decryption operations, both the data block and the key are divided into eight 8-bit subblocks used in the iteration rounds. More information about the algorithm can be found in [1].

The IWEP implementation consists of three parts: single XOR item, XORing round (includes four XOR items), and

IWEP itself (includes three XORing rounds and data permutations). Encryption and decryption blocks are almost similar: the differences are inside the single XOR item and in addition the data permutations are done backwards. As both the encryption and decryption result into almost the same amount of gates, only encryption is discussed here.

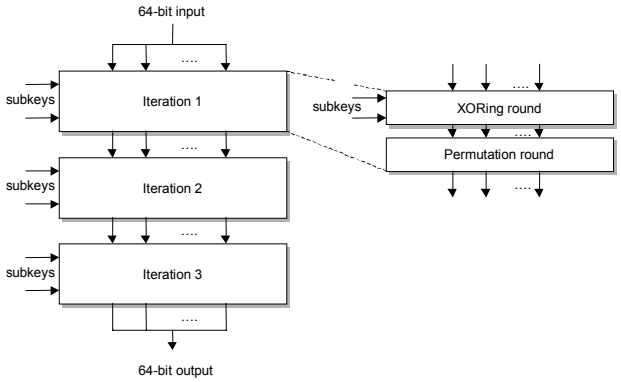


Figure 1. High level structure of Improved WEP.

The implementation was made in Very high-speed integrated circuit Hardware Description Language (VHDL) on Xilinx XC4000E [4] series chips. The used software was a PC version of Xilinx Foundation F1.5 [5]. Parts that fit to the XC4000E-4010EPQ160-2 were tested on the chip while others were only simulated. The on-chip testing was made on TUTWLAN demonstrator platform [6] using HP 16702A pattern generator/logic analyzer [7, 8].

1.1 Single XOR item

The single XOR item was implemented as shown in Figure 2. Each input and output is an 8-bit subblock of the actual data. The output is achieved one clock cycle after setting up the inputs.

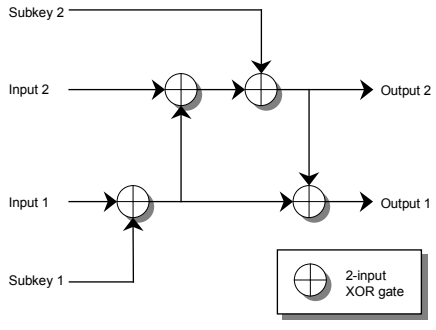


Figure 2. A Single XOR item.

This entity is very fast. It was estimated to work properly with up to 93-MHz clock on the used FPGA chip. More detailed results given by Xilinx Foundation are presented in Table 1.

Table 1. The implementation results of a single XOR item on Xilinx XC4000E-4013EPQ208-2 chip.

Quantity	Value
External I/O Buffers	49/160 (30%)
Configurable Logic Blocks (CLBs)	8/576 (1%)
CLB Flops	0/1,152 (0%)
4-input Look Up Tables	16/1,152 (1%)
Gate Count	192
Maximum Frequency	93.629 MHz
Data Throughput with Max. Freq.	187 MB/s
Data Throughput with 40-MHz clock of the TUTWLAN platform	80.0 MB/s

1.2 XORing round

The XORing round includes four parallel XOR items described above. Each item is fed with two 8-bit parts of the 64-bit data and similarly with two 8-bit parts of the 64-bit encryption key (see Figure 3). Since in one XORing round there are only four XOR items, the output here is also achieved after single clock cycle.

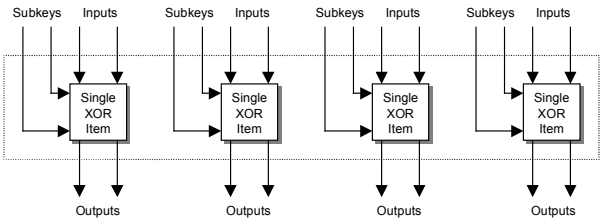


Figure 3. The implementation of one XORing round.

The single XORing round block was estimated to be able to work properly with 92 MHz. The slight decrease in efficiency compared to a single XOR item is due to the fact that four times larger logic was produced (32 CLBs). In both cases the critical path is the same. In this case the key input was excluded because of the limited amount of the input pins on the FPGA.

1.3 IWEP

The highest layer of the implementation is the IWEP module itself. In this module there are three layers of XORing rounds as defined earlier. The signals between the layers are connected according to the desired permutation.

Since it takes one clock cycle for each XORing round to produce its output, the whole encryption for the *first* message block takes three clock cycles. However, because there are registers after each XORing round it is possible to pipeline the process. Applying this means that a new data block can be fed to the cipher on every clock tick and (after the first three clock cycles) a new ciphertext block is received on every clock tick. Therefore, it can be estimated that encryption/decryption with IWEP takes only one clock cycle per block. The pipeline is depicted in Figure 4.

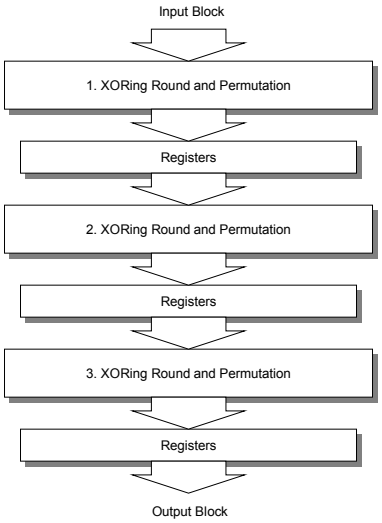


Figure 4. IWEP pipeline.

Table 2 presents the IWEP implementation results. As shown in the table, it was estimated that encryption will work properly with up to 86-MHz clock. Also here the key was held constant.

Table 2. The implementation results of IWEP on Xilinx XC4000E-4013EPQ208-2 chip.

Quantity	Value
External I/O Buffers	129/160 (80%)
Configurable Logic Blocks (CLBs)	64/576 (11%)
CLB Flops	128/1,152 (11%)
4-input Look Up Tables	96/1,152 (8%)
Gate Count	1,728
Maximum Frequency	86.081 MHz
Data Throughput with Max. Freq.	690 MB/s
Data Throughput with 40-MHz clock of the platform	320 MB/s

Because each XORing round in IWEP module is exactly similar, it was possible to implement the 64-bit cipher with only one XORing layer by adding a state machine that feeds the layer with correct inputs (implements the permutations). This way the module was expected to get smaller and more space on the chip was assumed to be saved for other applications. However, this result was not reached. Instead the implementation took more space (2166 gates), and the maximum clock frequency was markedly decreased down to 36 MHz.

Since the maximum clock frequency of the pipeline implementation is so high, one more improvement could possibly be achieved by removing the registers between the iteration rounds. This way the whole encryption/decryption could be executed during one clock cycle. This would increase also the throughput of short messages to the level of larger messages (i.e. only one clock cycle per one 64-bit block). The clock frequency should still be close to the target of 40 MHz.

3 RC4 IMPLEMENTATION

RC4 is a stream cipher, which means that it operates plaintext a single byte at a time. The high level structure of RC4 is shown in Figure 5. The algorithm consists of two main elements. In the figure, *F* denotes a key dependent function that initializes and mixes up the 256-byte memory array called substitution box (S-box). *Index* refers to the memory address and *data* to the stored or read byte.

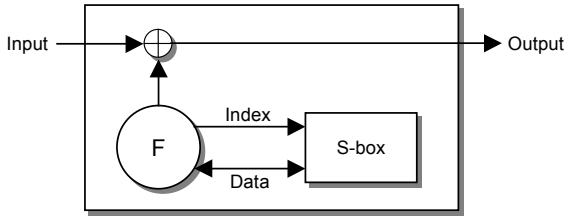


Figure 5. RC4 high level illustration.

The main idea of the algorithm is to produce an 8-bit pseudo random number series initialized by the given key. An encrypted output is then achieved by XORing the random bytes with the input data bytes. In this cipher the encryption and decryption methods are exactly similar and therefore the presented implementation can be used in either way. More detailed information on the algorithm can be found in [9].

RC4's FPGA implementation consists of three modules: a 256-byte memory array and two state machines. The memory array (S-box) is used for producing a pseudo random number stream. The state machines are needed for performing the initialization and mixing up of the array. The implementations were made using the same chips and tools as in IWEP implementation.

1.4 Substitution Box

A Xilinx design tool LogiBLOX [5] was used for implementing the S-box. The result was a 256-byte RAM module with asynchronous read operation and synchronous write operation. By using the tool it was possible to take advantage of the FPGA chip's internal RAM blocks. To test the functionality of the memory module, it was implemented and simulated without any other logic. The area and timing results are presented in Table 3.

Table 3. The implementation results of the substitution box on Xilinx XC4000E-4013PQ208-2.

Quantity	Value
External I/O Buffers	25/160 (15%)
Configurable Logic Blocks (CLBs)	88/576 (15%)
CLB Flops	0/1,152 (0%)
4-input Look Up Tables	176/1,152 (15%)
3-input Look Up Tables	72/576 (12%)
32X1 RAMs	64
Gate Count	8,516
Maximum Frequency	48.955 MHz

1.5 RC4 Cipher

The encryption function of the algorithm was implemented with two separate state machines: one for initializing the S-box and the other for the encryption itself. On the top level these modules were attached to the S-box to generate the cipher. The outcome of the resulting logic for the complete cipher is shown in Table 4. The presented throughput with 40 MHz is only theoretical because the attained maximum frequency is much lower.

Table 4. The implementation results of RC4 cipher on Xilinx XC4000E-4013EPQ208-2 chip.

Quantity	Value
External I/O Buffers	148/160 (92%)
Configurable Logic Blocks (CLBs)	255/576 (44%)
CLB Flops	142/1,152 (12%)
4-input Look Up Tables	444/1,152 (38%)
3-input Look Up Tables	103/576 (17%)
32X1 RAMs	64
Gate Count	11,372
Maximum Frequency	17.744 MHz
Data Throughput with Max. Freq. (after setup)	2.22 MB/s
Data Throughput with 40-MHz clock (after setup, theoretical)	5.00 MB/s

To find out if the results could be improved, the state machines were combined into a single unit. In fact, improvement was achieved in the number of reserved CLBs. However, this approach did not lead to any faster implementation because the maximum clock frequency was slightly decreased. Table 5 presents the results.

Table 5. The implementation results of RC4 with one state machine on Xilinx XC4000E-4013EPQ208-2 chip.

Quantity	Value
External I/O Buffers	148/160 (92%)
Configurable Logic Blocks (CLBs)	224/576 (38%)
32X1 RAMs	64
Gate Count	10,653
Maximum Frequency	16.940 MHz
Data Throughput with Max. Freq. (after setup)	2.12 MB/s
Data Throughput with 40-MHz clock (after setup, theoretical)	5.00 MB/s

4 DISCUSSION

As stated before, IWEP's hardware-oriented design made the implementation simpler and more efficient than that of RC4. Therefore, IWEP was considered very suitable for hardware implementations and especially for WLAN applications with limited processing capacity (only one clock cycle per 64 bits of data). However, IWEP is not very reliable against brute-force attack, and therefore it can only be used to encrypt time-critical data, i.e. data that is useful only for a short period of time. On the other hand, the design of IWEP makes the cipher very flexible for reconfiguration. Modifications, like changing of the permutations, altering the subkey and subdata schedule, and adding iteration rounds, can be easily done. With these frequent changes the unpredictability and consequently the security level of the cipher is increased.

Unlike IWEP, RC4 is commonly regarded as a very powerful cipher. Therefore, it is a more suitable alternative for long-term privacy. However, the cost of this improved security is the longer encryption time. Especially the initialization of the cipher is time-consuming. One solution is to implement the RC4's S-box without using a separate memory block with address and data buses, or to use RAM with also asynchronous write operation. This way the several read and write operations of the algorithm do not have such a dominating role.

REFERENCES

[1] K. Salli, T. Hämäläinen, J. Knuutila, J. Saarinen, "Security Design for A New Wireless Local Area Network TUTWLAN", The Ninth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'98), September 8-11, 1998, Boston, USA, pp. 1540-1544.

[2] M. Hännikäinen, J. Knuutila, A. Letonsaari, T. Hämäläinen, J. Jokela, J. Ala-Laurila and J. Saarinen, "TUTMAC: A Medium Access Control Protocol for A New Multimedia Wireless Local Area Network", The Ninth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'98), September 8-11, 1998, Boston, USA, pp. 592-596.

[3] IEEE P802.11 D5.0, "IEEE Standard for local and metropolitan area networks: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications", IEEE Standards Department, 19.7.1996.

[4] "XC4000E and XC4000X Series Field Programmable Gate Arrays", Product Specification, Xilinx, November 1997, USA.

[5] "Foundation Series 1.5i Install and Release Document", Xilinx, December 1998, USA.

[6] M. Kari, M. Hännikäinen, T. Hämäläinen, J. Knuutila and J. Saarinen, "Configurable Platform for a Wireless Multimedia Local Area Network", The 5th International Workshop on Mobile Multimedia Communications (MoMuC'98), October 12-14, 1998, Berlin, Germany, pp. 301-306.

[7] "HP 16522A 200-M Vectors/s Pattern Generator User's Guide", Publication 16522-97005, Hewlett-Packard Company, August 1997.

[8] "HP 16557D 135-MHz State/500-MHz Timing Logic Analyzer User's Reference", Publication 16557-97000, 1st edition, Hewlett-Packard Company, March 1998.

[9] B. Schneier, "Applied Cryptography: Protocols, Algorithms and Source Code in C", 2nd edition, John Wiley & Sons, Inc., 1996, USA, 758 pages.

PUBLICATION 5

P. Hämäläinen, J. Heikkinen, M. Hännikäinen, and T. D. Hämäläinen, “Design of Transport Triggered Architecture Processors for Wireless Encryption,” in *Proceedings of the 8th Euromicro Conference on Digital System Design – Architectures, Methods, and Tools (DSD 2005)*, Porto, Portugal, Aug. 30–Sept. 3, 2005, pp. 144–152.

Design of Transport Triggered Architecture Processors for Wireless Encryption

Panu Hämäläinen, Jari Heikkinen, Marko Hännikäinen, and Timo D. Hämäläinen
Tampere University of Technology / Institute of Digital and Computer Systems
P. O. Box 553, FI-33101 Tampere, Finland
{panu.hamalainen, jari.heikkinen, marko.hannikainen, timo.d.hamalainen}@tut.fi

Abstract

Transport Triggered Architecture (TTA) offers a cost-effective trade-off between the size and performance of ASICs and the programmability of general-purpose processors. In this paper TTA processors for the RC4 and AES encryption algorithms of the new IEEE 802.11i WLAN security standard are designed. Special operations efficiently supporting the ciphers are developed. The TTA design flow is utilized for finding configurations with the best performance-size ratios. The size of the configuration supporting both the algorithms is 69.4 kgates and the throughput 100 Mb/s for RC4 and 68.5 Mb/s for AES at 100 MHz in the 0.13 μm CMOS technology. Compared to commercial processors of the same wireless application domain, higher throughputs are achieved at significantly smaller area and lower clock speed, which also results in decreased energy consumption.

1. Introduction

The performance requirements for security implementations have significantly increased with the communication speeds of data networks. For example, virtual private networks require high-speed implementations in busy corporate firewalls. On the other hand, a need for low-cost and low-power designs has emerged as wireless technologies for embedded, battery-powered devices have been developed. By combining high performance and low power, the security processing requirements can be met with lower energy consumption, resulting in longer operating times.

Whereas Application Specific Integrated Circuits (ASIC) offer the highest performance at low energy and low cost (in high volumes), the design times are long and upgrading is time-consuming and expensive. The flawed IEEE 802.11 Wireless Local Area Network (WLAN) security is a good example of a product in which support for low-cost upgrading would have been beneficial. High performance with reconfigurability and short development times are achieved with Field Programmable Gate Arrays (FPGA). FPGAs are well-suited, e.g, for network routers but unsuitable for low-cost and low-power embedded devices. The embedded solutions should combine the benefits of ASICs and FPGAs by enabling good performance with a certain level of reconfigurability.

In this work a processor architecture called Transport Triggered Architecture (TTA) [4] is scrutinized for finding high-performance and low-cost processor designs with efficient support for wireless encryption. TTA is fully configurable and supports Instruction Level Parallelism (ILP) and application-specific operations. The development tools offer semi-automatic flow for straightforward processor design and implementation. In addition to good encryption performance-cost ratios, the designed processors and their special operations are general-purpose, enabling running other applications in them. Modifications are possible even in fielded devices by software updates. Compared to coprocessor schemes, TTA processors can offer better performance since off-chip communication bottlenecks are avoided.

The studied encryption algorithms are RC4 [22] and Advanced Encryption Standard (AES) [1] of the new IEEE 802.11i WLAN security standard [14]. Whereas the AES-based design is completely new, RC4 is included for backward compatibility with fielded hardware. Thus, support for both the algorithms is required in new products. Since the work of the 802.11i group as well as AES are the bases for a large number of new technologies, the TTA designs of this paper can be efficiently utilized in many other embedded products. For example, the new IEEE 802.15.4 low-rate, low-power personal area network uses AES. Currently, the maximum radio transmission speed of 802.11 is 54 Mb/s.

Generally, ASICs and FPGAs are implementation technologies and TTA is a processor architecture, implementable both in ASIC and FPGA technologies. However, in this work ASIC refers to a dedicated hardware implementation of an encryption algorithm. An ASIC is fixed and can only be utilized to this single task. Similarly, FPGA refers to an FPGA implementation of a single

Proceedings of the 2005 8th Euromicro conference on Digital System Design (DSD'05)
0-7695-2433-8/05 $20.00 © 2005 IEEE

algorithm. A TTA implementation refers to a TTA processor including software describing an algorithm. The processor is implemented in an ASIC technology.

Several specialized reconfigurable hardware architectures have been proposed for symmetric-key cryptography. The designs in [7], [9], [24], and [25] produce excellent results in terms of throughput and reconfigurability but areas are large and the application domain the same as that of FPGAs. References [19] and [10] are likely to achieve good performance-area ratios but they do not report the costs. Area-efficient designs are presented in [26] and [16]. However, they lack the flexibility and/or the automatic design flow of TTA. Implementing AES in an architecture comparable to TTA is studied in [20]. In this work considerably better performance is achieved with small area.

2. TTA Overview

Opposed to traditional, operation-triggered processor architectures, in TTA [4] operations occur as side effects of data transports. The execution begins when data are written to the operand registers of a Function Unit (FU). Only a single instruction, *move*, is required for low-level programming. The mirrored paradigm enables applying new scheduling, bypassing, and resource allocation techniques in high-level language compilers.

The TTA central processing unit is organized as a set of FUs and Register Files (RF). The data transfers between the entities are performed by an interconnection network consisting of a variable number of buses and bus connections. A FU may be anything between a simple shifter and a large Arithmetic Logic Unit (ALU). At least one FU operates as a Load-Store Unit (LSU) handling memory accesses.

The number of FUs, FU operands and results, RFs, RF ports, buses, and bus connections can be changed unlimitedly. The changes enable variable amount of ILP. TTA also allows adding special, application-specific operations implemented in Special Function Units (SFU). Currently, bus widths 1 and 32 bits are supported and 32-bit ALU operations have been implemented in the standard set of the FU operations. The buses are only simple connections multiplexed with AND-OR networks. Thus, even a larger number of buses does not excessively increase the processor size.

The TTA development environment [18], the *MOVE framework*, provides semi-automatic design process and short design times for application specific processors. The *MOVE compiler* translates the ANSI C/C++ description of the application into sequential MOVE code. The compiler supports all the described modifications. The *MOVE scheduler* is used for scheduling the sequential code to a specific TTA configuration. The *MOVE simulator* is provided for verifying and evaluating both the sequential and parallel codes and for assisting the scheduler.

Finding the best suited TTA configuration for a given application by hand is a time-consuming and error-prone task. Thus, the framework contains the *design space explorer* for automatically testing the suitability of a large number of TTA configurations for the application. The explorer evaluates the execution time, size, and energy consumption[1] and outputs configurations with the best values in all three categories. For the evaluations it utilizes the *MOVE estimator* which computes the measures according to presynthesized TTA resources. The exploration is performed in two phases. Initially, the explorer is given a large configuration with excessive amount of resources. The first phase reduces the resources but maintains the interconnect fully connected. In the second phase the unnecessary connections are removed from the configuration chosen by the designer.

The VHDL description for a TTA configuration can be automatically generated using the *MOVE Processor Generator* (MPG). The designer only has to provide the VHDL descriptions of SFUs. The TTA processor can be synthesized with third party synthesis tools.

3. Wireless Encryption Algorithms

The encryption algorithms of the 802.11i WLAN security standard [14] scrutinized for TTA implementations are RC4 and AES. The standard requires only the forward functionality of AES for en/decryption. Thus, the invert cipher is not discussed in this paper. However, with small modifications the TTA designs can be tuned for efficiently supporting also the inverted AES.

3.1. RC4

RC4 [22] produces a pseudo-random byte stream and en/decryption is performed by XORing the stream and the data. The encryption key size is between 1 and 256 bytes. Processing consists of two phases, initialization and pseudo-random byte generation. The internal state of the cipher is maintained in a 256-byte arrays S. During the initialization S is first linearly filled with the values from 0 to 255. The encryption key is used for scrambling the array. The phase is required for every new key.

The pseudo code of RC4 en/decryption (random byte generation) is presented in Fig. 1. The process uses two indices, i and j, for modifying the contents of S. The data is input and output in the byte array *data*. The process runs until all the bytes of the input are processed. During the random byte generation two entries in S are read and their locations are swapped. The third entry $S[t]$ is the produced

1 Currently, the evaluation of energy consumption is under development. Thus, the energy measures are not considered in this work.

```
rc4(data) {
  i = 0; j = 0; k = 0;
  while not end of data {
    i = (i + 1) mod 256;
    j = (j + S[i]) mod 256;
    swap(S[i], S[j]);
    t = (S[i] + S[j]) mod 256;
    data[k] = S[t] xor data[k];
    k = k + 1; } }
```

Figure 1. RC4 en/decryption.

pseudo-random byte. The operations involved are addition modulo 256, memory access (read and write), and XOR.

3.2. AES

The AES algorithm [1] is a symmetric cipher that encrypts data in 128-bit blocks. It supports key sizes of 128, 192, and 256 bits. AES consists of successive, similar iteration rounds. Depending on the key size, the number of the rounds is 10, 12, or 14. Each round mixes the data with a 128-bit *roundkey*, which is generated from the encryption key. Currently, the 128-bit-key version is generally considered to provide adequate security. Decryption requires inverting the iterations resulting in at least partly separate data path. Even though only the 128-bit-key version is studied in the paper, the designs support also the others. The 802.11i encryption mode requires two AES passes per block of data.

The AES round operations are presented in Fig. 2. The cipher maintains an internal, 4-by-4 matrix of bytes, called *state*, on which the operations are performed. Initially, *state* is filled with the input data block XORed with the encryption key. Each round, except the last one, contains operations called *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*. The last round bypasses MixColumns.

SubBytes is an invertible, nonlinear transformation. It uses 16 similar 256-byte substitution tables (*S-box*) for independently mapping each byte of *state* into another byte. S-box entries are generated by computing multiplicative inverses in *Galois Field* GF(2^8) and applying an affine transformation. SubBytes can be implemented either by computing the substitution [21] or using table lookups [13]. ShiftRows is a cyclic left shift of the second, third, and fourth row of *state* by 1, 2, and 3 bytes, respectively. MixColumns performs a modular polynomial multiplication in GF(2^8) on each column. Instead of computing, SubBytes and MixColumns can also be combined into four large tables, called *T-boxes*. Performing the operations requires table lookups and XORing. The size of a T-box is 1,024 bytes. During each round AddRoundKey performs XOR with *state* and the roundkey. Roundkey generation includes S-box substitutions, word rotations, and XOR operations performed on the encryption key.

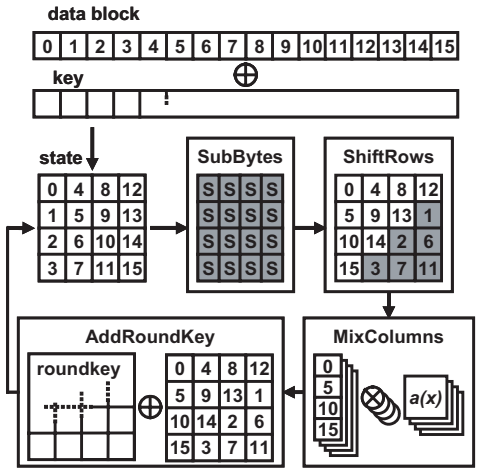


Figure 2. AES round operations.

4. Design Methodology

The design flow of the MOVE framework is utilized for developing TTA processors that efficiently support the two encryption algorithms. The analysis throughout the paper only concentrates on the encryption cores. Even though supported by the designed processors, the RC4 initialization and the AES roundkey generation are not examined. In order to estimate the possibilities for the utilization of ILP, the data dependencies of the algorithms are analyzed first. The parallelism analysis is valuable in estimating the amount of initial resources to be allocated for each algorithm and also for evaluating the results produced by the framework.

Next, the ciphers are scrutinized to identify operations that could be efficiently accelerated with SFUs. In this paper a requirement in the design of SFUs is that they can also be benefited from in other applications. A strictly application-specific SFU is added only if it has a significant effect on performance without a large processor area increase. For example, even though possible, implementing the complete AES cipher as an SFU was not considered. This finer-grained method produces a larger number of TTA configurations with different sizes and performances.

After the analyses the MOVE explorer is utilized for finding TTA configurations with the highest quality for the algorithms, with and without SFUs. The explorations are performed on the encryption cores. The *quality* of a design is defined as a throughput-size ratio

$$Q = T/S, \tag{1}$$

in which T is throughput in Mb/s and S is size in kgates. Hence, the unit of quality is Mb/s/kgate.

Even though tuning TTA for the encryption cores, the hardware is maintained general-purpose in order to allow running the rest of the cipher and other applications in it.

Only after finding the highest-quality configurations, they are further tuned to show how much the quality could be improved if the core was the only target application.

Two types of general-purpose, standard FUs are used in the designs, ALUs and LSUs. The ALU contains addition, substraction, logical operations, comparisons, shifts, and zero/sign extensions. Multiplication and division are not included. The size of an ALU is 3,019 gates and latency one clock cycle. The size of a LSU size 1,140 is and read/write latency three clock cycles. A multiplier FU would add 3,120 gates and a divider 4,160 gates to the processor size. The TTA processor size and throughput values in the paper are for a 0.13 μm CMOS standard-cell technology, synthesized for a 100 MHz system clock. However, at least 250 MHz clock frequency is achievable using the TTA design flow [11]. The complete TTA processor size estimations do not include data or instruction memories. The term *estimated size* refers to the size computed by the MOVE estimator and *synthesized size* to the size produced by the Synopsys Design Vision 2003.06 synthesis tool. The results will show that the estimated size is very close to the synthesized size.

4.1. Comparison to Other Platforms

In order to evaluate the quality of the results, the TTA designs are compared to 8051, ARM7 and ARM9 processor as well as FPGA and ASIC implementations of the algorithms. 8051 is a 8-bit microcontroller and the ARMs are 32-bit Reduced Instruction Set Computer (RISC) processors utilized in embedded systems, especially in wireless platforms. Similarly to TTA, memories are not included in the size estimates. The 8051 features are derived from [8]. 100 MHz system clock speed is achievable with the synthesizable design in a 0.25 μm technology. Generally, 8051 chips are run at much lower speed, around 15 MHz.

The ARM size estimates are based on ARM7TDMI and ARM968E-S cores without caches [2]. According to [15], the gate count of a design is

$$g = \frac{A}{320F^2}, \tag{2}$$

in which A is the total area of the standard-cell layout and F is the minimum feature size of the technology. At F = 0.13 μm the area A = 0.26 mm^2 for ARM7 and A = 0.59 mm^2 for ARM9. The clock speeds are 133 MHz and 220 MHz [2].

The RC4 performance in 8051 was estimated with the assembly implementation in [17]. The ARM performances were measured for the RC4 source code in [23], which is also used as the RC4 application in TTA. ARM Developer Suite 1.2 was utilized for producing speed-optimized measures. The RC4 implementation presented in [12] is used as the FPGA reference. The design was resynthesized to a newer Xilinx FPGA, XC2V250FG256-4. The equivalent gate count reported by the design tool is large due to the usage of embedded block RAMs for the S array. The VHDL design was also synthesized to the 0.13 μm technology at 200 MHz for producing an ASIC reference.

Platform	Size (kgates)	Clock (MHz)	RC4 (Mb/s)	AES (Mb/s)
8051	10.0	100	8.3	1.0
ARM7	48.1	133	36.7	14.2
ARM9	109.0	220	83.8	35.6
FPGA [12]	68.0	103	103.0	-
ASIC	3.0	200	200.0	-
FPGA [13]	670.0	134	-	1720.0
ASIC [21]	5.4	131	-	311.0

Table 1. Reference implementations.

The 8051 performance for the 128-bit AES is derived from [5]. In [8] an instruction takes four oscillator cycles. The ARM performances were measured for the C source code in [6]. The source was also used as the AES application in TTA. The code is one of the fastest for the *gcc* compiler on which the MOVE compiler is also based. About 17% better ARM performance compared to the ARM9 measure of this paper is achieved in [3]. Unfortunately, the processor size is not reported for quality comparison.

The 128-bit-key encryption implementation containing the key generation logic in [13] is taken as the AES FPGA reference, resynthesized to the newer FPGA. Block RAMs were used for the S-boxes. The smallest AES implementation in [21] is used as the ASIC reference. The design contains encryption, decryption, and 128-bit key generation. According to the authors knowledge, the paper presents the highest-quality ASIC implementation. The reference implementations are summarized in Table 1.

5. TTA Designs for RC4

The operation dependencies in the random byte generation loop of RC4 are shown in Fig. 3. The operations presented in parallel are independent and can be executed simultaneously. The figure shows, as also stated in [23], that almost every statement of the algorithm depend on the statement immediately before it. Hence, the benefits of ILP for RC4 acceleration are very limited. Only the two writes of the swap operation and the computation of the index t can be performed in parallel.

More parallelism can only be found by unrolling the RC4 loop [23]. However, unrolling requires additional comparisons before the XOR operation (Fig. 1) if the byte length of the input does not always match the length of the produced pseudo-random stream. Thus, manual unrolling was not considered for RC4.

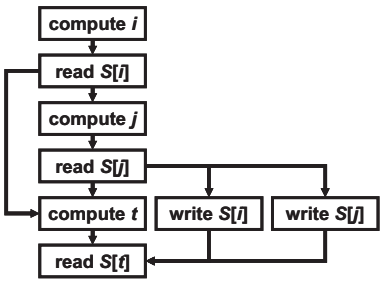


Figure 3. RC4 operation dependencies.

5.1. RC4 Special Operations

Three special operations were designed for RC4 acceleration. Since only 32-bit operations are currently implemented in the standard TTA hardware, a simple byte adder (i.e. modulo 256 adder) was implemented. A byte addition with a 32-bit adder requires masking the result with 0xFF or performing a zero extension, resulting in two clock cycles per addition in TTA. The single-cycle special operation *addb* was included in the ALU using the existing 32-bit adder hardware.

The latency of a memory access through the LSU is three clock cycles in TTA. Since a large portion of the statements in the RC4 loop requires accessing the *S* array, a decrease in the latency has a significant effect on the performance. A separate, single-cycle SFU (LUT-SFU) with read (*sread*) and write (*swrite*) operations was implemented for maintaining *S*. In addition to the short access latency, additional clock cycles are saved since the indices (*i*, *j*, *t*) can be directly used for referencing. Computing the absolute main memory addresses is avoided.

LUT-SFU contains a 256-byte single-port RAM with 8-bit address and data ports. The ports were wrapped inside a 32-bit interface in order to support connections to the 32-bit TTA buses. The size of the synthesized SFU is 2,012 gates. A large number of other encryption algorithms can make use of the SFU for look-up-table functionality [7]. Section 6 will utilize the SFU in the implementation of AES. Units for the same purposes have been designed in [7] and [16].

5.2. Design Space Explorations for RC4

Based on the parallelism analysis and manual experiments with few TTA configurations, it was deduced that two ALUs are sufficient for the RC4 initial configuration in the explorations. Similarly, the suitable amount for the other TTA resources was searched manually. A suitably-sized configuration results in shorter exploration time. The number of LSUs was also limited to two, since only single and dual-port memories were available in the used implementation technology. Also in general, two ports are often the maximum in memory technologies. The rest of the initial resources for the non-SFU configuration were three 16-register integer RFs, two 4-register boolean RFs, and six 32-bit buses. In the SFU configuration the same resources enhanced with the LUT-SFU were used.

5.3. RC4 Results

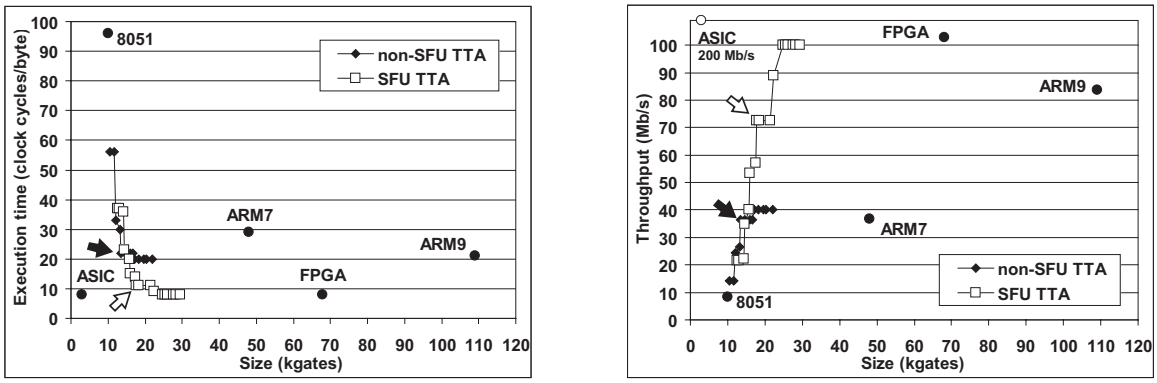
Fig. 4 presents the exploration results for the fully connected configurations output by the explorer. Due to the well-chosen initial configurations, the explorations took only about 20 minutes. The TTA throughputs are for 100 MHz. The arrows point to the TTA configurations with the highest qualities. The figure shows that after adding the special operations, the RC4 performance is significantly increased. In Fig. 4(a) the execution time of the fastest non-SFU configuration is 20 cycles/byte whereas for the fastest SFU configuration it is only 8 cycles/byte. In Fig. 4(b) the quality of a design increases the closer it is to the vertical axis and the further away it is from the horizontal axis. The characteristics of the highest quality configurations are presented in Table 2. The qualities are computed for the synthesized sizes. It can be seen that the estimator produces good results as the estimated sizes fairly precisely match with the synthesized ones. Note that also the RF sizes were reduced.

For comparison Fig. 4 includes RC4 measures for the reference implementations. TTA outperforms 8051 even with an equal-sized non-SFU configuration. Also, the throughput level of ARM7 is achieved at 70% smaller size. The throughput of ARM9 is higher due to the higher clock frequency. The ARM9 throughput is achieved at 100 MHz frequency with an 80% smaller SFU configuration. Even the cycle counts of FPGA and ASIC are reached in TTA with the special operations.

The comparison of the highest quality TTA configurations with the other platforms is presented in Fig. 5. The

Resource	non-SFU	SFU
ALUs	1	1
LSUs	1	1
integer RFs (4 registers)	3	3
boolean RFs (1 register)	1	1
32-bit buses	3	4
SFUs	-	1
Estimated size (kgates)	13.4	17.7
Synthesized size (kgates)	13.5	17.7
Throughput (Mb/s)	36.4	72.7
Quality (Mb/s/kgate)	2.70	4.11

Table 2. Non-SFU and SFU configurations with the highest quality for RC4.



(a) (b)

Figure 4. RC4 exploration results in TTA: (a) execution time against size and (b) throughput against size. The arrows point to the configurations with the highest quality.

figure shows the results for the synthesized highest-quality TTA designs. The configurations *non-SFU-opt* and *SFU-opt* were generated by removing unused ALU operations from the non-SFU and SFU configurations and performing the second exploration phase. The sizes for these were taken from the estimator. In addition to the other processors, the quality of TTA for RC4 is higher than that of FPGA.

6. TTA Designs for AES

Compared to RC4, AES contains much more parallelism. In this paper the parallelism is divided into three levels: *block parallelism*, *round parallelism*, and *sub-round parallelism*. Block parallelism refers to processing several data blocks simultaneously. The applicability depends on the used encryption mode. For example, the inputs of the Counter (CTR) mode are independent from each other and can be processed in parallel. On the other hand, the Cipher Block Chaining (CBC) mode requires that the previous data block is finished before processing the next one. The block parallelism is utilized, e.g., in pipelined hardware implementations.

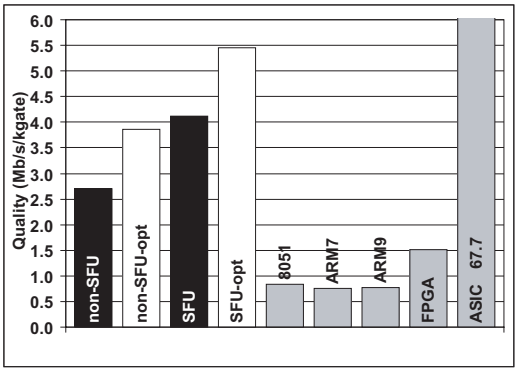


Figure 5. Qualities of RC4 implementations.

Round parallelism exists if some of the cipher rounds are independent from each other. Those rounds could be processed simultaneously. However, the dataflow-oriented AES requires the output of the previous round before the next one begins and round parallelism cannot be utilized.

The parallelism inside a round, sub-round parallelism, can be examined using Fig. 2. In SubBytes each byte-wise S-box substitution can be performed simultaneously. Similarly, in AddRoundKey each *state* byte (even each *bit*) can be independently XORed with the roundkey bytes. MixColumns requires at least a column (a word) of *state* before processing can begin. Despite of the inapplicability of the round parallelism, the sub-round analysis reveals possibilities for interleaving consecutive rounds. As soon as a byte from the previous round is ready, the next S-box substitution for the byte can be performed. The same applies for AddRoundKey. MixColumns for a word can be performed when the four required bytes are produced. The method can also be utilized for interleaving at the data block level in feedback encryption modes.

6.1. AES Special Operations

LUT-SFU designed for RC4 is also used for accelerating AES. In addition, two SFUs were designed, one for converting between byte and word representations (CONV-SFU) and another, AES-specific MIXADD-SFU. Processing an AES round with the special operations is depicted in Fig. 6.

LUT-SFU is included for fast look-up-table functionality. A single LUT-SFU carries out an 8-bit S-box substitution. Higher parallelism compared to a non-SFU design is possible since the number of simultaneous S-box accesses is not limited by LSUs. Initially, each LUT-SFU is filled with the S-box values from the main memory with the *swrite* operation, performed only once. Thereafter, the SFUs are used in read-only mode with the *sread* operation. The SFU can be utilized in the roundkey generation as well as performing the inverted AES operations.

Implementing AES in the 32-bit TTA requires maintaining *state* in four words containing the columns. A 32-bit MixColumns operation uses a byte from each row. The bytes are located in different columns (words) due to the ShiftRows operation. Similarly, the S-box (or T-box) lookups require referencing the table with bytes within the words. Thus, in order to generate the inputs for the operations, the bytes have to be shifted to suitable positions. The outputs are again packed to the words representing the columns of *state*. These conversions result in several shift and logic operations.

Another alternative is to maintain *state* in 16 words, each containing a byte of *state* aligned to the least significant byte of a word. With this scheme, ShiftRows and SubBytes can be implemented very efficiently as each byte can be accessed directly. However, the representation requires four times more moves for transporting *state* between FUs. In addition, it requires four XOR operations per 32 bits in AddRoundKey, totalling 16 XORs. When *state* is stored in four words, only four XOR operations are required.

CONV-SFU was designed for combining the benefits of both the representations. Its *bytes2word* operation can be used for packing four bytes into a word and *word2bytes* for expanding a word into four bytes in a single clock cycle. CONV-SFU was wrapped inside a 32-bit interface for supporting connections to the TTA buses. The size of the SFU is 880 gates in the 0.13 μm technology. It can be used in the invert AES and for performing any byte-wise permutation in other block ciphers. Assembling and decomposing network packets often require similar transformations.

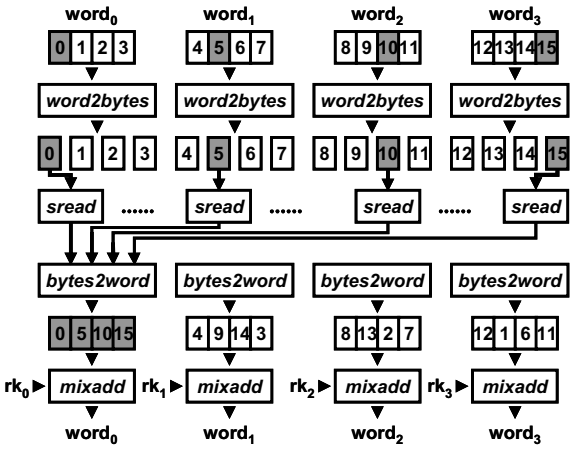


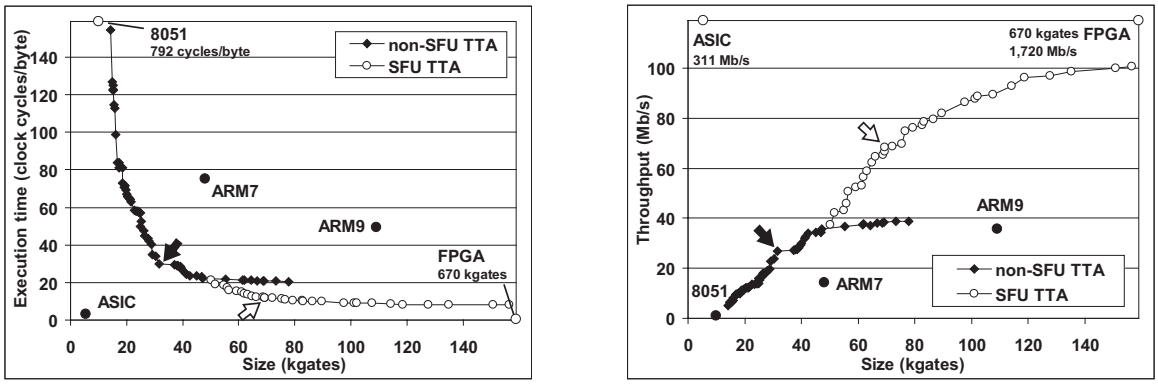
Figure 6. Using special operations for computing an AES round. The value rk_k is the kth word of the current roundkey.

MIXADD-SFU performs a combined 32-bit MixColumns and AddRoundKey operation (*mixadd*) in a single clock cycle. MixColumns is trivial in hardware [1] and can be implemented with reasonably small amount of resources. On the contrary, in a 32-bit processor a large number of shift and logical operations are required [3]. Including the XORing of 32-bits of a roundkey in the unit saves four clock cycles per round compared to XORing in the ALU. The size of MIXADD-SFU is 1,326 gates.

6.2. Design Space Explorations for AES

In the AES source [6] the encryption core is fully unrolled. For shorter scheduling time the unrolling factor was decreased to four rounds. This still allows exploiting interleaving between AES rounds. Means for block level parallelism or block interleaving were not provided in this work. Similarly to RC4, the number of LSUs was limited to two. The number of other resources was increased for supporting the higher parallelism of AES. It was tested that eight ALUs already fully utilize the memory bandwidth in the non-SFU configuration. The number of ALUs was decreased to four in the SFU configuration since most of the processing is performed in the SFUs. The number of SFUs was adjusted to allow processing two words of *state* simultaneously, i.e., eight LUT-SFUs, four CONV-SFUs, and two MIXADD-SFUs were included. This was seen as a reasonable trade-off between available ILP and the maximum processor size. The rest of the resources were three 32-register integer RFs, two 4-register boolean RFs, and 16 32-bit buses.

In the original source code an AES round is implemented with the T-box method and *state* is stored in four words. Thus, the most utilized operations are memory accesses. Shifting and logical operations are required for indexing the T-boxes, permuting bytes, and combining the read values to produce round outputs. The performance is limited by the memory bandwidth. The finer-grained implementation with the SFUs provides more alternatives for the utilization of ILP. It requires much less memory bandwidth. The intermediate computation results can be kept in RFs or directly conveyed to the next FU. The SFU implementation also requires smaller amount of memory. 256-byte S-boxes are used instead of 1,024-byte T-boxes.



(a) (b)

Figure 7. AES exploration results in TTA: (a) execution time against size and (b) throughput against size. The arrows point to the configurations with the highest quality.

6.3. AES Results

The AES exploration outputs for the fully connected configurations are presented in Fig. 7. The explorations took about six hours. The smallest non-SFU configuration is 72% smaller than the smallest SFU configuration. However, the smallest SFU configuration already achieves the performances of the fastest non-SFU configurations. On the other hand, the throughput of the largest SFU configuration is 160% higher than that of the largest non-SFU configuration. It was examined that the non-SFU performance saturates at 20 cycles/byte even if resources (other than LSUs) were added. It is restricted by the memory bandwidth. With the SFUs much higher performances are achievable.

Fig. 7 shows that the throughput of ARM7 is achieved at 48% smaller size with a non-SFU configuration. The throughput of ARM9 is also reachable with a non-SFU TTA at significantly smaller size. The smallest SFU configuration achieves the ARM9 throughput at the size of ARM7. The highest-quality TTA configurations are summarized in Table 3. The non-SFU configuration has 90% higher throughput at 35% smaller size than ARM7 and only 24% lower throughput than ARM9 at 71% smaller area. The size of the SFU configuration is between the ARMs. The throughput is significantly higher.

The dataflow-oriented AES algorithm can be implemented very efficiently in coarse-grained hardware. Therefore, the performances of the FPGA and ASIC designs are not achieved in TTA with the designed SFUs. Due to the fully utilized sub-round parallelism in the FPGA implementation, the throughput is very high, implying also good quality. The ASIC implementation computes 32-bits of a round at a clock cycle, resulting in a compact and very high-quality design. The quality comparison similar to Section 5.3 is presented in Fig. 8.

As the highest-quality AES SFU configuration is a superset of the RC4 configurations, it is very well suited for the 802.11i processing. The RC4 throughput is 100 Mb/s which is enough for the highest transmission speeds. The rate of 54 Mb/s is achieved for the 802.11i AES mode by increasing the clock speed to 158 MHz.

7. Conclusions

TTA processors for efficient encryption in the 802.11i wireless network were designed. The designs outperform the 32-bit commercial processors of the same application

Resource	non-SFU	SFU
ALUs	2	2
LSUs	2	2
integer RFs (12 registers)	3	3
boolean RFs (1 register)	1	1
32-bit buses	6	9
LUT-SFUs	-	4
CONV-SFUs	-	1
MIXADD-SFUs	-	1
Estimated size (kgates)	31.5	71.1
Synthesized size (kgates)	32.9	70.4
Throughput (Mb/s)	27.0	68.5
Quality (Mb/s/kgate)	0.821	0.973

Table 3. Non-SFU and SFU configurations with the highest quality for AES.

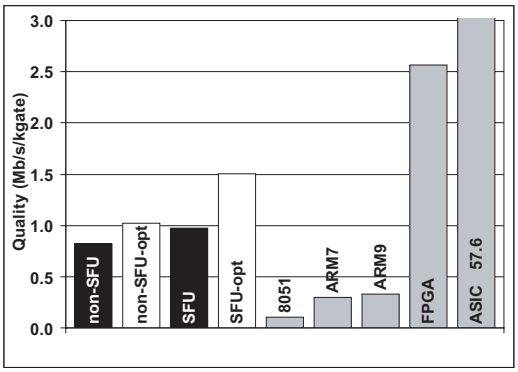


Figure 8. Qualities of AES implementations.

domain with significantly smaller area and lower clock frequency. For RC4 even the cycle counts of the ASIC were achieved. The AES performance can be increased closer to the dedicated hardware by adding resources and tuning the special operations more to the algorithm. In this work the die sizes were kept small and the designs general-purpose for low-cost and for supporting other network processor tasks. As supporting both the algorithms, the AES processor is very well suited for security processing in the embedded 802.11i devices. After the further development of the MOVE framework is finished, the energy consumptions of the designs will also be included in the evaluations.

Acknowledgments

Erno Salminen provided valuable assistance with the SFU design and the presentation of the results.

References

[1] Advanced encryption standard (AES). FIPS-197, 2001.

[2] ARM website. www.arm.com, 2005.

[3] K. Atasu, L. Breveglieri, and M. Macchetti. Efficient AES implementations for ARM based platforms. In *Proc. 2004 ACM Symp. Applied Computing (SAC 2004)*, pages 841–845, Nicosia, Cyprus, Mar. 14–17, 2004.

[4] H. Corporaal. *Microprocessor Architectures from VLIW to TTA*. Wiley & Sons, West Sussex, England, 1998.

[5] J. Daemen and V. Rijmen. AES proposal: Rijndael, 1999.

[6] C. Devine. AES code. www.cr0.net:8040/code/crypto, 2001.

[7] A. J. Elbirt and C. Paar. Instruction-level distributed processing for symmetric-key cryptography. In *Proc. 17th IEEE Int. Symp. Parallel and Distributed Processing (IPDPS 2003)*, pages 78–87, Nice, France, Apr. 22–26, 2003.

[8] Global UniChip Corporation, Taiwan. *UMCU-0051A-T 8-bit Microcontroller-Turbo*, 2001.

[9] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao. The Chimaera reconfigurable functional unit. *IEEE Trans. VLSI Systems*, 12(2):206–217, 2004.

[10] J. R. Hauser and J. Wawrzynek. Garp: A MIPS processor with a reconfigurable coprocessor. In *Proc. 5th IEEE Symp. FPGA-Based Custom Computing Machines (FCCM'97)*, pages 24–33, Napa Valley, CA, USA, Apr. 16–18, 1997.

[11] J. Heikkinen, J. Sertamo, T. Rautiainen, and J. Takala. Design of transport triggered architecture processor for discrete cosine transform. In *Proc. 15th Annu. Int. ASIC/SOC Conf.*, pages 87–91, Rochester, NY, USA, Sept. 25–28, 2002.

[12] P. Hämäläinen, M. Hännikäinen, T. D. Hämäläinen, and J. Saarinen. Hardware implementation of the Improved WEP and RC4 encryption algorithms for wireless terminals. In *Proc. EUSIPCO 2000*, volume 4, pages 2289–2292, Tampere, Finland, Sept. 5–8, 2000.

[13] P. Hämäläinen, M. Hännikäinen, M. Niemi, T. D. Hämäläinen, and J. Saarinen. Implementation of link security for wireless local area networks. In *Proc. IEEE Int. Conf. on Telecommunications (ICT 2001)*, volume 1, pages 299–305, Bucharest, Romania, June 4–8, 2001.

[14] Wireless LAN medium access control (MAC) and physical layer (PHY) specifications: Medium acccess control (MAC) security enhancements. IEEE Std. 802.11i, 2004.

[15] International technology roadmap for semiconductors: System drivers. Technical report, Sematech, 2003.

[16] M. Lewis and S. Simmons. A VLSI implementation of a cryptographic processor. In *Proc. Canadian Conf. Electrical and Computer Engineering (CCECE 2003)*, pages 821–826, Montreal, Canada, May 4–7, 2003.

[17] S. Ljungkvist. RC4 assembly source. www.8052.com/codelib/ArcFour.asm, 2003.

[18] MOVE project website. www.tkt.cs.tut.fi/∼move, 2005.

[19] D. Olivia, R. Buchty, and N. Heintze. AES and the Cryptonite crypto processor. In *Proc. CASES 2003*, pages 198–209, San Jose, CA, USA, Oct. 30–Nov. 1, 2003.

[20] S. Ravi, A. Raghunathan, N. Potlapally, and M. Sankaradass. System design methodologies for a wireless security processing platform. In *Proc. 39th Design Automation Conf.*, pages 777–782, New Orleans, LA, USA, June 10–14, 2002.

[21] A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A compact Rijndael hardware architecture with S-box optimization AES design. volume 2248 of *Lecture Notes in Computer Science*, pages 239–254, Germany, 2001. Springer-Verlag.

[22] B. Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. Wiley & Sons, USA, 2 edition, 1996.

[23] B. Schneier and D. Whiting. Fast software encryption: Designing encryption for optimal software speed on the Intel Pentium processor. In *Proc. Fast Software Encryption Workshop 1997*, pages 242–259, Haifa, Israel, Jan. 20–22, 1997.

[24] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho. MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE Trans. Computers*, 45(5):465–481, 2000.

[25] R. R. Taylor and S. C. Goldstein. A high-performance flexible architecture for cryptography. volume 1717 of *Lecture Notes in Computer Science*, pages 231–245. Springer, 1999.

[26] L. Wu, C. Weaver, and T. Austin. CryptoManiac: A fast flexible architecture for secure communication. In *Proc. 28th Annu. Int. Symp. Computer Architecture (ISCA 2001)*, pages 110–119, Göteborg, Sweden, June 30–July 4, 2001.

PUBLICATION 6

P. Groen, P. Hämäläinen, B. Juurlink, and T. D. Hämäläinen, "Accelerating the Secure Remote Password Protocol Using Reconfigurable Hardware," in *Proceedings of the 2004 ACM International Conference on Computing Frontiers (CF'04)*, Ischia, Italy, Apr. 14–16, 2004, pp. 471–480.

DOI: http://doi.acm.org/10.1145/977091.977157

Accelerating the Secure Remote Password Protocol Using Reconfigurable Hardware

Peter Groen
Computer Engineering Laboratory
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
P.O. Box 5031, 2600 GA Delft, Netherlands
P.T.Groen@ewi.tudelft.nl

Panu Hämäläinen
Institute of Digital and Computer Systems
Tampere University of Technology
P.O. Box 553, 33101 Tampere, Finland
panu.hamalainen@tut.fi

Ben Juurlink
Computer Engineering Laboratory
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
P.O. Box 5031, 2600 GA Delft, Netherlands
B.H.H.Juurlink@ewi.tudelft.nl

Timo Hämäläinen
Institute of Digital and Computer Systems
Tampere University of Technology
P.O. Box 553, 33101 Tampere, Finland
timo.d.hamalainen@tut.fi

ABSTRACT

The Secure Remote Password (SRP) protocol is an authentication and key-exchange protocol suitable for secure password verification and session key generation over insecure communication channels. The modular exponentiations involved, however, are very time-consuming, causing slow log-on procedures. This work presents the design of a hardware accelerator that performs modular exponentiation of very wide integers. The experimental platform is TUTWLAN, a Wireless Local Area Network (WLAN) being developed at Tampere University of Technology. It runs on the Altera Excalibur development board that contains a microprocessor and a chip with programmable hardware. The results show that a full modular exponentiation with 1023-bit inputs can be performed in less than 40 ms using less than 10,000 logic elements, each consisting of a 4-input lookup table and a register. By using the implemented hardware accelerator in the authentication protocol, the execution time is reduced by a factor of 4. In addition, proposals to improve the implemented modular exponentiation architecture are presented. An additional factor of 5 improvement (totaling a factor of 20) can be achieved by implementing the fastest design.

Categories and Subject Descriptors

B.2.4 [**Arithmetic and Logic Structures**]: High-Speed Arithmetic—*Algorithms*; C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

CF'04, April 14–16, 2004, Ischia, Italy.

General Terms

Design, security

Keywords

Secure Remote Password protocol, modular exponentiation, hardware acceleration, reconfigurable hardware, WLAN, authentication.

1. INTRODUCTION

In recent years Wireless Local Area Networks (WLANs) have gained popularity. They can be used to connect computers when building a wired network is impossible or inconvenient, e.g., when users are moving or when the network is set up only temporarily. Because data is transferred through the ether, security is needed to prevent data from being intercepted by unauthorized users as well as to authorize users and computers to the network. One way to obtain security is by using cryptographic operations. These operations, however, often require time-consuming computations that slow down network speed and increase log-on time. Furthermore, support for several algorithms and possibilities to update the implementation may be required. Trading chip area and power-consumption for execution speed depending on the application requirements may also be profitable. High speed with flexibility can be obtained by using reconfigurable hardware for the time-consuming operations and a microprocessor for the remaining parts of the security implementation.

The objective of this work is to accelerate the authentication protocol of a WLAN, called TUTWLAN, developed at Tampere University of Technology. The protocol employed is the Secure Remote Password (SRP) authentication protocol [19]. It makes extensive use of hash and modular exponentiation functions. Both of them are often used in security protocols and require a large amount of arithmetic operations. The aim is to find or design appropriate hard-

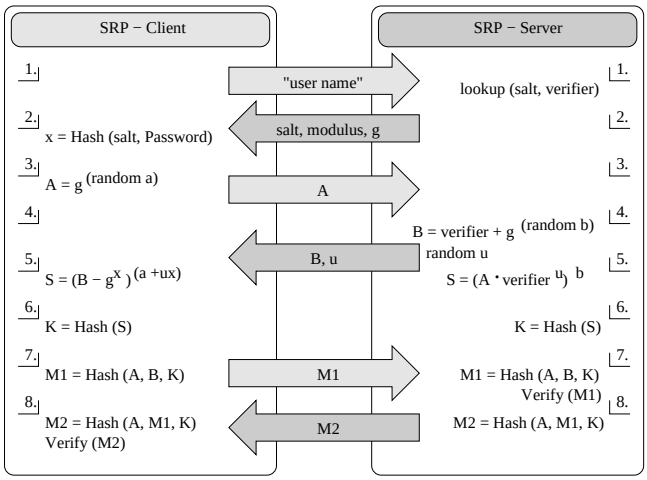


Figure 1: Steps in the SRP protocol.

ware accelerators that can be called from software routines in the SRP protocol.

This paper is structured as follows. Section 2 describes the SRP protocol. Section 3 explains the time-consuming functions in the protocol. The implemented hardware accelerator is described in Section 4. Section 5 presents the results and Section 6 discusses how execution time can be further reduced. A summary of related work is given in Section 7. Conclusions can be found in Section 8 and possibilities for future work are given in Section 9.

2. THE SRP PROTOCOL

The Secure Remote Password (SRP) protocol [19] is an authentication and key-exchange protocol suitable for secure password verification and session key generation over an insecure communication channel. In this section the protocol is described briefly.

The SRP protocol employs a method to exchange session keys called Asymmetric Key Exchange (AKE). In this method verifiers are stored instead of the passwords. The verifier is computed from the password using a one-way mathematical function [15]. The password and verifier correspond to private and public keys with the difference that the verifier is kept secret on the server instead of being publicly known. A stolen verifier is not sufficient to log-on because the password is still needed. To establish a secure connection it is sufficient when one side (server) stores the verifier(s) while the other (client) computes the verifier from a user-given password.

The initial setup of SRP goes as follows. First, the user enters a password. Then a verifier is computed from the password and a randomly generated password salt. Next the user name, salt and verifier are stored in the database on the server. Now the client is ready to authenticate to the server. The steps performed by the SRP protocol are depicted in Figure 1. All computations are performed modulo N. The modulus N is a large prime number with a length of hundreds of bits. SRP parameter sizes as used in this work are listed in Table 1 [8]. The authentication is successful if M_1 computed in Step 7 and M_2 in Step 8 are identical on the client as well as the server side.

In some of the steps, a one-way hash function is employed.

Table 1: Used SRP parameter sizes in bits.

Parameter	N = 511	N = 1023	N = 2047
user name, u	32	32	32
password (x)	64 (160)	64 (160)	64 (160)
salt s	80	80	80
generator g	8	8	8
verifier v	512	1024	2048
random a, b	256	256	256
public A, B	512	1024	2048
key K, M1, M2	320	320	320

The Secure Hash Algorithm (SHA-1) [16] is used for this. It produces a 160-bit hash value from a string of variable length. In addition, the protocol uses an interleaved hash that computes 2 hash functions and yields 320 bits. The first hash function is applied to the even bytes and the second to the odd bytes of the input. The hash outputs are byte-wise interleaved to get the final result.

The SRP protocol can be implemented in C using a library that allows large variables to be processed and contains optimized C/C++/Assembly routines. The MIRACL library developed by Shamus Software Ltd [17] is very suitable since it supports the ARM9 processor that is used in the TUTWLAN platform. The software performance of the SRP protocol has been evaluated for the ARM9 in [8].

3. FAST MODULAR EXPONENTIATION

In the SRP protocol, modular exponentiations of large integers are performed. This function requires many operations and consumes a large fraction of the total execution time of software implementations of the SRP protocol [8]. This work, therefore, focuses on designing appropriate hardware accelerators for modular exponentiations.

A straightforward implementation requires E multiplications to compute X^E. A more efficient algorithm is the Square and Multiply algorithm [9]. Algorithm 1 describes the right-to-left Square and Multiply algorithm: bits are processed from least significant (rightmost) to most significant (leftmost). The right-to-left method allows for parallel squaring and multiplying.

ALGORITHM 1. *Right-to-left Square and Multiply [9].*
Input parameters:
X: the number to be exponentiated.
E: the exponent, consisting of n bits. $E = \sum_{i=0}^{n-1} e_i 2^i$, *where* $e_i \in \{0, 1\}$.
N: the modulus.
Result:
$P = X^E \bmod N$

1. $Z = X,\ P = 1$
2. *FOR* $i = 0$ *to* $n - 1$ *DO*
3. *IF* $e_i = 1$ *THEN* $P = Z \cdot P \bmod N$ (*multiply*)
4. $Z = Z^2 \bmod N$ (*square*)
5. *END FOR*

It can be seen that the Square and Multiply algorithm requires a division after every multiplication. In Montgomery's algorithm [11] these expensive divisions are avoided by transforming the operands to N-residues. The computations are performed with these residues and afterwards the result is transformed back to normal representation. An N-residue of

x is defined as $x \cdot R \bmod N$ where $R = 2^n$ and n is the number of bits in modulus N, such that $R > N$. The key idea is to perform multiplications modulo R instead of modulo N, which replaces the expensive division after each multiplication by a division by a power of 2 (i.e., a rightshift). Montgomery has defined Algorithm 2 to compute $T \cdot R^{-1} \bmod N$ from T for $0 \leq T < RN$, where R^{-1} is the inverse modulus.

ALGORITHM 2. *Montgomery reduction [11].*
Input parameters:
N: the modulus, $N' = -N^{-1}$
R: the radix, such that $R > N$ *and* R *co-prime to* N
T: the number to be reduced, where $0 \leq T < RN$
Result:
$t = (T \cdot R^{-1}) \bmod N$
function $REDC(T)$
1. $m = (T \bmod R)N' \bmod R$
2. $t = (T + m \cdot N)/R$
3. *IF* $t \geq N$ *THEN* $t = t - N$

The radix R and the modulus N need to be co-prime (no common divider greater than one) to guarantee the existence of an inverse of N. Since R is a power of two, the algorithm works for any odd modulus, which is the case in SRP. Function $REDC(T)$ forms the basic framework to perform multiplication with N-residues. The output of this algorithm is $(T \cdot R^{-1}) \bmod N$. A product of residues of x and y is computed as:

$$z = REDC((xR \bmod N)(yR \bmod N)) \quad (1)$$
$$= (xy)R^2R^{-1} \bmod N \quad (2)$$
$$= xyR \bmod N \quad (3)$$

Transformation to and from Montgomery-residues can be done using function *REDC*. The initial values x and y have to be multiplied with $R^2 \bmod N$. Transformation from residues to normal representation can be done by multiplying with 1. Now only $R^2 \bmod N$ has to be computed using slow non-Montgomery multiplications. This has to be performed only once for a given modulus.

A multiplication is composed out of the addition of partial products. Each partial product is the result of multiplying a part of the multiplier with the full multiplicand. If l is the number of bits of the multiplier, a full multiplication would require l multiplications of 1 bit of the multiplier with the full multiplicand. If k bits of the multiplier are processed at once (radix-2^k), a full multiplication requires l/k multiplications. Algorithm 2 can be transformed to Algorithm 3 to realize a radix-2^k multiplier that is called l/k times [6, 5].

ALGORITHM 3. *Radix-*2^k *Montgomery modular multiplication [5].*
Notations:
l: the number of bits of the multiplier
$m = l/k$
$N' = -N^{-1} \bmod 2^k$
Input parameters:
A: the multiplier. $A = \sum_{i=0}^{m-1} a_i(2^k)^i$ *where* $a_i \in \{0, 1, \ldots, 2^k - 1\}$
B: the multiplicand
N: the modulus
Result:
$S = ABR^{-1} \bmod N$, *where* $R = 2^l$ *and R,N co-prime*
1. $S = 0$
2. *FOR* $i = 0$ *to* $m - 1$ *DO*
3. $q = (((S + a_iB) \bmod 2^k)N') \bmod 2^k$
4. $S = (S + qN + a_iB)/2^k$
5. *END FOR*
6. *IF* $S \geq N$ *THEN* $S = S - N$

The result of Algorithm 3 is $ABR^{-1} \bmod N$. The algorithm is suitable for implementation on reconfigurable hardware because large multipliers and divisions are avoided. The Montgomery algorithm allows a pipelined implementation because the next computation does not have to wait for the most significant bits in the determination of q_i. This makes Montgomery suitable for processing multiplications with very large moduli.

Algorithm 3 can be further improved by replacing the *IF* statement in Step 6 by two extra iterations. Two extra division will make the result always smaller than 2 times the modulus. The addition in Step 3 is avoided by multiplying B by 2^k. The product $a_iB \bmod 2^k$ will thus be 0 for all B. In Step 4, B can be taken out of the division. The modulus N must be odd to be co-prime to the Montgomery radix R, the inverse of N in Algorithm 3 for the radix-2 algorithm becomes 1 since $N' = -N^{-1} \bmod 2 = 1$ for every odd N. No inverse needs to be computed. Algorithm 4 shows the algorithm that is implemented in the hardware [10, 18].

ALGORITHM 4. *Radix-2 Montgomery modular multiplication [4].*
1. $S = 0$
2. *FOR* $i = 0$ *to* $m + 2$ *DO*
3. $q = S \bmod 2$
4. $S = (S + qN)/2 + a_iB$
5. *END FOR*

The fourth step contains the arithmetic operations that can be implemented in an array of processing units in reconfigurable hardware. Two multipliers and two adders are needed in each unit to implement the multiplications and additions.

4. IMPLEMENTATION OF RADIX-2 MODULAR EXPONENTIATION

The hardware acceleration for the SRP protocol has been implemented on the Altera Excalibur device. The chip consists of a 32 bit RISC ARM922 processor operating at up to 200 MHz combined with a programmable logic device (PLD). Processor and PLD are connected to each other by an embedded stripe. Communication between stripe and PLD goes through two dual ported RAM blocks (one for the smallest Excalibur device) or through AHB master/slave ports. The PLD can also interrupt the ARM directly through 6 interrupt lines.

The structure of the PLD in the Excalibur is equal to Altera's APEX20K devices at an internal voltage of 1.8. The APEX devices are based on a large number of Logic Elements (LEs). Each LE contains a 4-input look-up table (LUT) and one register which means that any boolean function of up to 4 bits can be implemented in one LE. It can also be configured to operate in a special arithmetic mode which has only two inputs but allows for fast propagation of carries between neighboring LEs. A register is available directly after each LUT. The output of the LUT can be routed

Table 2: Excalibur devices.

Device	EPXA1	EPXA4	EPXA10
Single-port RAM	32kbytes	128kbytes	256kbytes
Dual-port RAM	16kbytes	64bytes	128bytes
Typical gates	100.000	400.000	1.000.000
Logic Elements	4.160	16.640	38.400
ESB/RAM blocks	26	104	160
Max system gates	263.200	1.052.000	1.772.000
Max user pins	246	488	711

through the register, around the register or both. In addition, the PLD has ESB/RAM blocks which can store up to 2048 bits each. Refer to Table 2 and [1] for specifications of the Excalibur Devices.

4.1 Exponentiation Unit

As depicted in Figure 2, the computations are performed in an array of processing units. The multiplicand B is loaded into the units from a central point, all other signals are moved between neighboring units. In Algorithm 4, all intermediate values are at most two times as large as the modulus. This means that, internally, everything should be 1025 bits to allow a 1024-bit modulus. To simplify the design, the maximum modulus is set to one bit less: $2^n - 1$. Moduli will have lengths of for example 511, 1023 or 2047 bits. This implies that the length of all intermediate values are a power of two which simplifies the design.

There are five RAM blocks in Figure 2. The two RAM blocks in the lower-right corner store the modulus N and the Montgomery radix R^2 mod N. The Montgomery radix is used to transform variables to residues. The top-center RAM block stores the exponent E and the two blocks immediately below store the results of each multiplication and squaring step. The exponentiation unit is controlled by two state machines. The machine labelled 'Exp-Control' contains a counter to generate the address to select the appropriate bit of the exponent E. The machine labelled 'Mult-Control' controls the systolic array.

The procedure to compute X^E mod N using Montgomery is as follows: X is transformed to Montgomery-residue. This takes one Montgomery multiplication. Then the Square and Multiply algorithm is applied. The final output P is transformed back to normal representation which takes another Montgomery multiplication. In the radix-2 algorithm, only one bit of the multiplier is processed per multiplication thus one iteration of the Multiply and Square algorithm takes $(l + 3) \times 2$ cycles. All iterations together take $n \times (l + 3) \times 2$ cycles. A full exponentiation (exponent and modulus have the same length), including transformations, takes $(n + 2) \times (n + 3) \times 2$ cycles, where n is the bit length of the modulus.

4.2 Processing Units

The number of bits per unit is an important factor of the maximum frequency (fmax) since it directly affects the routing complexity and longest path. Eight registers are needed per unit to store the signals that are transported between the units. Processing more bits of the multiplicand per unit saves registers but increases the complexity and longest path. In this design, each unit processes eight bits of the multiplicand.

Figure 3 shows the structure of one processing unit. It computes $S_{i+1} = (S_i + q_i N)/2 + a_i B$. *Adder_1* computes $(S_i + q_i N)$ and *Adder_2* adds $a_i B$. The block labeled *Box* computes the 9th bit of the addition of S and $q_i N$. This bit is needed as extra input for the second adder because of the rightshift between the adders. For the highest unit, the carry-out bit of *Adder_2* comes back as $S(i)[0]$. *Adder_2* needs to be one bit larger and the box will contain a 2-bit adder instead of 1-bit. A unit has to perform the following steps to perform a Montgomery multiplication.

1. The modulus N (8 bits per unit) is loaded via B in Multiplexer+Reg_1. Multiplier_1 is set on multiplication by 1 to let the modulus pass through unchanged while the registers in Reg_1 are reading the data.

2. The multiplicand B (8 bit per unit) of Algorithm 4 is loaded via B in Multiplexer+Reg_1. This operand is the same for both squaring and multiplication step of the exponentiation Algorithm 1.

3. The multipliers a_i are moved through the units at one bit per clock (radix-2).

4. The result of the first multiplication (squaring) is fed back to Multiplexer+Reg_2 for the next iteration.

5. After the completion of the multiplications, both results are moved to Multiplexer+Reg_3. The result of the first multiplication (squaring) is sent to the previous unit and also to Multiplexer+Reg_1. The result of the second multiplication is moved to the previous unit only.

6. Step 2 to 5 are repeated to compute an exponentiation. In Step 2, S is loaded instead of B.

All signals are registered before they leave a unit. Thus a signal needs one clock cycle to move to the higher neighbor. The higher neighbor uses these signals and shifts its lowest bit back to the lower unit afterwards. This allows a two-cycle operating mode with interleaved squaring and multiplying. The unit could be transformed to a single-cycle version by not registering the bit of S that is transported to the lower unit. In a single-cycle version, the square and multiply steps in the exponentiation algorithm are performed in series. No long carry chains through the whole array are created as long as the adder is at least twice as large as the number of bits of S that are moved to the lower unit. For an 8-bit wide unit, up to 4 bits or radix-16 can be computed in the one-cycle mode.

A unit needs 70 LEs when implemented in the Altera APEX PLD. The registers after the adder are combined in one LE. The registers that store the modulus N are placed after the 1×8 Multiplier so that they fit in the same LE.

4.3 Communication with the ARM Processor

Communication between the PLD and the ARM processor can be done through DPRAM or AHB slave/master ports. The DPRAM ports allow for quick access to large amount of data. Since the exponentiation function needs large numbers before computation can start and the result has to be written back afterwards, the DPRAM ports are most suited. Both DPRAM ports are configured for 32-bit data size so reading of an exponent and base of 1023 bits takes 32 clock cycles.

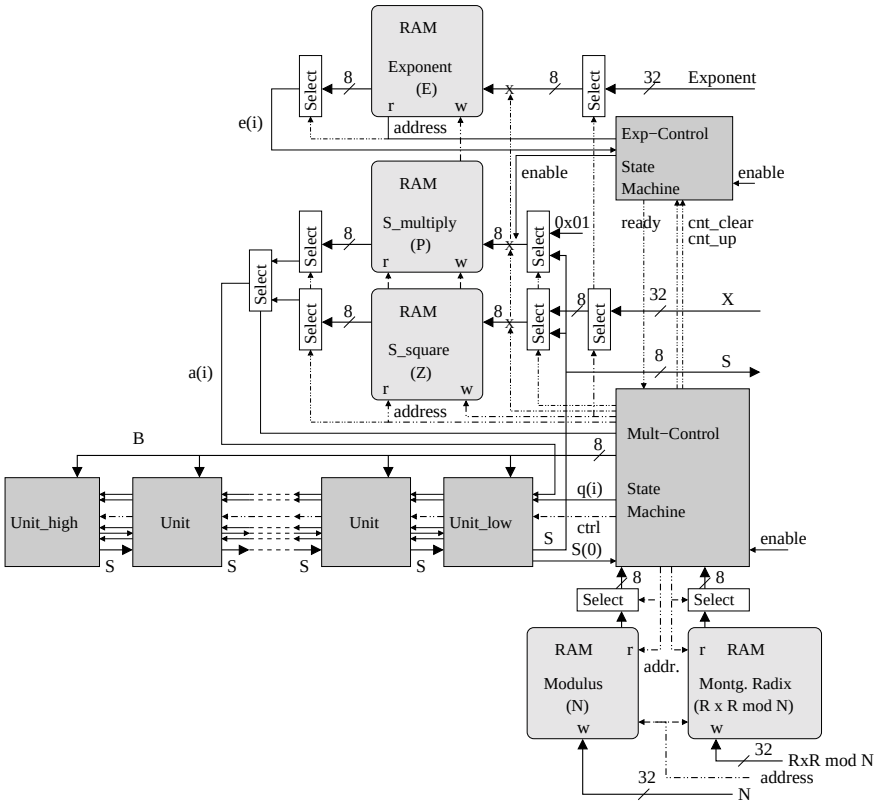


Figure 2: Systolic radix-2 exponentiation in PLD.

About 400 LEs are needed for the communication with the ARM, mainly caused by registers and control that is needed to move the data to and from the exponentiation unit. The hardware modular exponentiation can be enabled/disabled by sending a signal to an AHB-slave in the PLD. This allows other parts of the TUTWLAN to use DPRAM when no exponentiations are computed.

Four parameters are needed to perform modular exponentiation. The modulus N, the Montgomery radix R^2, a base, and an exponent. The read and write process is managed by a state machine (PLD_control in Figure 4.3). The base and the exponent are stored in the ESB/RAM blocks, and the addresses are generated by the exponentiation control unit. Reading of N and R^2 is controlled by a separate control unit to generate the address. This is because they are stored in ESB/RAM blocks before the state machine in the modular exponentiation hardware is enabled. They have to be read only when a new modulus is used.

4.4 Software Routines

The hardware exponentiation is controlled by the software running on the ARM. The software consists of three routines:

1. Read N, compute R^2 mod N and store them in DPRAM.

2. Read Base and Exponent, move them to DPRAM, start accelerator.

3. Read results from DPRAM.

The radius R needs to be transferred to the Montgomery domain (R^2 mod N) by conventional divisions. Although this could be performed in hardware it is performed in software, since it is likely that the modulus is not frequently changed in SRP. Changing the modulus requires computing new verifiers for each client and updating the whole server database (it is assumed that the whole database uses the same modulus).

The SRP implementation processes variables in a special large format supplied by the MIRACL library. Before moving variables to DPRAM, they are transformed to strings, then to a series of 32-bit integers and stored in DPRAM. The results are transformed from integers to strings and then back to the MIRACL format. The UART port is configured to output to a terminal on a regular PC. Data can be input and output through this terminal.

5. EXPERIMENTAL RESULTS

During the experiments, the ARM processor was configured to run at a frequency of 50 MHz and the PLD at 33 MHz. The Quartus software was used together with the ARM Developer Suite [1][2] to compile and simulate the designs. The hardware part of the design is implemented in VHDL and Altera standard components from the Quartus II software. Leonardo Spectrum was used for synthesis. The Quartus software produces a *hex* file that was downloaded into the flash memory of the Excalibur development platform. The design has been tested by comparing the results of the exponentiation in hardware with the results in

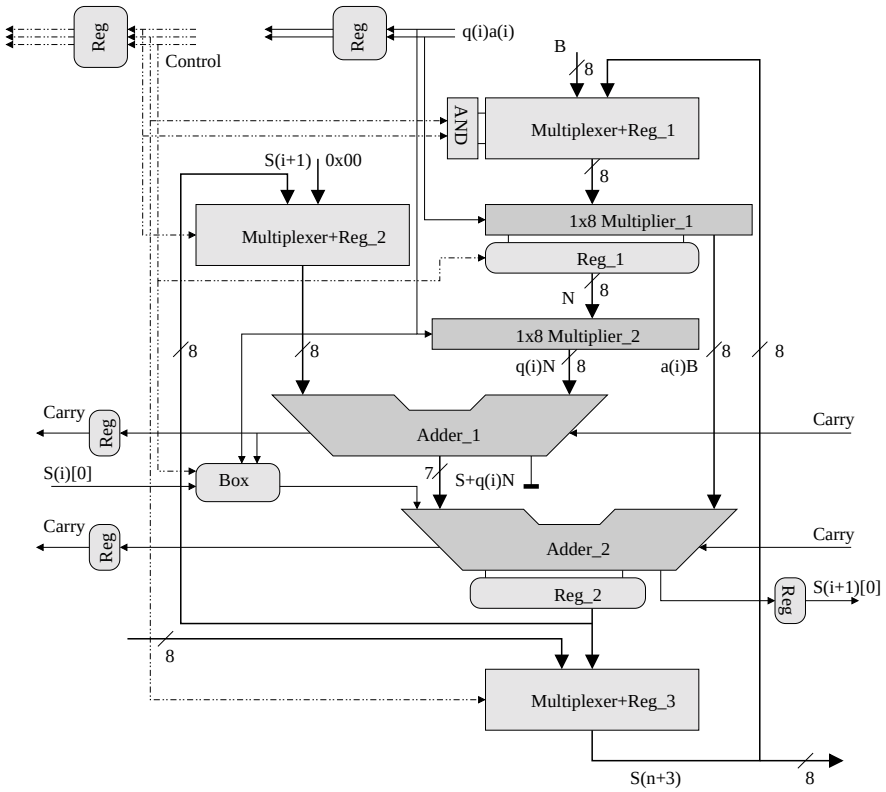


Figure 3: 8-bit radix-2 processing unit.

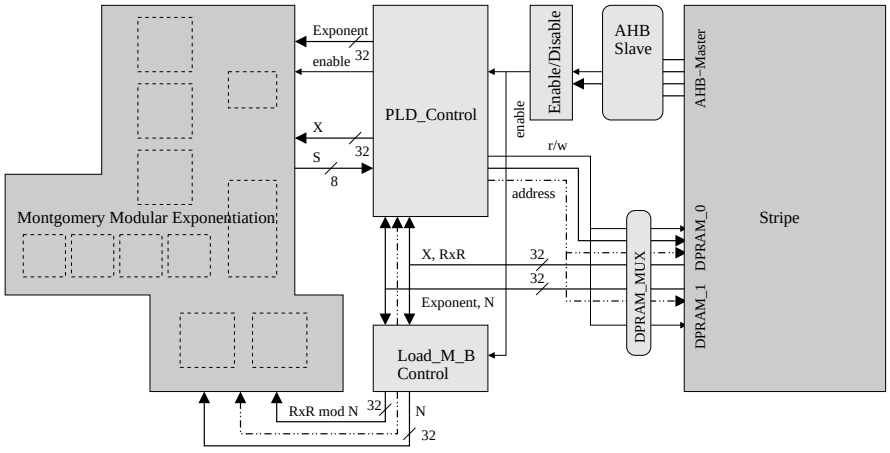


Figure 4: Communication between the ARM and the Exponentiation Unit.

software. The compiler returns values for the maximum frequency and the number of LEs needed in its compilation report. These values are depicted in Table 3.

Table 3: Max. frequency and LE usage radix-2.

Modulus	LEs	RAM-bits	fmax (MHz)
511 bit	5149	3584	65.37
1023 bit	9644	5120	65.11
2047 bit	18186	10240	58.01

The clock cycles in the PLD have been measured with a counter in the PLD that starts when the software enables the hardware exponentiation and stops immediately after the results have been written to the DPRAM. Table 4 shows how many clock cycles have been measured by the counter and how many full exponentiations are possible per second.

Table 4: Cycles for a full radix-2 exponentiation.

Modulus	clock cycles/exp.	ms./exp.	exp./sec.
511 bit	530,704	16	61.24
1023 bit	2,109,968	32	30.86
2047 bit	8,414,224	73	13.79

The operations g^x, g^a, g^b and the exponentiation for the session key S are computed in hardware, where the length of the exponent is 256 bits. The server computes v^u in Step 5 of the SRP protocol, where u is 32 bits width, in software. The execution time of this step is significant but performing it in hardware would be even slower because the hardware always runs through the maximum length of the exponent. Also in g^x the exponent is smaller (160 bits). The efficiency of the hardware accelerator would be a little higher when the hardware is adapted to stop after all bits of the exponent have been processed.

Figure 5 compares the performance of our hardware accelerated implementation of the SRP protocol to that of an optimized software implementation [8]. In order to obtain these execution times, the frequency of the ARM9 was set to 200 MHz (its maximum frequency) and the frequency of the PLD was set to 50 MHz, even though its maximum frequency is higher (cf. Table 3). It can be seen that the hardware accelerated implementation is much faster than the pure software implementation. For example, when the modulus is 1023 bits wide, our implementation in which exponention is performed in hardware is about four times faster than the software implementation. Furthermore, the speed-up increases with the size of the modulus. Since exponentiation consumes most of the execution time of the SRP protocol, a speed-up of this function will result in a similar speed-up of the whole protocol.

6. IMPROVEMENTS

The radix-2 array architecture performs multiplications at one bit of the multiplier per cycle. The design needs at minimum $2 \times n \times n$ clock cycles to complete a full modular exponentiation with Montgomery residues, where both X and E in X^E have a bit-length of n. The execution time can be further improved in the following ways:

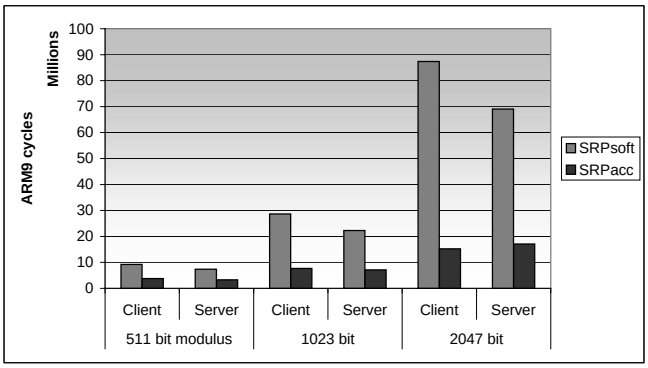


Figure 5: Performance comparison of the hardware accelerated implementation of the SRP protocol (SRPacc) and an optimized software implementation (SRPsoft).

1. Compute more than 1 bit per Montgomery multiplication (higher radix).
2. Compute more than 1 bit per iteration of the exponentiation (M-ary).
3. Compute multiplications and squaring in parallel.

6.1 Higher Radix

In higher radix, several bits of A are inserted into the units instead of only one. Contrary to the radix-2 design, the inverse modulus is needed as in Algorithm 3. The inverse modulus N' can be easily computed [14]. The time to compute the inverse is negligible since the inverse is small compared to the modulus. The higher radix Montgomery multiplication algorithm can be derived from Algorithm 3 in the same way as was done for the radix-2 design [5]. Radix 2^k takes $2 \times n \times n/k$ cycles.

Most of the implemented radix-2 design can also be used for a higher-radix version. To omit the need for multipliers, all possible outputs of the products $a_i 2^k S$ and $q_i 2^k N$ in Algorithm 3 have to be pre-computed and are stored in ESB/RAM. For a 1023 bit exponentiation, 2048 bits have to be read from RAM per clock cycle, which requires 128 ESB blocks. A block stores a 16 bit part of either $a_i 2^k S$ or $q_i 2^k N$. The outputs of $a_i 2^k S$ have to be pre-computed for each iteration of Algorithm 1. One 16-bit multiplexor and a 16-bit adder are needed to perform these pre-computations. Addressing for the RAM blocks takes 2 LEs per bit of a_i and q_i per processing unit. Together, approximately 50 extra LEs are needed per 16 bits of the modulus. For a modulus of 1023 bits and radix-16 this will be about 3200 extra LEs. The method becomes less efficient for radices larger than 16 because of the large amount of multiplications that need to be pre-computed.

6.2 M-ary

The M-ary method allows processing of several bits of the exponent per multiplication in the Square and Multiply Algorithm [9]. Processing 2 instead of 1 bit per multiply of the exponent reduces the number of clock cycles with almost 25%. Only one multiply operation is needed for every two square operations. The M-ary method requires parameter P in Algorithm 1 to be stored several times. If, for example, 2

bits of the exponent are processed per multiplication, three Ps have to be stored. The first P is modified if the bits are '01', the second if the bits are '10' and the third if the bits are '11'. In the end $P2 = P2 \cdot P3$ and $P1 = P1 \cdot P2$ and $P = P3 \cdot P2 \cdot P1$ are computed to get the final result. The M-ary method can be used on any number of bits but at a certain point so many post multiplications are needed that the benefit disappears. The clock cycles can be roughly computed as $n+n/(number\_of\_bits)$ where n is the bit-length of the modulus.

Implementation of the M-ary method requires an extra ESB/RAM block for each P and LEs to build multiplexors for selection of the appropriate P. Since at most 8 bits are selected at each clock cycle, the cost of the multiplexor is small: a 2-ary with 3 multiples requires 16 LES, a 4-ary with 15 multiples requires 80 LEs to select the appropriate P. Furthermore, the control unit may increases in size, especially due to at least two extra states to cover the extra multiplications.

6.3 Parallel Arrays

Two parallel arrays for squaring and multiplying generally require twice as much area at a speed gain of a factor 2. Parallel arrays require the processing units to be single cycle. This is possible up to radix-16 when 4 out of 8 result bits are connected to the lower neighboring unit without being registered. Another modification is that instead of shifting between the adders in a processing unit, the entire inputs are shifted. An extra benefit of parallel squaring and multiplying is that squaring can be done in higher radix, while the M-ary method can reduce the number of multiplications simultaneously.

6.4 Execution Time Estimates

Although the designs described above have not been implemented due to project time limitations, the number of clock cycles required by each method can be fairly accurately estimated by combining the execution time formulas for M-ary, higher radix, and parallel. Figure 6 depicts execution time estimates for the following methods: radix-4, 16, and 64 (labeled 'R4', 'R16', and 'R64', respectively), radix-64 and 256 where only the odd half of the possible products is stored in RAM (labeled 'R64*' and 'R256*'), and parallel arrays with radix-2, 4, and 16 (labeled 'R2-par', 'R4-par', and 'R16-par', respectively). Pre- and post-computations are included in the execution times. Since the length of the critical path within the units does not increase significantly for higher radix and not at all for M-ary, it is assumed that a clock frequency of 50 MHz is feasible, which is 25% smaller than the maximum frequency of the radix-2 design. The results show that radices larger than 16 do not provide a significant performance improvement. The results also show that the improvement of the M-ary method decreases for higher radices due to the fact that a larger number of cycles is needed for pre/post-multiplications.

All designs fit on the EPXA10, the largest Excalibur device. Expansion to 2047-bit data-width is only possible for radix-2, 1-ary, 2-ary and 4-ary designs and will take approximately 4 times as many cycles as the radix-2. The fastest design is the radix-16, 4-ary parallel which is estimated to require less than 290.000 clock cycles at an area consumption of about 25,000 LEs and nearly all ESB/RAM blocks in the PLD.

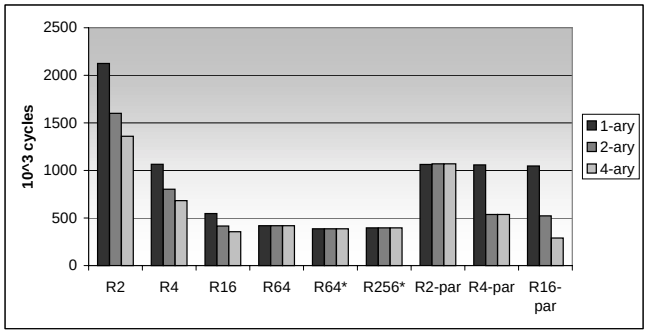


Figure 6: Estimated execution times of faster exponentiation methods. The size of the modulus is 1023 bits.

7. RELATED WORK

For the implementation of modular exponentiation in hardware, several approaches are available. Existing designs of modular exponentiations for Field Programmable Gate Array (FPGA) platforms have been studied and are briefly described below.

There are basically two methods to compute a Montgomery modular multiplication. One is the redundant representation, where addition to a binary representation is only done for the final result to prevent very long carry propagations. Intermediate results are kept in the carry-save state. In redundant representation, carries and sums are kept separated after each iteration. In a systolic array, a multiplication is done by a large number of parallel units that each compute a couple of bits of the result. This is a line of parallel processing units, in which values (q_i and a_i in Algorithm 3) are 'pumped' through the units while carries are moved to the next unit in each clock cycle. The systolic array approach was used in this work.

There are also modular multiplication methods not based on Montgomery. Johan Groszschaedl [7] uses Barret's modular reduction but did not implement it on an FPGA.

Blum and Paar's design is one of the fastest implementation of Montgomery Modular Exponentiation on an FPGA [4, 3]. They have developed a resource efficient and a speed efficient design for Xilinx FPGA, both based on an optimized version of Algorithm 3. The speed efficient design is a radix-16 (k=4) implementation in which the 16 multiples of the modulus and the multiplicand B are precomputed, stored in RAM and addressed with q_i and a_i respectively [5]. Their implementations are based on a systolic array approach in which the multiplication and squaring are performed interleaved in the same array. Blum and Paar's radix-2 implementation runs in approximately $2n$ cycles and their radix-16 in almost $0.5n$ cycles where n is the length of both the exponent and the base. Others have modified the systolic array design for Montgomery modular exponentiation which allows their implementation to run in $1.5n$ cycles [18].

The structure of the radix-2 exponentiation design presented in this paper is close to Blum and Paar's radix-16 design. The major differences are the smaller radix, wider processing units (8 bits instead of 4 bits), different technology (Altera), the design of the communication between the processor and the accelerator, and the software design (the interface functions and the rest of the SRP protocol).

Another implementation alternative is to have separate arrays for multiplying and squaring. Naturally this will cost twice as much area to have the full exponentiation done in n cycles instead of $2n$. In [14] two arrays are used in an implementation on Xilinx FPGAs. This design needs four times as much resources as Blum's radix-16 design but is only 20% faster.

A comparison between a systolic array and an iterative array can be found in [12]. It turns out that the systolic approach requires about 1.5 times more resources for large moduli and is at least 2 times faster than the iterative approach. In [4] it is argued that a systolic array needs more resources than the redundant approach to store the signals between the units in the array, but it is easier to route and can therefore be faster than a redundant representation on FPGAs. Poldre et al. [13] also describe both approaches but did not implement them in hardware.

8. SUMMARY

A hardware exponentiation unit consisting of a radix-2 systolic array has been designed and implemented on the Altera PLD. It can be configured to perform modular exponentiations using moduli of 127, 255, 511, 1023, or 2047 bits. When the size of the modulus is 1023 bits, our implementation of the SRP protocol that employs the exponentiation unit is approximately 4 times faster than an optimized software implementation. Furthermore, it requires less than 10,000 LEs or about 1/4th of the PLD of the largest Excalibur chip. The unit can be used continuously.

Faster hardware exponentiations are possible by using a higher radix, by employing the M-ary method, and/or by using two arrays to perform squaring and multiplying in parallel. The fastest design would be one that uses radix-16, 4-ary and has 2 parallel arrays. We estimate that this design would require less than 300,000 cycles. The maximum frequency, however, will be lower than the maximum frequency of the implemented radix-2 design due to the higher complexity. Nevertheless, if it would drop to 40 MHz, it would still be 5 times faster than the radix-2 design. Compared to the pure software implementation running on the ARM9 at maximum frequency, it would be 20 times faster.

9. FUTURE WORK

We are currently implementing the faster designs on the Altera PLD. An implementation of the radix-16, 4-ary modular exponentiation employing two parallel arrays might be one of the fastest designs for modular exponentiation on reconfigurable hardware. Other possible improvements are:

- Modify the design such that it can handle modulus sizes other than powers of two minus one.
- The design can be optimized such that it can adapt to the size of the base and exponent instead of always assuming the maximum size. This would make the hardware accelerator also useful in RSA public key encryption.
- Perform hashing in reconfigurable hardware. An SHA-1 hash function is available from Altera.
- The design presented here can be compiled for other, newer (Altera) PLDs. A PLD with a large number of multipliers allows for new, faster designs for modular exponentiations. Also more ESB/RAM blocks can be useful.
- The area consumption of the loading procedure can be reduced by about 75 % when only eight bits are loaded per clock instead of two times 32 bits.
- Taking a closer look to types other than systolic arrays with 8-bit units or modular multipliers.

10. REFERENCES

[1] Altera Homepage. www.altera.com, 2003. Excalibur HW Reference Manual, APEX 20K Programmable Logic Device Family Data Sheet, Application Note 142 Using the Embedded Stripe Bridges.

[2] ARM Ltd. Homepage. www.arm.com, 2003.

[3] T. Blum. Modular Exponentiation on Reconfigurable Hardware. Master's thesis, Worchester Polytechnic Institute, Apr 1999.

[4] T. Blum and C. Paar. Montgomery Modular Exponentiation on Reconfigurable Hardware. In *14th IEEE Symposium on Computer Arithmetic*, pages 70–77. IEEE, Apr 1999.

[5] T. Blum and C. Paar. High Radix Montgomery Modular Exponentiation on Reconfigurable Hardware. In *IEEE Transactions on Computers*, volume 50(7). IEEE, Jul 2001.

[6] S.E. Eldridge and C.D. Walter. Hardware Implementation of Montgomery's Modular Multiplication Algorithm. In *IEEE Transactions on Computers*, pages 693–699. IEEE, Jul 1993. 42(6).

[7] J. Groszschaedl. High-Speed RSA Hardware based on Barret's Modular Reduction Method. In *Cryptographic Hardware and Embedded Systems (CHES 2000)*, volume 1965/2000 of *Lecture Notes in Computer Science*. Springer-Verlag Heidelberg, Aug 2000.

[8] P. Hämäläinen, M. Hännikäinen, M. Niemi, and T. Hämäläinen. Performance Evaluation of Secure Remote Password Protocol. In *IEEE International Symposium on Circuits and Systems*, volume 3, pages 29–32, Scottsdale, Arizona, USA, May 2002. IEEE.

[9] D.E. Knuth. *The art of computer programming: Seminumerical algorithms*, volume 2. Addison-Wesley, 2002.

[10] C.K. Koc, T. Acar, and B.S. Kaliski. Analyzing and Comparing Montgomery Multiplication Algorithms. In *IEEE Micro*, pages 29–33. IEEE, Jun 1996. (16)3.

[11] P.L. Montgomery. Modular Multiplication Without Trial Division. In *Mathematics of Computation*, pages 519–521, Apr 1985. 44(170).

[12] N. Nedjah and L. de Macedo Mourelle. Two Hardware Implementations for the Montgomery Modular Multiplication: Sequential versus Parallel. In *15th Symposium on Integrated Circuits and Systems Design*, Porto Alegre, Brazil, Sep 2002.

[13] Juri Poldre, Kalle Tammemäe, and Marek Mandre. Modular Exponent Realization on FPGAs. In *8th International Workshop*, volume 1482 of *Lecture Notes in Computer Science*, Tallinn, Estonia, Aug 1998. Springer-Verlag Heidelberg.

[14] T. Ristimäki and J. Nurmi. Implementation of a Fast 1024-bit RSA Encryption on Platform FPGA. In *6th*

IEEE International Workshop on Design and Diagnostics of Electronics Circuits and Systems (DDECS'03), Poznan, Poland, Apr 2003.

[15] B. Schneier. *Applied Cryptography Second Edition*. John Wiley & Sons. Inc., 1996. ISBN 0471128457.

[16] The Secure Hash Standard. Federal information processing standards (fips) - publication 180-1, National Institute of Standards and Technology (NIST), USA, 1995.

[17] Shamus Software Ltd, Dublin, Ireland. *M.I.R.A.C.L. Users Manual*, Sep 2002.

[18] C.D. Walter. An Overview of Montgomery's Multiplication Technique: How to make it Smaller and Faster. In *Cryptographic Hardware and Embedded Systems (CHES 1999)*, Lecture Notes in Computer Science, Worcester, Massachusetts, USA, Aug 1999. Springer-Verlag Heidelberg.

[19] T. Wu. The Secure Remote Password protocol. In *Internet Society Network and Distributed Systems Security Symposium (NDSS)*, pages 97–111, San Diego, California, USA, Mar 1998.

PUBLICATION 7

P. Hämäläinen, N. Liu, M. Hännikäinen, and T. D. Hämäläinen, “Acceleration of Modular Exponentiation on System-on-a-Programmable-Chip,” in *Proceedings of the 2005 IEEE International Symposium on System-on-Chip (SoC 2005)*, Tampere, Finland, Nov. 15–17, 2005, pp. 14–17.

Acceleration of Modular Exponentiation on System-on-a-Programmable-Chip

Panu Hämäläinen, Ning Liu, Marko Hännikäinen, and Timo D. Hämäläinen
Institute of Digital and Computer Systems
Tampere University of Technology
Tampere, Finland

Abstract— **Computing modular exponentiations with long integers is required in a number of security protocols. Since security procedures typically consume large amount of processing capacity in network devices, efficient implementations are needed. As a solution, this paper presents an exponentiation accelerator suited for efficient processing in security protocols using public key schemes, such as TLS and IPsec. The accelerator is implemented on a System-on-a-Programmable-Chip, partitioned into software control and hardware processing. Compared to previous radix-2 designs, significantly higher performance is achieved. The design computes a full exponentiation in (*n*+*k*)(*n*+4) clock cycles, in which *n* is the bit length of the modulus and the exponent and *k* is the number of ones in the binary representation of the exponent. In the average case, the design executes the exponentiation 25% faster than the previous hardware designs at equal clock speeds. The proposed exponentiation control and 1-cycle processing mode can also be utilized for improving higher radix designs.**

I. Introduction

One of the most utilized operations in security protocols is modular exponentiation of long integers. It is required in most public key cryptosystems for a number of purposes, including encryption, authentication, key agreement, and digital signing. For example, Transport Layer Security (TLS) and Internet Protocol Security (IPsec) require the exponentiations in their widely utilized RSA and Diffie-Hellman schemes. Authentication methods using large exponentiations are also extending to wireless technologies for improving network access control.

Large exponentiations are recognized as expensive operations. For example, in [8] it is measured that they take 90% of the processing time in the TLS handshake. Security procedures are also generally among the tasks requiring most processing capacity in network devices. Especially, in wireless devices overall processing limitations and energy consumption can be significantly decreased with efficient implementations. On the other hand, e.g., virtual private networks require high-speed implementations in busy corporate firewalls. With an efficient design, compromising the security level and degenerating response times due to limited processing resources is avoided. As a solution, this paper presents an accelerator for large modular exponentiations.

Altera Excalibur System-on-a-Programmable-Chip (SoPC) [1], consisting of programmable logic integrated with an ARM processor, is used as the implementation platform for the exponentiation accelerator. The exponentiation processing is implemented in hardware and the control in software. In [5] it is estimated that with a dedicated hardware the cycle count of a full-width modular exponentiation can be reduced by the factor of several tenfolds from software, which is also proven with the results of this work. As the expensive computations are completely performed by the hardware, the software part of the accelerator can be run even in a microcontroller with very limited capabilities without decreasing the performance.

The paper continues the work of [4], in which an exponentiation hardware was designed for accelerating Secure Remote Password protocol (SRP) [7]. Similarly to public key cryptosystems, most of the processing time in SRP is used for the exponentiations [5]. SRP has been proposed to be used in the TLS handshake as well as in the Wireless Local Area Network (WLAN) authentication. In our previous work the execution time of SRP using 256-bit exponents was reduced to $1/4$ compared to ARM software [4]. With longer exponents the performance increase is even higher. Compared to our previous work, improvements in terms of performance as well as design scalability and reusability are achieved in this work.

II. Modular Multiplication and Exponentiation

Without applying a proper algorithm, modular exponentiations of large integers can consume substantially large amount of time and space. Generally, a square-and-multiply method with a reduction algorithm is utilized. The power is calculated as a composition of squarings, multiplications, and reductions, e.g., $a^5 \ mod \ N = ((a^2 \ mod \ N)^2 \ mod \ N)a \ mod \ N$.

Each squaring and multiplication is performed with a modular multiplication algorithm. A widely used algorithm is Montgomery Multiplication (MM) [6]. It is particularly suitable for hardware since it only requires additions and shifts. Even though pre-processing and post-processing are needed, MM is advantageous when the multiplication is performed several times with the same modulus – as in exponentiation.

Algorithm 1 presents the radix-2 MM algorithm utilized in this work [2]. The radix refers to that one bit of the multiplier is processed at a time. Higher radices require less clock cycles but they are also more complex and consume more hardware resources [3]. However, the accelerator is designed considering its extension to higher radices in the future. The extra term r^{-1} is eliminated by first performing MM on the inputs (α, β) with $r^2 \ mod \ N$ and then on the result with 1.

The MM algorithm is utilized in the square-and-multiply exponentiation presented as Algorithm 2 [2]. In this work the exponent is processed one bit at a time. Similarly to higher

0-7803-9294-9/05/$20.00

Algorithm 1 Montgomery multiplication (radix-2)

Inputs: α, β, N; Output: $\delta_{n+3} = \alpha\beta r^{-1} \ mod \ N$
$\alpha = \sum_{i=0}^{n+2} a_i 2^i, a_i \in \{0,1\}, a_{n+1} = a_{n+2} = 0$
$\beta = \sum_{i=0}^{n} b_i 2^i, b_i \in \{0,1\}$
$N = \sum_{i=0}^{n-1} m_i 2^i, m_i \in \{0,1\}$
$N \ mod \ 2 = 1$; $A, B < 2N$; $r = 2^{n+2}$

1: $\delta_0 = 0$
2: **for** $i = 0$ to $n+2$ **do**
3: $\quad q_i = \delta_i \ mod \ 2$
4: $\quad \delta_{i+1} = (\delta_i + q_i N)/2 + a_i \beta$
5: **end for**

Algorithm 2 Square-and-multiply modular exponentiation

Inputs: X, E, N; Output: $P_l = X^E \ mod \ N$
$E = \sum_{j=0}^{l-1} e_j 2^j, e_j \in \{0,1\}$

1: $P_0 = 1, Z_0 = X$
2: **for** $j = 0$ to $l-1$ **do**
3: $\quad Z_{j+1} = Z_j^2 \ mod \ N \ (square)$
4: $\quad$ **if** $e_j = 1$ **then**
5: $\quad\quad P_{j+1} = (P_j Z_j) \ mod \ N \ (multiply)$
6: $\quad$ **else**
7: $\quad\quad P_{j+1} = P_j$
8: $\quad$ **end if**
9: **end for**

radices, processing more exponent bits decreases the execution time but increases the resource consumption. Each round in Algorithm 2 uses MM twice, once for squaring and conditionally once for multiplication. As the squaring and multiplication are independent, they can be performed concurrently. When combined with MM, the pre-transformations are computed for X and P_0 and the post-transformation for P_l.

III. Exponentiation Accelerator Design

The exponentiation accelerator is implemented on an Altera Excalibur SoPC and partitioned into software control and hardware processing. The hardware consist of a MM entity and control logic for utilizing it in square-and-multiply exponentiations. The overall processing is controlled by the software. Since the processing requirements for the software part of the accelerator are very low, it can be run even in a microcontroller with very limited processing capabilities.

A. Implementation Platform

The implementation platform of the accelerator is the Altera Excalibur EPXA10 DDR development kit [1], shown in Fig. 1. The main component of the board is the EPXA10F1020C2 SoPC, which consists of an integrated ARM922T processor core and the Altera APEX20KE-like Programmable Logic Device (PLD). The PLD contains a large number of programmable Logic Elements (LE) and Embedded System Blocks (ESB) are provided for implementing a variety of memory functions. ARM9 and the PLD are connected through two Advanced Microcontroller Bus Architecture (AMBA) High-performance Bus (AHB) bridges, a shared Dual-Port RAM (DPRAM), and interrupt lines.

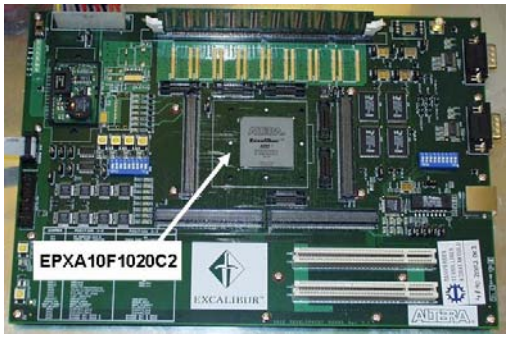


Fig. 1. Implementation platform of the accelerator.

B. Montgomery Multiplication Hardware

MM is implemented as a systolic array of Processing Units (PU) in the PLD. A PU, depicted in Fig. 2, processes an 8-bit piece of the line 4 of Algorithm 1 at a single clock cycle. A PU operates as follows:

1) An 8-bit piece of the modulus N is loaded to N_reg. Also, the next higher bit of N is loaded to n_reg.
2) An 8-bit piece of the multiplicand β is loaded to β_reg.
3) The signals a_i and q_i are input. The two additions are computed and the result is stored in δ_i_reg. The adder carries ($c1\_out$, $c2\_out$) are output to the higher PU and the lowest bit of the result ($\delta_i^0\_out$) to the lower PU. Also, a_i, q_i, and the control signals are forwarded to the higher PU.
4) Step 3 is repeated until all the bits of α have been processed. The 8-bit piece of the MM result is written to res_reg and also conditionally stored in β_reg.

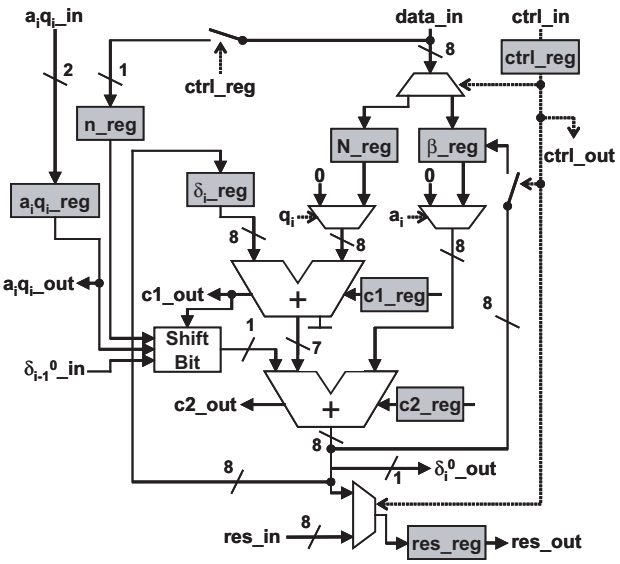


Fig. 2. An 8-bit processing unit.

As the processing is performed in the systolic architecture, the PUs compute their parts of the additions on concurrent clock cycles. Thus, $\delta^0_{i-1}\_in$ from the higher PU, n_reg, and $c1\_out$ are required for precomputing the highest bit of the shifted operand of the second adder (*Shift Bit*).

The MM systolic PU array is presented in Fig. 3. The number of required PUs is $n/8$, in which n is the bit length of N. For example, 128 PUs are required for a 1024-bit modulus. The highest PU is slightly different as it has to handle the highest bit of β and the carries internally. The input bits (a_i, q_i) and the control signals flow into PU_0 and propagate through the array, a PU at a clock cycle. Each PU is fed with the 8-bits of β synchronously with the control flow. Successive PUs compute their pieces of the sum at consecutive clock cycles. Upon finishing, the 8-bit MM results are shifted backwards from the higher PUs to the lower ones, through the array of the res_reg registers in the PUs. The highest bit of the result δ_{n+3} (h_bit) is output separately and stored in the control entity for further use. The next MM can begin already before the previous one is completely finished, i.e., when PU_0 is ready.

C. Modular Exponentiation Accelerator

The modular exponentiation accelerator design on the SoPC is shown in Fig. 4. The hardware consists of the MM array, storage elements, and control logic. The overall processing is controlled by the ARM9 software. *I/O Ctrl* transfers the exponentiation inputs and outputs between the PLD and ARM9 through the DPRAM. *I/O Ctrl* contains an AHB bridge slave, which is utilized for the software control. When the hardware has finished an exponentiation, ARM9 is interrupted (IRQ). The figure also includes a pseudo-code example of using the Application Programming Interface (API) of the accelerator provided in ARM9.

In order to initiate the accelerator, ARM9 computes $R2 = r^2 \ mod \ N$. It loads the modulus N into the PUs through *R2_RAM* and stores $R2$ in *R2_RAM*. The computation of $R2$ has to be performed only once for a given modulus. For example, in SRP it is likely that the modulus is changed infrequently [4]. In [7] it is also assumed that the modulus is known before the authentication is initiated, and thus, $R2$ can be precomputed and stored with N.

To begin an exponentiation, ARM9 loads the base X and the exponent E to *Z_RAM* and *E_RAM*, respectively, and releases the control to the hardware. *Loader* transforms the

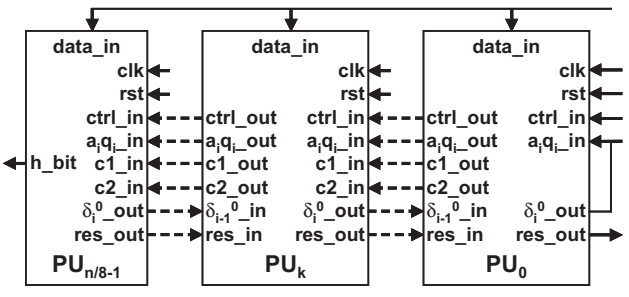


Fig. 3. Systolic processing array.

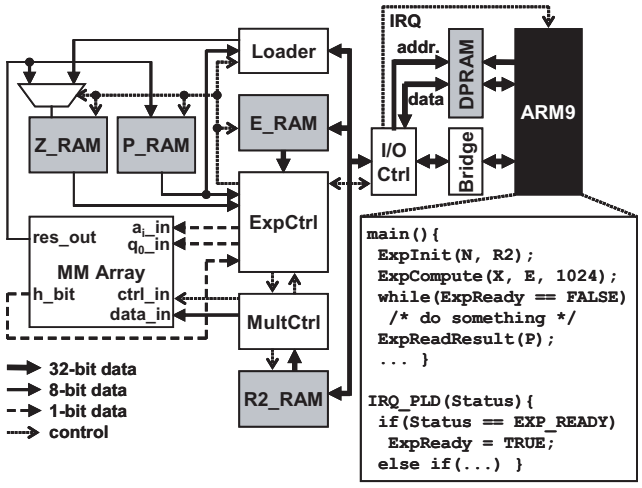


Fig. 4. Modular exponentiation accelerator on the SoPC.

32-bit *Z_RAM* inputs from *I/O Ctrl* into 8-bit pieces. All the RAM entities are implemented in the ESBs of the PLD.

In the exponentiation *ExpCtrl* reads in 8-bits of the multiplier (either from *P_RAM* or *Z_RAM*) and 32-bits of the exponent at a time and feeds the systolic array with a_i. It stores the multiplication results into their corresponding RAMs and instructs *Loader* to output the final exponentiation result. *MultCtrl* initializes the systolic array and feeds in the control signals during a single multiplication.

In the beginning of the exponentiation, MM is performed using X as the multiplier and $R2$ as the multiplicand for the pre-transformation. The multiplicand is read from *R2_RAM* in 8-bit pieces and input to the PUs through $data\_in$ as β. The multiplier is read from Z_RAM also in 8-bit pieces. Initially, q_0 is the lowest bit of the multiplier and the next values are the internal δ^0_i feedbacks in PU_0. The MM result is written to Z_RAM and P_RAM in 8-bit pieces and also kept as the next multiplicand (Z_0) in the PUs.

The hardware performs the modular exponentiation according to Algorithm 2 as follows. Computation is finished when all the exponent bits have been processed. The exponent length is defined as the third parameter of *ExpCompute* API function.

1) The contents of Z_RAM is used as the multiplier (Z_j) and the same Z_j in PUs as the multiplicand. The MM result is written back to Z_RAM as Z_{j+1} and also kept in PUs. This step is repeated until $e_{j+1} = 1$. When $e_{j+1} = 1$ the result is also stored in P_RAM, j is incremented, and the execution jumps to Step 3.
2) (*Multiply*) If $e_j = 0$ and $j < l-1$, j is incremented and the execution jumps to Step 3. If $e_j = 0$ and $j = l-1$, the PUs contain P_l and the execution jumps to Step 4. Otherwise, the contents of P_RAM is used as the multiplier (P_j) and Z_j in PUs as the multiplicand. If $j < l-1$, the MM result is written to P_RAM as P_{j+1}, Z_j is kept in PUs, and the execution moves to Step 3. If $j = l-1$, the computed P_l is kept in PUs and the execution moves to Step 4.

TABLE I

SYNTHESIS RESULTS FOR THE 1024-BIT ACCELERATOR.

Entity	LEs	Memory bits
Exponentiation	12,853	4,096
ARM-PLD Communications	461	0
I/O Ctrl	394	0
Loader	180	0
Others	72	0
Total	13,960	4,096
Maximum clock		63 MHz
Ref. [4]	9,644	5,120
Maximum clock		65 MHz
Ref. [2]	8,448	n/a
Maximum clock		42 MHz

3) (*Square*) If $j = l - 1$, the execution jumps to Step 2. Otherwise, the contents of Z_RAM is used as the multiplier (Z_j) and the same Z_j in PUs as the multiplicand. The MM result is written to Z_RAM as Z_{j+1} and kept in PUs. The execution moves to Step 2 and j is updated.
4) P_l in the PUs is multiplied with 1 to obtain the exponentiation result, which is written to P_RAM.

The final result is output via *Loader* and *I/O Ctrl* to the DPRAM and ARM9 is interrupted. The processor reads the exponentiation result from the DPRAM.

IV. RESULTS & COMPARISON

The synthesis results for the exponentiation accelerator with 1024-bit inputs on EPXA10F1020C2 produced by the Quartus v4.1 tool are presented in Table I. The table also presents results from our previous implementation [4] and the radix-2 implementation on which it was based [2]. In order to estimate the equivalent LE count of [2], the reported number of Xilinx Configurable Logic Blocks (CLB) has been doubled.

As shown, the LE count in this work has increased. In [4] and [2] the PUs were carefully tuned and placed to fit in a compact area whereas in this work the placement of the modified PUs was left to the synthesis tool. Also, certain buses are wider and the new design supports full length inputs compared to [4]. Memory bits have been saved from [4] since the modulus N is stored in the PUs instead of ESBs. The clock frequency has slightly decreased because the combinatorial path in PUs has been made longer to reduce cycle counts.

Despite the frequency decrease, the reduced cycle count increases the overall performance of the accelerator compared to the reference designs. The references interleave the squaring and multiplication of Algorithm 2. For interleaving, each PU contains an additional 8-bit register and operate in a 2-cycle mode, in which one cycle is used for squaring and the other for multiplication. The design of this paper operates in a 1-cycle mode, in which the PUs compute the multiplications and squarings in series. Whereas in the 2-cycle mode the line 5 of Algorithm 2 is computed every time (if $e_i = 0$, the result is discarded), in the 1-cycle mode it is only computed when $e_i = 1$. A MM operation is saved whenever $e_i = 0$. Statistically, the 1-cycle mode is expected to require $1.5 \times l$ MM operations whereas the 2-cycle mode always requires $2 \times l$ MM operations. With equal clock speeds the design of this paper is always faster than the reference designs.

TABLE II

COMPARISON OF CYCLE COUNTS OF A FULL EXPONENTIATION.

Design	Cycles
This work	$(n + k)(n + 4);\ 0 < k \leq n$
Ref. [4]	$(2n + 4)(n + 4)$
Ref. [2]	$(2n + 4)(n + 4)$

The parameter k is the number of ones in the binary representation of the exponent E. The parameter n is the bit length of N and E.

Compared to the reference implementations, the exponentiation control has also been tuned. Step 1 of Section III-C makes the pre-transformation of $P_0 = 1$ and the first multiplication with P_j unnecessary, saving two more MM operations. In addition, the control, e.g., skips squaring when $j = l - 1$.

Table II presents the comparison of the cycle counts of the designs for a full exponentiation ($l = n$). The table assumes that N and $R2$ are already computed and stored in the PLD. In the average case of $n = 1024$ and $k = 512$, the accelerator computes the exponentiation 25% faster than the previous designs at equal clock speeds. Compared to the software measurements of [5], the cycle counts are decreased by the factor of 11 from Pentium III and 19 from ARM9. The 1-cycle PUs with the tuned control can also be utilized for improving, e.g., the higher radix design [3] in the same way.

V. CONCLUSIONS

This paper presented a modular exponentiation accelerator utilizable in a number of security protocols. The accelerator processing was implemented in hardware and the control in software on a SoPC. The performance was increased by the factor of several tenfolds from software implementations. Compared to the previous radix-2 designs, significant performance increase through 1-cycle PUs and improved control was achieved. Also, the required amount of resources can be decreased to the same level with manual tuning.

REFERENCES

[1] "Altera website," http://www.altera.com, 2005.
[2] T. Blum and C. Paar, "Montgomery modular exponentiation on reconfigurable hardware," in *Proc. 14th IEEE Symp. Computer Arithmetic (ARITH 14)*, Adelaide, Australia, Apr. 14–16, 1999, pp. 70–77.
[3] ——, "High radix Montgomery modular exponentiation on reconfigurable hardware," *IEEE Trans. Computers*, vol. 50, no. 7, pp. 759–764, 2001.
[4] P. Groen, P. Hämäläinen, B. Juurlink, and T. D. Hämäläinen, "Accelerating the Secure Remote Password protocol using reconfigurable hardware," in *Proc. 2004 ACM Computer Frontiers Conf. (CF'04)*, Ischia, Italy, Apr. 14–16, 2004, pp. 471–480.
[5] P. Hämäläinen, M. Hännikäinen, M. Niemi, and T. D. Hämäläinen, "Performance evaluation of Secure Remote Password protocol," in *Proc. 2002 IEEE Int. Conf. Circuits and Systems (ISCAS 2002)*, vol. 3, Scottsdale, AZ, USA, May 26–29, 2002, pp. 29–32.
[6] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.
[7] T. Wu, "The SRP authentication and key exchange system," RFC 2945, Sept. 2000.
[8] L. Zhao, S. Makineni, and L. Bhuyan, "Anatomy and performance of SSL processing," in *Proc. 2005 IEEE Int. Symp. Performance Analysis of Systems and Software (ISPASS 2005)*, Austin, TX, USA, Mar. 20–22, 2005, pp. 197–206.

PUBLICATION 8

T. Alho, P. Hämäläinen, M. Hännikäinen, and T. D. Hämäläinen, "Design of a Compact Modular Exponentiation Accelerator for Modern FPGA Devices," in *Proceedings of World Automation Congress 2006 (WAC 2006) – Special Session on Information Security and Hardware Implementations*, Budapest, Hungary, July 24–27, 2006, 7 pages.

DESIGN OF A COMPACT MODULAR EXPONENTIATION ACCELERATOR FOR MODERN FPGA DEVICES

Timo Alho, Tampere Univ. of Tech., Finland, timo.a.alho@tut.fi
Panu Hämäläinen, Tampere Univ. of Tech., Finland, panu.hamalainen@tut.fi
Marko Hännikäinen, Tampere Univ. of Tech., Finland, marko.hannikainen@tut.fi
Timo D. Hämäläinen, Tampere Univ. of Tech., Finland, timo.d.hamalainen@tut.fi

ABSTRACT

We present a compact FPGA implementation of a modular exponentiation accelerator suited for cryptographic applications. The implementation efficiently exploits the properties of modern FPGAs. The accelerator consumes 341 logic elements, 1 DSP block, and 13 604 memory bits in Altera Stratix EP1S40. It performs modular exponentiations with up to 2250-bit integers and scales easily to larger exponentiations. Excluding pre and post processing time, 1024-bit and 2048-bit exponentiations are performed in 28.03 ms and 212.09 ms, respectively. Due to its compactness, standard interface, and support for different clock domains, the accelerator can effortlessly be integrated into a larger system in the same FPGA.

KEYWORDS: Modular exponentiation, Montgomery multiplication, cryptography, hardware, FPGA, compact.

1. INTRODUCTION

Modular exponentiation of long integers is required in a number of public-key cryptosystems, e.g. RSA, Digital Signature Algorithm (DSA), Diffie-Hellman key exchange protocol, and Secure Remote Password protocol (SRP). Performing such an operation is computationally very expensive. Efficient exponentiation implementations are especially required in wireless and embedded devices with limited processing capabilities as well as in heavily used network servers and firewalls [1][2]. Compared to software implementations on general-purpose CPUs, the exponentiations can be computed significantly more efficiently with a hardware design tailored for the task [3].

Reprogrammable logic circuits, specifically Field Programmable Gate Arrays (FPGA), offer a cost-effective and high-performance hardware alternative to Application Specific Integrated Circuits (ASIC) in low- and mid-volume products. Furthermore, FPGAs are becoming important building blocks in embedded systems in general [3]. According to the recent trend, they are no longer used as single parts of embedded systems but rather as System-on-Chip (SoC) platforms for implementing complete applications. Modern, commercial FPGA devices contain large functional blocks, such as high-speed multipliers, embedded multiport memories, Phase Locked Loops (PLL), and even programmable CPU cores. For cryptographic applications FPGAs offer high performance, possibility to modify and change algorithms in already fielded devices, and potential to share resources through run-time reconfiguration [3].

By carefully mapping designs to the resources of modern FPGA devices, compact and high-performance implementations can be realized. Specifically, the computationally expensive modular exponentiations, which are widely required in cryptographic applications, can significantly benefit from these resources. In this paper we present the design of a compact exponentiation accelerator for modern FPGA devices. Due to the compact size, standard interface, and support for different clock domains, the accelerator can effortlessly be used as a part of a SoC implemented in the same FPGA device. Furthermore, the design can easily be scaled for larger operand sizes by reserving more embedded memory.

2. MODULAR ARITHMETIC

Modular exponentiations are typically calculated using repeated square-and-multiply algorithms with modular reductions in between. The most basic type of these algorithms is the binary modular exponentiation that has two variations, right-to-left and left-to-right, presented as Algorithm 1 and Algorithm 2. More efficient algorithms reducing the number of multiplications and squarings also exist [4] but they are more complex and require more resources as a hardware implementation. For the compact area and simplicity we chose to use the left-to-right binary exponentiation in this work. Compared to the right-to-left algorithm, only one temporary result (P) needs to be stored instead of two (P and Z). The advantage of the right-to-left algorithm is that it allows calculating multiplications and squarings in parallel. However, this requires two separate arithmetic units, increasing the area of the hardware.

Since the exponentiation algorithm consists of repeated multiplications and squarings, an efficient multiplication algorithm is required. Montgomery Multiplication (MM) [5] is a widely used method for performing combined modular reductions and multiplications [3]. The version of MM that we utilize in this work is based on Algorithm 3 [6]. The algorithm calculates MM(A, B, M) = ABR^{-1} mod M which is, with a suitable choice for R, significantly more efficient to compute on a typical processor than AB mod M.

In order to cope with the extra term R^{-1}, we need to transform the operands to a special form called M-residue respect to R, defined as $A^* = AR \bmod M$ [4]. This transformation can be performed with a single MM operation since MM(A, R^2, M) = $AR^2R^{-1} \bmod M = AR \bmod M = A^*$. The inverse transformation can also be performed with MM since MM(A^*, 1, M) = $ARR^{-1} \bmod M = A$. Despite of the conversions, MM is specifically beneficial in modular exponentiation, in which the MM results are repeatedly multiplied using the same modulus. The conversions are required only for the initial input and the final result.

Algorithm 4 presents the digit-serial modification of the MM algorithm used in this work. A similar variation of the algorithm, which is referred to as Finely Integrated Operand Scanning (FIOS), is presented in [11]. The algorithm uses multi-precision integers with k-bit digits as inputs and the intermediate variable c of size k+1 bits. The notation $(c, s) \leftarrow x$ denotes that k least significant bits of x are assigned to s and the most significant k+1 bits to c. The division by 2^k in Algorithm 3 is satisfied by delaying the inputs b_i and a_j one cycle. In [6] it is shown that the final result $S(n+3)$ is bounded by $2\tilde{M}$, which guarantees that it can be fed back as an input for the next multiplication. Also, the intermediate result $S(i)$ is bounded by $2^k A + \tilde{M}$ and fits into n+3 k-bit digits. Therefore, the last step of Algorithm 4 is valid even though c is one bit wider than s.

3. FPGA APPROACHES FOR MONTGOMERY MULTIPLICATION

A typical hardware implementation approach is to partition the MM algorithm so that the n-bit multiplicand B is decomposed into n/k digits, 2^k being the chosen radix, while the multiplier A and modulus M are used in their full widths. However, when performing the additions of Algorithm 3 with these long integers, the carry propagation delay becomes a problem. In order to

Input : Positive integers
$x, M, E = (e_{n-1}e_{n-2}\ldots e_1e_0)_2$
Output : $P = x^E \bmod M$

$P \leftarrow 1, Z \leftarrow x$
for $i = 0$ **to** $n-1$ **do**
  $Z \leftarrow Z \times Z \,(\bmod M)$
  if e_i=1 **then** $P \leftarrow P \times Z \,(\bmod M)$
end for

Algorithm 1. Right-to-left binary modular exponentiation.

Input : Positive integers
$x, M, E = (1e_{n-2}\ldots e_1e_0)_2$
Output : $P = x^E \bmod M$

$P \leftarrow x$
for $i = n-2$ **to** 0 **do**
  $P \leftarrow P \times P \,(\bmod M)$
  if e_i=1 **then** $P \leftarrow P \times x \,(\bmod M)$
end for

Algorithm 2. Left-to-right binary modular exponentiation.

Input : $M > 2$ and M is odd.
Positive integers k and n such that
$4\tilde{M} < 2^{k(n+2)}$, where $\tilde{M} = (M' \bmod 2^k)M$
and $M' = -M^{-1} \bmod 2^{k(n+2)}$.
A and $B = \sum_{i=0}^{n+2} 2^{ik} b_i$, where
$b_i \in \{0,1,\ldots 2^k - 1\}, b_{n+2} = 0,\ 0 \le A, B < 2\tilde{M}$.

Output : $S(n+3) \equiv ABR^{-1} \bmod M$,
where $S(n+3) < 2\tilde{M}$,
and $2^{k(n+2)} R^{-1} \bmod M = 1$.

$S(0) \leftarrow 0$
for $i = 0$ **to** $n+2$ **do**
$q_i \leftarrow S(i) \bmod 2^k$
$S(i+1) \leftarrow (S(i) + q_i \times \tilde{M}) / 2^k + b_i \times A$
end for

Algorithm 3. Montgomery modular multiplication.

Input : $\tilde{M} = \sum_{i=0}^{n+2} 2^{ik} \tilde{m}_i, \tilde{m}_i \in \{0,1,\ldots 2^k - 1\}$,
$\tilde{m}_{n+2} = 0, \tilde{m}_{n+1} = 0$.
$A = \sum_{i=0}^{n+1} 2^{ik} a_i, a_i \in \{0,1,\ldots 2^k - 1\}$
$B = \sum_{i=0}^{n+2} 2^{ik} b_i, b_i \in \{0,1,\ldots 2^k - 1\}, b_{n+2} = 0$,
where $\tilde{M} = MM'$, $M > 2$, M is odd,
$M' = -M^{-1} \bmod 2^k$, and $0 \le A, B < 2\tilde{M}$.

Output : $S(n+3) \equiv ABR^{-1} \bmod M$,
where $S(n+3) < 2\tilde{M}$ and $2^{k(n+2)} R^{-1} \bmod M = 1$.

$S(0) \leftarrow 0$
for $i = 0$ **to** $n+2$ **do**
$q_i \leftarrow s(i)_0$
$(c, -) \leftarrow s(i)_0 + q_i \times \tilde{m}_0$
for $j = 0$ **to** $n+1$ **do**
$(c, s(i+1)_j) \leftarrow s(i)_{j+1} + q_i \times \tilde{m}_{j+1} + b_i \times a_j + c$
end for
$s(i+1)_{n+2} \leftarrow c$
end for
where $S(i) = \sum_{j=0}^{n+2} 2^{jk} s(i)_j$,
$s(i)_j \in \{0,1,\ldots 2^k - 1\}$ and $c \in \{0,1,\ldots 2^{k+1} - 1\}$

Algorithm 4. Digit-serial Montgomery multiplication.

overcome this, a redundant number system can be used [7][8] or the additions can be performed in a systolic array [9][10]. Typically, these methods result in high performance but also in high consumption of computational resources in FPGAs as each addition is performed fully parallel.

As an alternative partitioning supporting compact designs, all the MM inputs can be decomposed into n/k digits and the computations performed a digit at a time (digit-serial). The method resembles a software approach in which the intuitive choice for radix 2^k is the word size of the processor [11]. With this partitioning the amount of FPGA resources dedicated for computation can be decreased at the expense of performance. Ref. [14] presents an implementation in which the multiplicand is in the radix-2 form, the multiplier is broken up into k-bit digits, and the computations are performed in the serial fashion.

Ref. [12] presents an instruction set extension for modular arithmetic that enables computing the operation $x \times y + z + w$ in one clock cycle and thus speeds up the digit-serial loop of Montgomery's algorithm. The paper reports considerable speedup on a MIPS32-compatible CPU when the instruction set extension is used. In addition, the approach scales well to different key sizes as well as algorithms. The method can also easily be adapted to FPGA implementations based on soft-core CPUs, such as NIOS II of Altera [13]. In this paper we utilize a related approach in our MM data path.

According to our knowledge, the only modular exponentiation implementation efficiently exploiting the embedded multipliers of a modern FPGA device is reported in [7]. The implementation uses a large number of multipliers in parallel on a Xilinx FPGA, resulting in a high-performance design. However, unlike in our implementation, the amount of consumed resources is very high and thus the implementation allows integrating only very little other functionalities into the same FPGA.

Clock cycles for a modular multiplication	$(n+3)(n+4)$
Number of modular multiplications for modular exponentiation	$(l+p)$, where l is the bit length of the exponent and p is the number of ones in its binary representation
Required pre processing	given $x < M$, $M < 2^{kn}$, $R = 2^{k(n+2)}$ calculate: $xR = xR \bmod M$ and $\tilde{M} = MM'$
Required post processing	given $P \equiv x^e R \bmod M$ calculate: $x^e \bmod M = PR^{-1} \bmod M$

Table 1. Summary of accelerator features.

4. ACCELERATOR ARCHITECTURE

This section describes the architecture of our modular exponentiation accelerator. As the accelerator is intended to be used as a part of a larger SoC containing also a general-purpose processor, it only computes the most time consuming parts of the modular exponentiation. That is, the pre and post transformations must be performed outside the accelerator. The features of our accelerator design are summarized in Table 1.

4.1. FPGA Platform

We used Altera Stratix EP1S40 as the target device. It is a modern FPGA containing a large number of programmable Logic Elements (LE) as well as RAM and Digital Signal Processing (DSP) blocks. In our design the RAM blocks are used as Dual-Port RAM (DPRAM) and shift registers, and the DSP blocks are configured to a multiplier-adder mode. Similar building blocks can also be found in the Stratix II and Cyclone II devices of Altera as well as the Virtex II, Virtex 4, and Spartan 3 devices of Xilinx.

4.2. Exponentiation Architecture

Figure 1 depicts the high-level architecture of the accelerator. Initially, the accelerator inputs are stored into the DPRAMs *xR-ram*, *M-ram*, and *E-ram* as k-bit digits. In addition, the number of iterations for a single multiplication (n) and the total number of bits in the exponent are passed to the control logic. During the exponentiation, *s1-ram* and *s2-ram* are used as temporary storages, one storing the intermediate result ($S(i)$ of Algorithm 4) during multiplication and the other one holding the result of the previous multiplications (P of Algorithm 2).

The exponentiation control logic, *Exp control*, manages the four multiplexers. Output of p_1 is always selected to be the result of the previous multiplication (P) and the output of p_2 to be the intermediate result of the active multiplication ($S(i)$). The multiplexers p_a and p_b select which multiplication, $xR \times xR$, $xR \times P$, or $P \times P$, is calculated. The registers before the multiplexers p_a and p_b are used for delaying b_i and a_j by one clock cycle.

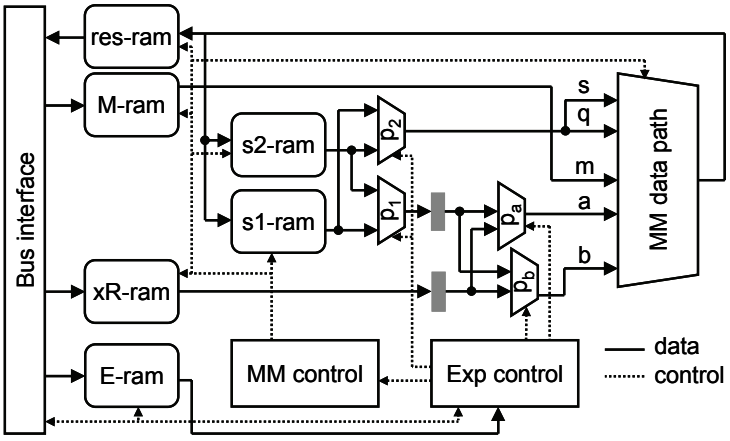


Figure 1. Accelerator architecture.

4.3. Modular Multiplier

For MM the integers A, B, and the pre-calculated $\tilde{M}$ are broken up to operands of length k-bits as required by Algorithm 4. The modular multiplier entity consists of a control state machine, *MM control*, and a data path performing the computations of the body of the inner loop in Algorithm 4. *MM control* counts the values of the loop variables i and j and uses them for addressing the operand memories. In addition, it maintains a delayed version of the counter j for addressing *res-ram*, *s1-ram*, or *s2-ram* to write back the results in parallel with operand reading. At the beginning of the each iteration of i, MM control loads the values of $q_i = s(i)_0$ and b_i into the data path input and clears the intermediate variable c. During the first iteration of i the data path input is cleared. The data path is shown in Figure 2(a). The result of the data path addition always fits into $2k$+1 bits.

4.4. Mapping to FPGA

Figure 2(b) depicts the mapping of the data path to our target FPGA device, data path being 18 bits wide. The intermediate variable c is divided into two registers, r and r', and always computed during one clock cycle. The data path is pipelined into five stages for minimizing the combinatorial logic delays. However, the pipelining does not cause any stalls in the operation, since the only data dependencies between back-to-back operations are due to the intermediate variable c.

Each DSP block in EP1S40 can be configured to support either eight 9-bit, four 18-bit, or one 36-bit multipliers. In addition, the DSP blocks contain various other elements such as registers and adder/accumulator blocks. We chose to use the data path width of 18 bits, which allows us to perform two multiplications, sum the results of the multiplications, and pipeline the calculations using only half of the resources of one DSP block. The shift register for the input s is mapped into a RAM block and the two additional adders and the rest of the registers into LEs.

The operand, intermediate, and result memories are mapped into RAM blocks configured to DPRAM mode with one read and one write port. This enables simultaneous writing of results and reading of the next operands. The DPRAMs of the accelerator interface (*E-ram*, *xR-ram*, *M-ram*, and *res-ram*) are also configured to support two clock domains, allowing running the accelerator at a higher clock speed than the rest of the system. The support for different clock domains explains also the extra *res-ram*, which could otherwise be replaced by *s1-ram* or *s2-ram*.

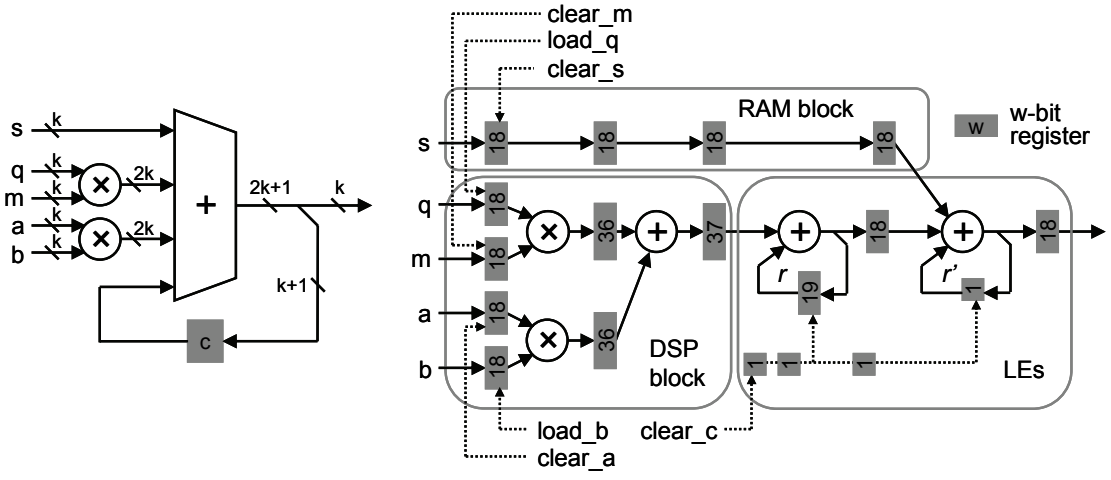


Figure 2. Montgomery multiplication data path: (a) architecture and (b) mapping to EP1S40.

5. RESULTS AND COMPARISON

The synthesis results of our accelerator implementation including a fully functional, standard bus interface to NIOS II are summarized in Table 2. The accelerator was described in VHDL and synthesized using Altera Quartus II 5.0. The memory sizes were chosen to be 128 x 18 bits, which supports computing full exponentiations with up to 2250-bit operands. The design can be scaled for longer exponentiations simply by increasing the amount of memory. As can be seen, our accelerator consumes only a very small amount of resources on the target FPGA.

For the execution time comparison, we measured that a full 1024-bit exponentiation takes over a second in the 32-bit NIOS II at 100 MHz with a C-language OpenSSL implementation. By carefully hand tuning the software for NIOS II, the performance could be improved, but not to the level of our accelerator. Ref. [15] presents a highly optimized software implementation in a DSP processor. They report that a 1024-bit RSA signing can be performed in 11.7 ms at 200 MHz by exploiting the Chinese Remainder Theorem (CRT), which makes direct execution time comparisons impossible. Compared to our data path, the DSP processor data path would also very likely map less cost-efficiently to the FPGA resulting in larger resource consumption.

Table 2 compares our implementation with other MM-based modular exponentiation FPGA implementations relevant to this work. It should be noted that a fair comparison is difficult as different FPGA technologies are used. All the reference implementations support 1024-bit exponentiations. One Xilinx *slice* corresponds roughly to two Altera LEs. Ref. [9] presents a radix-16 systolic array design mapped carefully to slices for high performance. A similar design for radix-2 with tuned control on an Altera device is reported in [10]. Ref [14] is a serial implementation that exploits the characteristics of slices for compact size. The execution times in Table 2 exclude the pre and post processing, which varies among the implementations. We have calculated the full execution time for [14] since it has not explicitly been reported.

The comparison shows that the consumption of the general-purpose resources (LEs/slices) in our implementation is considerably lower than in the reference implementations. Whereas [7] utilizes 62 multipliers, we only need two, residing in a single DSP block. Our execution time is significantly better than that of the most compact reference [14]. Compared to [7], [9], and [10], the execution time of our design for the 1024-bit exponentiation is longer but we also support longer exponentiations with a single implementation.

If the target application is parallelizable, its performance can be further improved by

Design	FPGA device	Logic blocks	Memory bits	DSP elements	Max. freq. [MHz]	Op. length [bits]	Execution time [ms]
Ours	Altera Stratix EP1S40	341 LEs (0.8%)	13 604 [(1)] (0.4%)	1 DSP [(2)] (7.1%)	198	1024	28.03
						2048	212.09
Ref. [7]	Xilinx Virtex-II XC2V3000	14 334 slices	- [(3)]	62 18-bit multipliers	90	1024	2.33
Ref. [9]	Xilinx XC40250XV	6 633 CLBs [(4)]	- [(3)]	-	45	1024	11.95
Ref. [10]	Altera EPXA10	13 960 LEs	4096	-	63	1024	25.06
Ref. [14]	Xilinx Virtex-E 2000-8	1 188 slices	- [(3)]	-	86	1024	208

[(1)] Memory bits are mapped to 11 M512 (2.9%) and four M4K (2.2%) RAM blocks.
[(2)] Half of the resources of the DSP block are free, including two 18-bit multipliers.
[(3)] A slice can be configured to two 16 x 1 bit memories. Thus, dedicated memory blocks are not necessary.
[(4)] One XC40250XV Configurable Logic Block (CLB) equals to one Virtex slice.

Table 2. Results and comparison (percentages of total resources in parenthesis).

utilizing multiple parallel accelerators. This is the case e.g. with RSA, in which the modulus *M* is a product of two or more primes [4]. When the primes are known, the exponentiation can be divided into smaller, parallel exponentiations using CRT.

6. CONCLUSION

In this paper we presented the design of a compact modular exponentiation accelerator suited for a number of cryptographic applications. By mapping the design to the resources of modern FPGA devices, we achieved high performance with very low resource consumption. In addition, our implementation is effortlessly scalable to exponentiations with larger operands. Due to its compact size, standard bus interface, and support for different clock domains, the accelerator can easily be integrated into a larger SoC implemented in the same FPGA.

7. REFERENCES

[1] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in embedded systems: design challenges," *ACM Trans. Embedded Comp. Systems*, vol. 3, no. 3, pp. 461-491, Aug. 2004.

[2] L. Zhao, R. Iyer, S.Makineni, L. Bhuyan, "Anatomy and performance of SSL processing," in *Proc. IEEE Int. Symp. Performance Analysis of Systems and Software*, March 2005, pp. 197-206.

[3] T. Wollinger, J. Guajardo, and C. Paar, "Security on FPGAs: state-of-the-art implementations and attacks," *ACM Trans. Embedded Comp. Systems*, vol. 3, no. 3, pp. 534-574, Aug. 2004.

[4] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*, Boca Raton (CA): CRC Press, 1997.

[5] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519-521, Apr. 1985.

[6] H. Orup, "Simplifying quotient determination in high-radix modular multiplication," in *Proc. 12th Symp. Computer Arithmetic*, July 1995, pp. 193-199.

[7] S. Tang, K. Tsui, and P. Leong, "Modular exponentiation using parallel multipliers," in *Proc. IEEE Int. Conf. Field-Programmable Technology*, Dec. 2003, pp. 52-59.

[8] S. E. Elridge and C. D. Walter, "Hardware implementation of Montgomery's modular multiplication algorithm," *IEEE Trans. Comput.,* vol. 42, no. 6, pp. 693-699, June 1993.

[9] T. Blum and C. Paar, "High radix Montgomery modular exponentiation on reconfigurable hardware," *IEEE Trans. Comput.,* vol. 50, no. 7, pp. 759-764, July 2001.

[10] P. Hämäläinen, N. Liu, M. Hännikäinen, and T. D. Hämäläinen, "Acceleration of modular exponentiation on system-on-a-programmable-chip", in *Proc. IEEE Int. Symp. System-on-Chip*, Nov. 2005, pp. 14-17.

[11] Ç. K. Koç and B. S. Kalinski Jr., "Analyzing and comparing Montgomery multiplication algorithms," *IEEE Micro*, vol. 16, pp. 26-33, June 1996.

[12] J. Großschädl, "Instruction set extension for long integer modulo arithmetic on RISC-based smart cards," in *Proc. 14th Symp. Computer Architecture and High Performance Computing*, Oct. 2002, pp. 13-19.

[13] Altera Corporation, "Nios II Embedded Processor," [Online document], [cited 2005 Nov 21], Available at HTTP: http://www.altera.com/products/ip/processors/nios2/ni2-index.html

[14] A. Mazzeo, L. Romano, and G. P. Saggese, "FPGA-based implementation of a serial RSA processor," in *Proc. Design, Automation and Test in Europe*, 2003, pp. 582-587.

[15] K. Itoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara, "Fast implementation of public-key cryptography on a DSP TMS320C6201," in *Proc. 1st Int. Workshop Cryptographic Hardware and Embedded Systems (CHES 1999)*, 1999, pp. 61-72.

PUBLICATION 9

P. Hämäläinen, N. Liu, R. Sterling, M. Hännikäinen, and T. D. Hämäläinen, “Design and Implementation of an Enhanced Security Layer for Bluetooth,” in *Proceedings of the 8th IEEE International Conference on Telecommunications (ConTEL 2005)*, Zagreb, Croatia, June 15–17, 2005, pp. 575–582.

Design and Implementation of an Enhanced Security Layer for Bluetooth

P. Hämäläinen, N. Liu, R. Sterling, M. Hännikäinen, and T. D. Hämäläinen
Tampere University of Technology / Institute of Digital and Computer Systems
P. O. Box 553, FI-33101 Tampere, Finland
{panu.hamalainen, ning.liu, risto.leppisaari, marko.hannikainen, timo.d.hamalainen}@tut.fi

Abstract— **This paper proposes a new Enhanced Security Layer (ESL) for Bluetooth. The security level is increased by replacing the encryption with AES and adding integrity protection. As ESL is placed on the top of the standard controller interface, it can be integrated into any Bluetooth implementation. A prototype implementation of ESL is presented. The security processing is implemented in hardware for high performance. The design consumes fewer resources and has higher throughput (214 Mb/s) than the standard design. The programming interface supports straightforward application development.**

I. Introduction

Bluetooth [1] is a technology for short-range wireless communications developed by Bluetooth Special Interest Group (SIG). Originally it was intended as a simple, low-cost and low-power serial cable replacement for any electronic device. However, the number of application scenarios has significantly grown from the original purposes. Presently Bluetooth allows ad hoc networking and access point functionality for Internet connections. The ongoing development is extending the technology with new features, such as support for quality of service, higher data rates, multicasting, and lower power consumption. Currently, Bluetooth can be found in mobile phones, PDAs, laptops, printers, digital cameras, headsets, portable payment terminals (e.g. for facilitating credit card payments in restaurants), cars, and medical equipment. The application area expands as new products with the Bluetooth capability are constantly introduced.

In addition to low-cost and robust operation, Bluetooth applications often require protected communications. For example, private data transfers between personal devices and transactions with payment terminals have to be protected. Due to the wireless link, the transmitted data are available to anyone within the radio coverage without security procedures. The Bluetooth specification [1] defines methods for key management, authentication, and encryption. Unfortunately, researchers have identified several vulnerabilities in the security design of Bluetooth. They derive from the usage of Personal Identification Number (PIN) in authentication key generation, improper key management and authentication, and the possibility of tracking Bluetooth devices. The encryption algorithm has been found to provide lower level of security than what was intended. The transmitted data should also be protected with a cryptographic integrity check, i.e., with a Message Authentication Code (MAC).

This paper proposes a new Enhanced Security Layer (ESL) for improving the security of Bluetooth technology. The security level is increased by replacing the original Bluetooth encryption with a design based on Advanced Encryption Algorithm (AES) [2]. ESL adds MACs to the transmitted data for cryptographic integrity protection. The proposed layer is added on the top of the standard Bluetooth controller interface, which allows integrating it as additional module into any standard Bluetooth chip or as a software layer into a host.

The paper also presents the prototype implementation of ESL. For high performance and low energy, the security processing of ESL is implemented in hardware. The design supports the new *Counter mode with Cipher block chaining Message authentication code* (CCM) encryption mode [3], which is adopted into several new wireless IEEE standards, e.g., 802.11i. Thus, the same hardware design can be utilized in other, standard implementations. In addition to the enhanced security, the ESL implementation offers an Application Programming Interface (API) for a Bluetooth device by hiding the low-level Bluetooth commands from the application.

The paper is organized as follows. In Section II the Bluetooth technology is briefly introduced and its security vulnerabilities are reviewed. Section III presents the proposed Bluetooth ESL design. The implementation of ESL is presented in Section IV. Finally, Section V concludes the paper and outlines the future work.

II. Bluetooth Overview

The Bluetooth technology [1] consists of several protocol layers. The protocol stack is depicted in Fig. 1. Host Controller Interface (HCI) separates the stack into two parts, *Bluetooth host* and *Bluetooth controller*. It provides the host with a low-level, uniform interface to the hardware capabilities of the controller. The host is connected to the HCI firmware through a physical bus, such as UART or USB.

Originally, the nominal transmission rate of the Bluetooth radio was 1 Mb/s. The newly released specification extends the rate to 3 Mb/s [1]. There are two types of Bluetooth links: Asynchronous Connectionless (ACL) and Synchronous Connection Oriented (SCO). ACL is utilized for data transfers and SCO for audio. Both the links have several network packet types of different lengths. The host transmits and receives data in HCI data packets. The HCI packets are fragmented to and assembled from the ACL

8th International Conference on Telecommunications - ConTEL 2005
ISBN: 953-184-081-4, June 15-17, 2005, Zagreb, Croatia

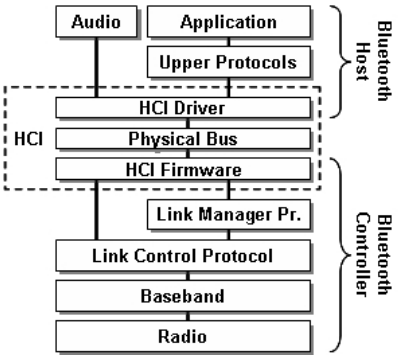


Fig. 1. Bluetooth protocol stack.

and SCO network packets by the Bluetooth controller. The highest data payload rate is 723.2 kb/s with the 1 Mb/s radio and 2178.1 kb/s with the 3 Mb/s radio over an asymmetric ACL link with the largest packets [1].

The Bluetooth security design provides key management, authentication, and encryption for link level protection. The utilized encryption algorithm is a stream cipher called E_0. It is based on linear feedback shift registers. Authentication and key generation utilize SAFER+ block cipher. The standard security design and its hardware implementation are presented in [4].

A. Bluetooth Security Vulnerabilities

The security design of Bluetooth has been found vulnerable. Ref. [5] was the first to identify several attacks. An attacker is able to recover link and encryption keys passively by exhaustively searching PIN values. Impersonation and eavesdropping are possible after valid communication with a *unit key*. Consequently, its usage is deprecated in the newest version of the specification [1]. Another type of attack enables tracking Bluetooth devices by passively listening to their transmissions. Ref. [6] demonstrates man-in-the-middle attacks that can be implemented during the authentication procedure by relaying messages. Relaying is not beneficial if encryption is applied after authentication. In [7] more advanced man-in-the-middle attacks are presented.

The lack of cryptographic integrity protection allows an attacker to manipulate Bluetooth transmissions without detection. Particularly, the utilization of the stream cipher makes this applicable. If the transmitted plaintext is known, the attacker can change the contents into whatever she wishes by flipping bits. For example, this allows conveying higher protocol data into a different destination by altering the higher layer addressing fields which often reside at known locations in Bluetooth packets. In order to prevent all the attacks of [7], it is stated that adding MAC computation into Bluetooth is inevitable. Ref. [8] suggests a MAC scheme with user authentication. However, the scheme is not clearly described.

In addition to the protocol attacks, weaknesses in the Bluetooth stream cipher have been exposed. For example, [5], [9], [10], [11], and [12] show that the cipher does not provide as good security as a 128-bit-key algorithm should. Even though the attacks so far are theoretical and cannot be benefited from in practice, they may become practical in the future. Thus, the Bluetooth SIG is considering replacing the Bluetooth encryption with a new, probably AES-based design [13].

III. Enhanced Security Layer

In order to overcome the shortcomings of the Bluetooth encryption algorithm and fix the lack of cryptographic integrity protection, Enhanced Security Layer (ESL) for Bluetooth is designed. ESL replaces the proprietary encryption with AES and adds MACs to the transmitted data. It supports four well-scrutinized encryption modes with different characteristics and processing requirements. The application may choose the preferred one according to its security requirements. In addition to the increased security, ESL hides the security processing and low-level HCI operations and provides a high-level application programming interface (ESL API) for straightforward application development. To further improve the level of security, the design allows extending ESL to support, e.g., enhanced authentication and key exchange techniques. For the interoperability with devices that do not contain the enhanced features, the support for the standard Bluetooth security is also included in ESL.

The ESL architecture is presented in Fig. 2. As shown, ESL is placed above HCI. Generally, Bluetooth technology is provided as fixed chips, which implement the Bluetooth functionalities below HCI. Application developers have only access to the Bluetooth controller through the standard HCI. Therefore, in order to improve the security, the most universally applicable method is to add the enhancements above HCI. This way ESL can be integrated as an additional module into any standard Bluetooth controller or host.

The same method is used, e.g., in the implementations of the redesigned 802.11 Wireless Local Area Network (WLAN) authentication (802.11i). Instead of reimplementing WLAN cards, the old authentication is disabled and the existing cards are only utilized for transferring higher layer data. The new authentication is performed above the old WLAN link layer. This requires only reimplementing the software in the host computers.

Another advantage of placing ESL on the top of HCI and not into the baseband is that the method results in lower packet overhead. Added protocol fields are transmitted in the HCI data packets instead of every Bluetooth network packet.

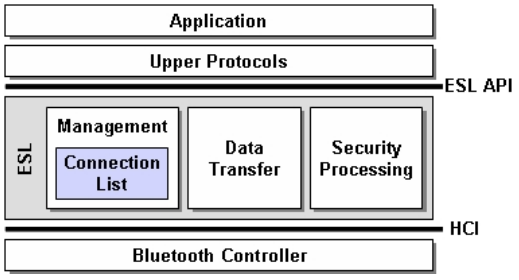


Fig. 2. Architecture of Bluetooth ESL.

ConTEL 2005, ISBN: 953-184-081-4

A. Enhanced Security Design

The AES algorithm [2] utilized in ESL is a symmetric cipher that encrypts data in 128-bit blocks. It supports key sizes of 128, 192, and 256 bits. As several other block ciphers, AES consists of successive, similar iteration rounds. Depending on the chosen key size, the number of the rounds is 10, 12, or 14. Each round mixes the data with a roundkey, which is generated from the encryption key. Currently, the 128-bit-key version is generally considered to provide adequate security and it is also utilized in this work. Decryption requires inverting the iterations resulting in at least partly separate data path. However, the encryption modes of ESL only use the forward cipher.

Applying an encryption algorithm alone without a proper encryption mode is not secure. The counter mode (CTR) [14] is generally regarded as a good choice and it is also utilized in ESL. CTR has a proven security bound [15] and it provides most performance trade-offs for implementations. In CTR a block cipher produces a key stream from a secret key and a counter. The key stream is generated a block at a time by encrypting counter values until the stream length matches the data length. After each algorithm pass the counter is incremented. The stream is exclusive-ored (XOR) with the plaintext to get the ciphertext and vice versa. If the data length is not a multiple of the block length, only the required bits of the last key stream block are used. It is important that the same counter value is used only once during the lifetime of a key. Another security requirement is that at maximum 2^{64} encryptions are be performed per key.

The CTR mode requires that the encrypted data is accompanied with MAC. Otherwise the bit manipulation attacks of Section II-A still apply. In this work MAC is computed using the Cipher Block Chaining MAC (CBC-MAC) method. This allows utilizing the same algorithm for both encryption and data authentication. CBC is a feedback encryption mode in which the previous ciphertext block is XORed with the plaintext block before encryption. CBC-MAC operates in the same way, except that only the result of the last encryption is output as MAC. CBC-MAC has a security proof [16].

ESL supports plain CBC-MAC (*MAC mode*) and two combinations of the CTR encryption and CBC-MAC computation. In the combined modes MAC may be computed over the plaintext (*MAC-then-encrypt mode*) or over the ciphertext (*encrypt-then-MAC mode*). The phases utilize separate keys. In the MAC-then-encrypt mode MAC is encrypted. The MAC mode is suited for applications in which privacy is not required. The combined modes provide both privacy and authenticity. In [17] it has been shown that the encrypt-then-MAC mode is generally secure. However, for example, in [18] it is suggested that it should be the plaintext that is authenticated. Adding the other mode to a design that implements one of the modes requires only little additional resources. Thus, both the modes are supported in ESL.

In addition, ESL supports a special MAC-then-encrypt mode, CCM [3]. Originally CCM was proposed to the IEEE 802.11i working group for improving the security of the IEEE 802.11 WLAN. It was also adopted. Later it has been sent for the evaluation as a standard block encryption mode. The described components of the mode, CTR and CBC-MAC, have been well-known for decades but CCM is a new definition for their combined usage. The security of the CCM mode has been proven in [19].

Whereas the CTR encryption can process arbitrary length data, CBC-MAC requires that the input is padded to match a block boundary. In ESL the last input data block is padded with zeroes if required in all the modes. The padding does not affect CTR. Instead, the en/decrypted output is truncated back to the original length. The MAC size can be chosen to be 64 or 128 bits, which allows trade-offs between the protection level and communication overhead. If the 64-bit MAC is chosen, the 64 least significant bits of the MAC output are discarded. MAC is appended to the HCI payload data.

In order to prevent repeated counter values, the CTR counter is composed of concatenated *nonce* and *block counter* values in all the combined modes. The nonce is a constant value for a transmission and the block counter is incremented for each data block within the transmission. The nonce is provided by the application. For example, [3] presents recommendations for choosing the nonce in CCM. Good practice is to include at least the sender's Bluetooth address and the transmission's sequence number in the nonce. In this work, the nonce size was chosen to be 96 bits to allow including sufficient amount of information in it. Thus, the block counter size is 32 bits. In addition to the nonce, in CCM there are fixed flags in the CTR input. In this paper the flags are regarded as a part of the nonce.

Since ESL is located above the standard HCI, only the data placed in the HCI data packet payloads can be encrypted. However, the application can protect the known Bluetooth packet header fields (e.g. Bluetooth addresses) with MAC. In addition, the application must ensure that the nonce can be generated at the receiving device. The generation data can be predefined or transmitted in the HCI data packets. It does not have to be kept secret, as long as it is protected with MAC.

The format of an ESL packet, placed in the HCI data payload, is presented in Fig. 3. The maximum size of the application data per ESL packet depends on the chosen MAC size. It is assumed that the complete nonce is transmitted in an ESL packet. If a combined mode is chosen, the ESL payload field is encrypted. The HCI payload is transmitted in a Bluetooth ACL network packet. If the HCI payload does not fit into a single network packet, the Bluetooth controller automatically fragments the payload into several network packets. In the figure the Bluetooth ACL packet is the ACL data packet with the largest payload size.

B. ESL Entities

As depicted in Fig. 2, ESL consists of three entities: *security processing*, *data transfer*, and *management*. ESL is accessed through ESL API. The security processing entity performs the computations related to the security

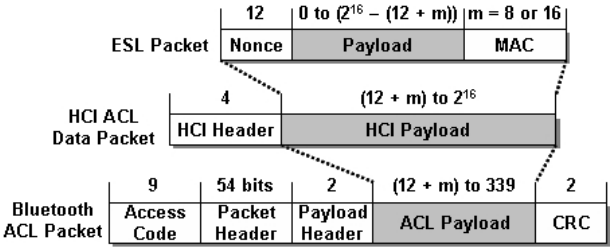


Fig. 3. ESL, HCI ACL, and Bluetooth ACL packet formats. The field sizes without units are presented in bytes.

design of Section III-A, i.e., the AES algorithm itself and the operations of the encryption modes. It appends MACs to the transmitted application data and verifies the received MACs. Failing packets are silently dropped. It also pads the input data to the block boundary and truncates the computed MACs into the desired length. The entity is not utilized if only the standard Bluetooth security is used.

The data transfer entity conveys application data between ESL API and the Bluetooth controller by constructing and decoding HCI data packets. If only the standard Bluetooth security is used, in transmission the entity places the application data into a HCI data packet payload field and forwards the HCI packet to the Bluetooth controller. Upon the reception of a HCI data packet, it decodes the application payload from the packet and gives it to ESL API. If the enhanced encryption is utilized, the payload to be transmitted in a HCI data packet is received from the security processing entity. Similarly, the payload of the received HCI data packet is first processed by the security processing entity.

The management entity controls the other entities and Bluetooth links. It initiates the Bluetooth controller, establishes and closes connections, and provides keys and other parameters (encryption mode and MAC length) to the security processing entity. The management entity constructs HCI commands and receives information from the Bluetooth controller in HCI events. The controller initiation consists of preparing the device to function as a slave or a master and, if the standard security is applied, providing the controller with the security parameters.

The management entity maintains a *connection list* which contains handles to the established Bluetooth links and their ESL parameters, i.e., the encryption and MAC keys and chosen MAC length. If the enhanced security is not utilized for a link, the management entity sets ESL to bypass the enhanced security processing. Otherwise the entity advises the security processing entity to en/decrypt the HCI data payload and, if valid, give it to the data transfer entity for further processing.

The ESL processing and HCI are hidden behind the high-level ESL API. It provides procedures for device initiation, connection establishment, sending and receiving application data, and disconnecting. After the initialization, the security processing is transparent to the application. The interface is only requested to transmit or receive application data and provide nonces. More details and the implementation of ESL API is presented in Section IV-C.

IV. Prototype Implementation

Altera Excalibur EPXA10 DDR Development Kit, presented in Fig. 4, was used as the ESL prototype implementation platform. The main component is the EPXA10F1020C2 programmable chip, which consists of an integrated ARM922T processor core and an Altera APEX20KE-like Programmable Logic Device (PLD). In the implementation, a 256-megabyte SDRAM was used as an external memory.

The PLD consists of a large number of programmable Logic Elements (LE). In addition, Embedded System Blocks (ESB) are provided for implementing a variety of memory functions. ARM9 and the PLD are connected through two Advanced Microcontroller Bus Architecture (AMBA) High-performance Bus (AHB) bridges, a shared Dual-Port RAM (DPRAM), and interrupt lines.

The daughter card of Ericsson Bluetooth Starter Kit [20] was utilized as the Bluetooth controller. It provides the host with HCI via UART or USB. The radio transmits at 1 Mb/s. The card was connected to an expansion header of the development kit and accessed via UART.

The architecture of the ESL prototype is presented in Fig. 5. The components implemented in the hardware (PLD) were Direct Memory Access (DMA), UART and its control, security processing and control , and processor-PLD communications. The security processing was implemented in hardware for high performance and low energy. The hardware design was captured in VHDL and the software in C.

A. Security Processing Hardware

The iterative, 128-bit-key AES core presented in [21] was utilized in the ESL implementation. The design computes the roundkeys on-the-fly and one encryption round at a clock cycle. It offers high throughput and does not require set-up time for switching the encryption key. Due to the feedback loop of the ESL encryption modes, only the iterative versions are reasonable choices for a single-core implementation. The on-the-fly key schedule is well-suited for the implementation since the processing is constantly altered between encryption and MAC computation with different keys, except in the MAC mode. A

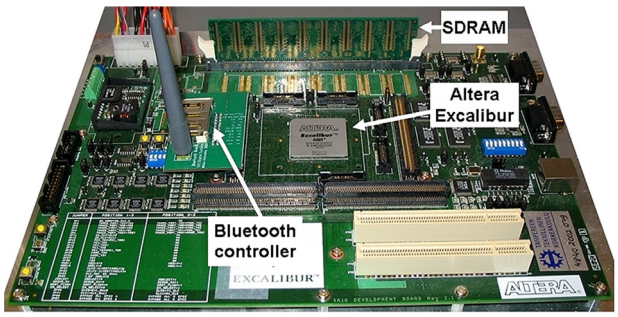


Fig. 4. ESL prototype implementation platform.

ConTEL 2005, ISBN: 953-184-081-4

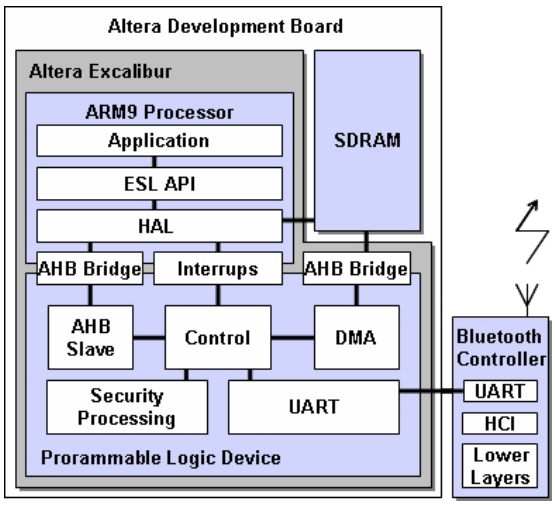


Fig. 5. ESL prototype architecture.

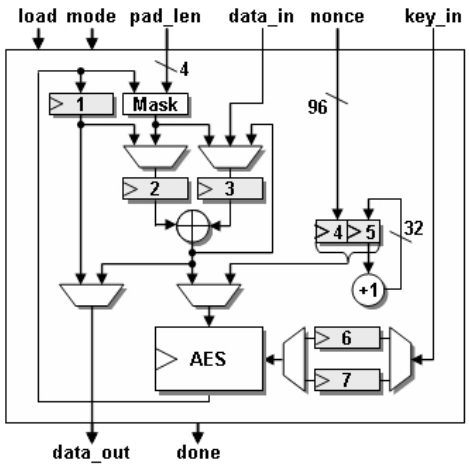


Fig. 6. Security processing hardware architecture.

precomputed schedule would require set-up latency and storage for the roundkeys.

The security processing architecture is presented in Fig. 6. Input data encoding and MAC value comparisons are performed by an external entity (i.e. ARM9). The internal signals are 128 bits wide, except the defined ones. The parameter updated internally is the block counter. The *load* signal setups the module for reading a new encryption key, MAC key, and nonce values from the *key_in* and *nonce* ports. The keys are stored in Registers 6 and 7, nonce in Register 4. The nonce is concatenated with the block counter in Register 5. The *load* port allows also maintaining the old key values and updating only the nonce. The block counter is reset by updating the encryption key or nonce.

Feeding a new data block is done through the *data_in* port. The block is stored in Register 3. The signal *mode* defines whether the module operates in the MAC, MAC-then-encrypt, encrypt-then-MAC, or CCM mode and sets the module to encrypt or decrypt. The *pad_len* signal is used for informing the number of padding bytes in the last input block. This information is required in the combined modes. The *data_out* port is used for outputting the en/decrypted data blocks as well as the MAC values.

In the MAC mode MAC is computed using the MAC key. The chaining value in Register 1, obtained after processing the previous data block, is transferred to Register 2. Initially, the value is zero. After XORing the chaining value with the data block in Register 3, the result is processed by the AES core and the output is written back to Register 1. After the last data block the contents of Register 1 is output and the register is reset. The MAC verification are carried out in the same way.

In the MAC-then-encrypt mode, first, the design performs the MAC computation for a block with the MAC key as described. Then, the operation is switched to the CTR encryption with the encryption key. In CTR the nonce and the block counter are fed to the AES core. Initially, the block counter is set to zero. The result is written through the *mask* block to Register 2 and XORed with the data block maintained in Register 3. Finally, the result is output and the block counter is incremented. After the last data block, the nonce and the block counter are processed once more and the result is written to Register 3 (through the *mask* entity). The contents of Register 1 are transferred to Register 2 and XORed with Register 3. The result is output as the encrypted MAC.

In the encrypt-then-MAC mode the processing order of MAC-then-encrypt is inverted. The only difference is that after encrypting a data block, the output is also written back to Register 3 for the MAC computation.

In the MAC-then-encrypt decryption the processing is mainly the same as in the encrypt-then-MAC encryption. After the last data block the received, encrypted MAC is input through *data_in* to Register 3. The nonce and the block counter are input to the AES core. XORing the AES output with Register 3 yields the decrypted MAC, which is output for comparison with the earlier output, computed MAC. The encrypt-then-MAC decryption processing is the same as the MAC-then-encrypt encryption processing. The received MAC is not input since it is already in the plaintext form.

The CCM mode operation is similar to the MAC-then-encrypt mode. The encryption and MAC keys are set to the same value. The difference is that in the CCM mode the block counter starts initially from one and the MAC value is encrypted/decrypted with the block counter value zero.

If the data length is not a multiple of 16 bytes, the last output en/decrypted data block has to be truncated to the original length. This implies the need for the *mask* entity in the encrypt-then-MAC encryption, MAC-then-encrypt decryption, and CCM decryption. Before XORing the last key stream block with the data block, the bytes of the key stream block corresponding to the extra bytes (zeroes) of the data block have to be masked to zero. This way the XOR result of the input block and the last key stream block, which is used as the input in the MAC computation, has the correct padding (zeroes). The mask logic is implemented with a ROM, containing 16 masking entries of size 16 bytes, and an AND gate.

B. Communications

As shown in Fig. 5, the external SDRAM is used for data transfers between ARM9 and PLD in the implementation. The external memory is larger than the fixed-size DPRAM and it can be switched, which makes the ESL implementation scalable for processing larger amounts of data. By sharing SDRAM, the processor does not have to transfer the data from the data memory in SDRAM to DPRAM for the PLD usage. Instead, PLD can access SDRAM directly, which is faster and lets ARM9 perform other tasks concurrently. The DMA entity was implemented for the purpose in PLD. It accesses the memory via an AHB bridge. An UART entity was implemented in PLD for transferring data between the development board and the Bluetooth controller.

The data transferred through SDRAM consists of HCI commands and data to the Bluetooth controller, HCI events and data from the controller, and the data to/from the security processing entity. The nonce, encryption and MAC keys, and the UART initialization data are also conveyed through the memory. The nonce and keys are only transmitted to the security processing entity when the values are initialized or changed.

After writing a HCI command or data to SDRAM, ARM9 utilizes the other AHB bridge for initiating operations in PLD. The control entity receives the processor requests through the AHB slave in Fig. 5. The slave contains logic for interfacing the AHB bus as well as control and memory address registers to which ARM9 requests are written from the bus. Depending on the request, the control entity begins an UART transmission of a HCI command or the en/decryption of data. ARM9 is interrupted after an operation is finished. The processor reads the reason for the interrupt from the AHB slave.

When a HCI data packet is received from the Bluetooth controller it is written to SDRAM by DMA and ARM9 is interrupted. If the packet payload is not encrypted, the processor decodes the packet and gives the ESL payload to the application. Otherwise it provides the security processing entity with the keys and nonce for decrypting the payload. When the payload is decrypted ARM9 is again interrupted. If a MAC scheme is used, the processor verifies that the received and computed MAC values match and conveys the data to the application.

Each HCI command has a corresponding event (acknowledgement) with which the Bluetooth controller replies to the command. In addition, the network operations trigger events. For simplicity and removing unnecessary memory accesses, the PLD control entity filters out the events uninteresting to the processor.

The communications between Bluetooth devices utilize the ACL link. The used ACL packet types can be defined with a HCI command.

C. Software Interfaces

In Fig. 5 Hardware Abstraction Layer (HAL) implements the ESL functionalities on the ARM9 side. It constructs HCI commands and data to and reads HCI events from SDRAM as well as decodes payload data from the HCI data packets. It also handles the interrupts initiated by PLD. HAL controls the security processing entity and UART by modifying the memory mapped control and address registers in the AHB slave. HAL allows utilizing the Bluetooth's own security features as well as choosing among the enhanced security implementations. If an enhanced security mode is utilized, upon the reception of a HCI data packet, HAL compares the MAC values. It also pads the input data and truncates MACs.

The ESL API implementation provides procedures for initiation, connection management, and sending and receiving application data. A pseudo-code example of the API usage is presented in Fig. 7.

In the example, first, the Bluetooth device is initialized. It is defined that the standard security features are not used. The *InitBd* procedure returns the unique Bluetooth address of the controller and the maximum size of the payload for a single transmission. After the initialization the device is able to operate in the slave mode. Next, the role of the device is switched to the master mode. The *PrepareMaster* procedure also scans for devices in the range and returns a list of found device addresses. Before connecting to a device the parameters for the enhanced security features are set. Successful connection creation returns the handle of the created link. Application data can be sent over the link with a single procedure call. The data size must respect the maximum payload size defined in the initialization. Finally, the connection is closed.

Even though not utilized in the example, each procedure also returns a boolean value that informs whether the performed operation was successful or not. For example, if the transmit buffer of the Bluetooth controller is full, the *TransmitData* procedure fails. Changing the security parameters requires closing the link and calling the *InitBd* and/or *SetEncMode* procedures again. The

```
void main() {
  ...
  // initialize Bluetooth device:
  // authentication disabled, no link key type
  // defined, no PIN input
  InitBd(FALSE, NULL, NULL, OwnBdAddress,
       payloadMaxSize);

  // set device into master and scan devices nearby
  PrepareMaster(numberOfResponses, deviceArray);

  // set values for Bluetooth's own link key and
  // for the keys of the enhanced encryption and
  // data authentication, choose encrypt-then-MAC
  SetEncMode(NULL, encKey, macKey, ENC_THEN_MAC);

  // create connection to the first found device
  // with the security parameters above
  ConnectToBd(deviceArray[0], connectionHandle);

  // send data to the connected device
  TransmitData(connectionHandle, payloadSize,
       payload, nonce);

  // close connection
  Disconnect(connectionHandle);
}
```

Fig. 7. A pseudo-code example of using the ESL API implementation. Values for the parameters in italics are returned by the procedure calls.

ConTEL 2005, ISBN: 953-184-081-4

application must implement a separate callback procedure for receiving data. Upon the reception of a data packet, the procedure is automatically called by the ESL API implementation. Similarly to the standard Bluetooth, the implemented software enables up to seven simultaneously active Bluetooth connections.

D. Results and Comparison

The hardware entities were tested in VHDL simulation with ModelSim SE PLUS 5.8d 2004.06. The security processing entity was separately verified against Brian Gladman's AES software library [22]. The hardware netlist was generated with Precision RTL Synthesis 2003b.41 and the netlist was synthesized with Quartus II v4.1. The software was compiled with ARM Developer Suite 1.2. The complete ESL implementation was tested in practice with a test application between two development boards. The hardware synthesis results on the PLD of EPXA10F1020C2 are presented in Table I.

The throughput of the security processing entity is 214 Mb/s (26 cycles per block) in the MAC-then-encrypt, encrypt-then-MAC, and CCM modes and 507 Mb/s (11 cycles per block) in the MAC mode. Compared to the maximum Bluetooth transmission speeds (Section II), negligible latencies are implied by the added processing.

The maximum size of the data payload that can be transmitted/received in a HCI ACL data packet is 2^{16} bytes [1]. However, the buffers of the Ericsson Bluetooth controller support only 672-byte HCI payload. Regarding this, using 128-bit MACs, and assuming that the 96-bit nonce is transmitted in the payload and that the controller transmits at the maximum payload speed of 723.2 kb/s, it can be computed that the maximum application data throughput of the ESL implementation is 693 kb/s. In addition to the buffers, the total throughput is further limited by the fixed UART implementation of the Ericsson Bluetooth controller. Its highest speed is 460 kb/s.

Table II compares the hardware part of the ESL prototype with the hardware implementation of the standard Bluetooth security [4]. In addition, the table presents comparisons between the cryptographic cores of the Bluetooth design (E_0 and SAFER+) and ESL (AES). In order to evaluate the AES core of this paper, the table includes measures for two other programmable logic designs. According to the authors' knowledge, [23] presents the most compact and [24] the fastest iterative AES design suited for feedback encryption modes.

All the reference implementations were made on Xilinx Field Programmable Gate Arrays (FPGA), which differ from the Altera PLDs. However, the basic building blocks of the FPGAs, Logic Cells (LC), are very similar to the Altera LEs. Each programmable FPGA component, *slice*, contains two LCs. Thus, a fairly precise estimate for the number of LEs was obtained for the reference implementations by doubling the number of slices.

Compared to the Bluetooth security design [4], considerably lower LE consumption and higher encryption throughput were achieved with the ESL prototype. However, part of the ESL control is implemented in the ARM9, which slightly decreases the occupied PLD resources. On the other hand, the ESL implementation includes the ARM9 and Bluetooth controller interfaces. The number of memory bits was increased but this can be significantly reduced by modifying the AES core according to the AES reference implementations. Compared to [23] and [24], the core is an average implementation with reasonable resources and throughput.

Instead of using E_0 for encryption and SAFER+ for authentication and key generation, in ESL all three procedures can utilize AES. Table II shows that this reduces the hardware resources and also implies shorter latencies due to the higher performance.

V. Conclusions & Future Work

In this paper the design and implementation of a new Enhanced Security Layer (ESL) for Bluetooth was presented. ESL significantly improves the Bluetooth security by replacing the proprietary encryption with an AES-based design and adding cryptographic message integrity verification. In addition to the security enhancements, an easy-to-use API was designed. The API implementation offers an application developer simple access to the wireless link and transparent security processing. The throughput of the implemented security processing hardware is 214 Mb/s with the modes offering the highest security level. This implies only a negligible processing latency, which is also lower than that of the standard security design.

TABLE I

ESL HARDWARE SYNTHESIS RESULTS ON EPXA10F1020C2

Entity	Logic elements	Memory bits
AHB slave	440	0
Control	2,170	0
DMA	894	0
Security processing	3,527	43,008
UART	208	0
Others	5	0
Total	7,244	43,008
(% of the chip's maximum)	(18%)	(13%)
Maximum clock		43.55 MHz

TABLE II

COMPARISON OF HARDWARE IMPLEMENTATIONS

Security design implementations				
Design	**LEs**	**Mem.**	**Clock**	**Throughput**
Bluetooth [4]	39,810	38,272	15 MHz	15 Mb/s
ESL (this paper)	7,244	43,008	43 MHz	214 Mb/s
Cryptographic cores				
Design	**LEs**	**Mem.**	**Clock**	**Throughput**
E_0 [4]	1,790	0	15 MHz	15 Mb/s
SAFER+ [4]	8,116	6,272	20 MHz	320 Mb/s
AES [23]	292	35,584	123 MHz	358 Mb/s
AES [24]	4,514	0	127 MHz	1,563 Mb/s
AES (this paper)	1,246	40,960	43 MHz	507 Mb/s

Currently, ESL offers means for the enhanced security processing. Assuming that devices have earlier reliably authenticated each other and shared keys for encryption and MAC computation, applying ESL makes the communications secure. ESL is further developed and the support for authentication, key exchange, freshness protection, nonce management, and input data encoding for the new encryption modes are added. For example, in order to be compatible and secure, the CCM mode encoding will follow its specification [3]. Similarly, the CBC-MAC mode is extended to automatically add protocol fields (e.g. the length of the data) required to be secure [16].

ESL is also developed to prevent the known attacks on the Bluetooth's own security by allowing only safe combinations for its parameters. A key exchange protocol will be integrated to ESL for automatic PIN exchange. It will only allow 128-bit encryption keys and prohibit the usage of the default values. Moreover, to improve the prototype implementation, the bus between the development board and the controller will be switched from UART to USB.

REFERENCES

[1] *Specification of the Bluetooth System*, Bluetooth Special Interest Group (SIG) Std. 2.0 + EDR, 2004.

[2] *Advanced Encryption Standard (AES)*, Federal Information Processing Standards Std. FIPS-197, 2001.

[3] D. Whiting, R. Housley, and N. Ferguson. (2003, Jan.) Counter with CBC-MAC (CCM) - AES mode of operation. [Online]. Available: http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/ccm/ccm.pdf

[4] P. Kitsos, N. Sklavos, K. Papadomanolakis, and O. Koufopavlou, "Hardware implementation of Bluetooth security," *IEEE Pervasive Computing*, vol. 2, no. 1, pp. 21–29, 2003.

[5] M. Jakobsson and S. Wetzel, "Security weaknesses in Bluetooth," in *Proc. Cryptographer's Track at RSA Conf. 2001 (CT-RSA 2001)*, San Francisco, California, USA, Apr. 2001, pp. 176–191.

[6] A. Levi, E. Cetintas, M. Aydos, C. K. Koc, and M. U. Caglayan, "Relay attacks on Bluetooth authentication and solutions," in *Proc. 19th Int. Symp. on Computer and Information Science (ISCIS 2004)*, Antalya, Turkey, Oct. 2004, pp. 278–288.

[7] D. Kügler, "Man in the middle attacks on Bluetooth," in *Proc. 7th Int. Financial Cryptography Conf. (FC'03)*, Gosier, Guadeloupe, French West Indies, Jan. 2003, pp. 149–161.

[8] L. Nguyen, R. Safavi-Naini, W. Susilo, and T. Wysocki, "Secure authorization, access control and data integrity in Bluetooth," in *Proc. 10th IEEE Int. Conf. on Networks (ICON 2002)*, Singapore, Aug. 2002, pp. 428–433.

[9] M. Hermelin and K. Nyberg, "Correlation properties of the Bluetooth combiner," in *Proc. Int. Conf. on Information Security and Cryptology (ICISC'99)*, Seoul, Korea, Dec. 2000, pp. 17–29.

[10] S. Fluhrer and S. Lucks, "Analysis of the E0 encryption system," in *Proc. 8th Annu. Int. Workshop on Selected Areas in Cryptography (SAC 2001)*, Toronto, Canada, Aug. 2001, pp. 38–48.

[11] J. D. Golic, V. Bagini, and G. Morgari, "Linear cryptanalysis of Bluetooth stream cipher," in *Proc. EUROCRYPT 2002*, Amsterdam, The Netherlands, Apr. 2002, pp. 238–255.

[12] Y. Lu and S. Vaudenay, "Faster correlation attack on Bluetooth keystream generator E0," in *Proc. 24th Annu. Int. Cryptology Conf. (CRYPTO 2004)*, Santa Barbara, California, USA, Aug. 2004, pp. 407–425.

[13] K. Ritvanen and K. Nyberg, "Upgrade of Bluetooth encryption and key replay attack," in *Proc. 9th Nordic Workshop on Secure IT Systems (Nordsec2004)*, Espoo, Finland, Nov. 2004.

[14] H. Lipmaa, P. Rogaway, and D. Wagner. (2000, Oct.) CTR-mode encryption. [Online]. Available: http://csrc.nist.gov/CryptoToolkit/modes/workshop1/papers/lipmaa-ctr.pdf

[15] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. (2000, Sept.) A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation. [Online]. Available: http://www.cs.ucsd.edu/users/mihir/papers/sym-enc.html

[16] M. Bellare, J. Killian, and P. Rogaway, "The security of the cipher block chaining message authentication code," *Journal of Computer and System Sciences*, vol. 61, no. 3, pp. 362–399, 2000.

[17] H. Krawczyk, "The order of encryption and authentication for protecting communications (or: How secure is SSL?)," in *Proc. 21st Annu. Int. Cryptology Conf. on Advances in Cryptology (CRYPTO 2001)*, Santa Barbara, California, USA, Aug. 2001, pp. 310–331.

[18] N. Ferguson and B. Schneier, *Practical Cryptography*. Indianapolis, Indiana, USA: Wiley Publishing, Inc., 2003.

[19] J. Jonsson, "On the security of CTR + CBC-MAC," in *Proc. 9th Annu. Int. Workshop on Selected Areas in Cryptography (SAC 2002)*, St. John's, Newfoundland, Canada, Aug. 2002, pp. 76–93.

[20] *BSK Technical Documentation*, BSK/S10311/1.2, Ericsson Microelectronics AB, 2001.

[21] P. Hämäläinen, M. Hännikäinen, M. Niemi, T. D. Hämäläinen, and J. Saarinen, "Implementation of link security for wireless local area networks," in *Proc. IEEE Int. Conf. on Telecommunications (ICT 2001)*, vol. 1, Bucharest, Romania, June 2001, pp. 299–305.

[22] (2004) Brian Gladman's AES code. [Online]. Available: http://fp.gladman.plus.com/AES

[23] G. Rouvroy and F.-X. Standaert, "Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications," in *Proc. IEEE Int. Conf. Information Technology: Coding and Computing (ITCC 2004)*, vol. 2, Las Vegas, NV, USA, Apr. 4–6, 2004, pp. 583–587.

[24] F.-X. Standaert, G. Rouvroy, J.-J. Quisquarter, and J.-D. Legat, "A methodology to implement block ciphers in reconfigurable hardware and its application to fast and compact AES Rijndael," in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays (FPGA 2003)*, Monterey, CA, USA, Feb. 23–25, 2003, pp. 216–224.

PUBLICATION 10

P. Hämäläinen, M. Hännikäinen, T. D. Hämäläinen, R. Soininen, and R. Rautee, "Design and Implementation of Real-time Betting System with Offline Terminals," *Elsevier Electronic Commerce Research and Applications*, vol. 5, no. 2, pp. 170–188, 2006.

Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Commerce Research and Applications 5 (2006) 170–188
Electronic Commerce Research and Applications
www.elsevier.com/locate/ecra

Design and implementation of real-time betting system with offline terminals

Panu Hämäläinen [a,*], Marko Hännikäinen [a], Timo D. Hämäläinen [a], Riku Soininen [b], Risto Rautee [b]

[a] Tampere University of Technology, Institute of Digital and Computer Systems, P. O. Box 553, FI-33101 Tampere, Finland
[b] Oy Veikkaus Ab, FI-01009 Veikkaus, Finland

Received 29 March 2005; received in revised form 12 September 2005; accepted 20 December 2005
Available online 20 March 2006

Abstract

Even though enabling electronic betting as an expanding entertainment field, the advanced communication and e-commerce technologies have not changed the nature of betting itself. This paper presents the novel offline Real-Time Betting (RTB) system overcoming the online processing and scalability limitations. It changes betting services into interactive sessions by supporting short, unpredictable target incidents as well as crediting winnings in real-time. The security design specifies reliable methods for time-stamping, storing bets, and verifying the authenticity of operations. A prototype system has been implemented for WLAN and DVB environments. Its performance and usability are evaluated and compared to traditional online systems.

Keywords: Real-time betting; Electronic betting; Offline; Security; Synchronization; Interactive systems; Entertainment; Wireless communications

1. Introduction

Advanced mobile terminals, multimedia networks, as well as e-commerce technologies have enabled numerous new applications for business and pleasure. Among others, betting organizers have begun offering their services through electronic systems, especially on the Internet. Instead of placing bets in front of a retail operator at a specific location, customers can attend betting remotely – virtually anywhere – with their own PCs, laptops, mobile phones, and set-top-box devices. The setup time and effort have markedly decreased and bets can be placed nearer the beginning of an event. In addition, winnings are credited online.

While the service has become more convenient for the customers, the fundamental nature of betting itself has not changed. The targets in electronic systems are still the same as in manual systems, and often provided in both ways. 'Will this penalty kick result in a goal' is an example of an unsuitable incident in a soccer match. There is not enough time or processing resources in user terminals, networks, and servers for handling such a bet. In order to change also the characteristics of betting from a static, slow procedure into an entertaining, interactive session that lasts throughout the target event, research on Real-Time Betting (RTB) has been carried out at Tampere University of Technology, Finland. The research has especially focused on providing betting with new characteristics over wireless user terminals, such as mobile phones, Personal Digital Assistants (PDA), and set-top-boxes.

Opposed to existing online systems, the developed system, referred to as *offline RTB*, does not require uplink communication from users to the organizer during the betting session. Instead, the user terminals only receive

* Corresponding author. Tel.: +358 3 3115 11; fax: +358 3 3115 4561.
E-mail addresses: panu.hamalainen@tut.fi (P. Hämäläinen), marko.hannikainen@tut.fi (M. Hännikäinen), timo.d.hamalainen@tut.fi (T.D. Hämäläinen), riku.soininen@veikkaus.fi (R. Soininen), risto.rautee@veikkaus.fi (R. Rautee).

1567-4223/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.elerap.2005.12.003

broadcast transmission from the organizer server and locally time-stamp and store the placed bets. New reliable methods are required for ensuring the authenticity of the operations during the session, especially at the user devices. The solutions presented in this paper allow implementing the user terminals as dedicated betting devices. Alternatively, the support for the offline RTB can be added to the existing wireless consumer devices, e.g., as a standard peripheral card.

Offline RTB achieves two fundamental goals for enabling interactive betting as a large-scale e-commerce application. First, it supports frequent bets on quick incidents and gives instant feedback when the incident outcomes are known. Second, the offline architecture is scalable, as it does not limit the number of betting terminals attending a session, while supporting the fast-phased betting. Offline RTB places only reasonable processing requirements for the system components.

The paper is organized as follows. In Section 2 the related work in the field is introduced. The concept of RTB and the requirements for a RTB system are defined in Section 3. Section 4 presents simulation results showing that current online methods with the existing user terminals cannot fulfill the requirements placed on RTB. Section 5 introduces the novel offline RTB architecture. Section 6 describes the offline RTB user terminal design. Suitable terminal and network technologies for an offline RTB implementation are discussed in Section 7. A prototype implementation for Wireless Local Area Network (WLAN) and Digital Video Broadcasting (DVB) environment with PDAs, PCs, laptops, and set-top box devices is presented in Section 8. The section also includes the results of real end-user experiments. Finally, Section 9 concludes the paper.

2. Related work

In traditional betting systems users place their bets in front of a retail operator. It is easy to take care of timing aspects: the betting counter is closed before the target event begins. The winnings are paid out after the event is finished. The practice for electronic betting is depicted in Fig. 1. There are numerous companies providing these so-called online services on the Internet. The betting terminal is a PC or a mobile phone. Also, dedicated devices operated by an authorized betting agent are used. The main server and duplicated bet databases are located in the organizer premises and a supervising third party is included for controlling the authenticity of operations.

In an online system users place bets with terminals, which transmit them to the main server. At the closing time the server stops accepting bets. When the outcome of the incident is clear, users may collect their winnings. Traditionally, the money reserved for a bet is paid to the organizer after choosing a bet target and winnings are paid out to a person presenting a winning voucher. Users may also have dedicated betting accounts for deducting stakes and crediting winnings.

In traditional online systems bet placements have to be completed early, before the target event begins. Bets are usually placed on the result of a complete match, race, or period. In addition, the availability of a certain bet incident with its exact opening and closing times is accurately defined in advance. A user placing a bet easily looses her excitement if incident outcomes become obvious already at the beginning of the event. The user must decide before the event starts whether or not to place bets, on which targets, and how much money to invest. If an incident is very short ('will team Finland win the opening face off'), it probably has a small winning ratio. Thus, users choose not to participate.

2.1. New commercial systems

At the end of year 2002, Fintoto in Finland released a tote service on the Internet. It enables betting on horses online, starting from few days before until the start of a heat. The betting server is prepared for the high peaks during the last few minutes of an open bet. It is capable of processing the maximum of hundred requests per second (10 ms per request). This setup is claimed to be sufficient for five years. Even though state-of-the-art technology is utilized, the system still provides traditional betting for a result of a heat [1].

Global Interactive Gaming (GIG) [2] has developed a real-time software betting system. The system enables betting on discrete parts of an event. However, it is also an online system, in which bets are collected before incidents

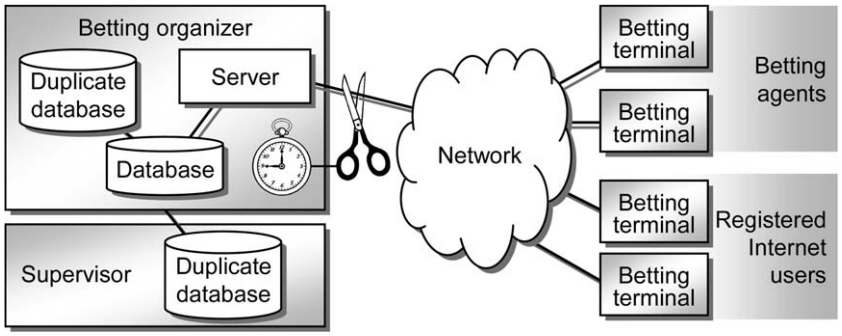


Fig. 1. General organization of electronic betting systems.

are settled. The main novelties are that the bet announcements are broadcast to terminals and the system constantly recalculates odds according to betting activity.

Oy Veikkaus Ab, a Finnish lottery company, has been developing online real-time betting service based on GSM Short Message Service (SMS). Several experiments have been carried out in 2003 and 2004. Short bets were introduced during ice-hockey and soccer matches. The tests have been extremely popular and the limit of a thousand volunteers has been reached instantly.

As the number of users increases and betting intervals shorten, the described systems require more processing resources (network throughput and server processing capacity) for maintaining the quality of service. Hence, they are not easily scalable.

2.2. Academic research

In addition to the commercial systems, academic research on the design of electronic betting and gambling has been carried out. The research has focused on making games and monetary transactions fair, secure, and verifiable for both users and organizers, not on real-time capabilities and scalability.

Protocols for electronic lotteries and casinos are described in [3]. The designs remove the requirement to trust the organizer without resorting to another trusted party. Instead, users are able to verify the correctness of execution. Similarly, [4] presents methods for users and organizers to prove and verify the validity of actions. In [5], a fair and verifiable gambling scheme is developed for devices with limited processing capabilities, such as mobile phones.

In [6] a gambling system consisting of an untrusted device and a smart card is defined. It can be used for gambling on randomized events (e.g. card games) without communicating with an organizer. The system does not support betting on time-limited events or on events the outcomes of which are defined by an external source, not by the smart card.

Zhao et al. [7] propose a gambling scheme for the Internet. The research concentrates on ensuring the fairness of games and payments and uses a trusted third party for resolving disputes. Sako [8] presents a server with which groups of users can create, participate, and verify their own lotteries. Similar lottery systems, in which the participants blindly influence on the values of the randomly-generated winning numbers, are developed in [9, 10, and 11].

While making the systems secure and verifiable, the solutions are not easily scalable since they still require two-way communications before the target incident is resolved.

3. Real-time betting

RTB is defined as an activity in which bet targets are continuously and frequently announced during an event and users try to predict the outcomes. After the outcome of an incident is known, users receive their winnings and invest them on new incidents. *Real-time* refers to that there is no limit for the minimum length of a bet, noticeable to users. A bet may be open only a second and the system is still able to handle it.

The components of a RTB system are depicted in Fig. 2. An *event* is a target happening which contains discrete *incidents* from which bet targets are chosen. An *operator* is the entity which follows the event, announces bets, and defines results. The announcements are transferred to a betting server via a secure channel. The server processes and stores bet and user information. It informs users about the announcements through another secure channel. A user follows the event and places bets with a betting terminal.

In RTB, a suitable incident must be a distinctly separable occurrence during an event. It must have an unarguable opening time, closing time, and discrete outcome. When the event itself is well predictable and sequential, bets are easier to define. For example, long jumping is an event having these characteristics. On the other hand, incidents in a soccer match are more complicated to distinguish. Commonly repeating bet types can be prepared as *templates* in advance, which makes it easier for an operator to react when potential incidents occur.

A key requirement in RTB is fairness. Each user should get her bets placed independently of the geographical location within the service coverage area. In addition, it should be possible to place a bet just before the closing time when most information on the probable outcome is available. How near the closing time the decision is made, should not significantly affect the bet acceptance probability. Fulfilling the fairness requirement is important for creating the excitement of RTB.

When the number of users is relatively small, existing online systems can be used for implementing limited RTB. The maximum number of users is set by server processing capacity and network throughput. Also, these define how short bets are possible. Fairness requires preparing for high peak traffic. Particularly, the service cannot meet the fairness requirement on the Internet since users with better connections are favored.

An advanced implementation consists of distributed, trusted time-stamping devices [12]. They communicate with the terminals appending placements with unchangeable timing information. At a suitable time, the betting server

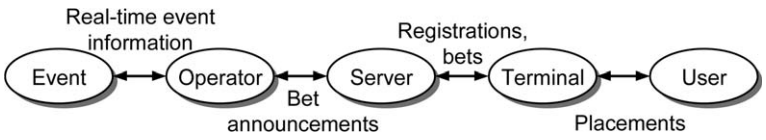


Fig. 2. Components of RTB system.

retrieves the data and computes the results. The main drawback for RTB is that, for balancing the processing load, the distribution of the devices has to be well-designed and the location of terminals known. For example, the devices by Symmetricom [13] can produce 50–125 protected time stamps per second, implying a large number of devices for large-scale RTB.

4. Online performance for RTB

In order to evaluate online system performance for RTB, a simulation model was developed in Specification and Description Language (SDL) [14]. SDL is a high-level description language for implementing entities on different abstraction levels with parallel extended finite state machines. The entities can be connected to form a single system model for design verifications and simulations. The design tools enable generating application (C code) from the SDL descriptions.

For allowing the free movement of users, wireless networks are preferred in providing RTB services. Being one of the wireless technologies with the highest capacity and the lowest delay, IEEE 802.11b WLAN [15] was chosen for the connection in the model. Several stadiums in Finland already contain WLAN access points with Ethernet backbone network. The medium access control protocol was implemented in the Distributed Coordinate Function (DCF) [15] mode.

A bet was placed by sending a 100-byte packet from a terminal to the server. If the transmission was successful, the server responded with a 100-byte acknowledgement after processing the request. A bet was regarded valid only if the server managed to acknowledge it before the closing time. In order to simulate real wireless environment with errors bursts, a discrete two-state Markov model was added to the channel [16]. The probability of transitioning from the good state to bad was 5% and 30% vice versa, taken from [17]. A transmission was successful in the good state and failed in the bad state.

Fig. 3 shows the results for two different simulations. 500 terminals started transmitting placements simultaneously at the beginning of Simulation 1. The WLAN access protocol ensured that the channel did not become jammed. The server processing time was 11.7 ms per bet on average, which is in line with the Fintoto system [1]. It was estimated to include all the network and processing delays outside the WLAN. Despite the heavy access contention and rather pessimistic error model, all the terminals were able to transmit within 0.57 s. However, the server became the bottleneck. Its processing queue started decreasing only after the terminals had finished transmitting. It took 5.9 s before the server had finished processing. Because of the growing queue, the later a terminal managed to transmit its bet, the longer it had to wait for the response. The last terminal had to wait for 5.3 s.

The bet placements of the 500 terminals were randomized in Simulation 2. The simulation was divided into small time intervals, during which a new terminal started attempting to transmit with 10% probability. The server processing time was 12.6 ms per bet. The other parameters were unchanged. The terminal transmissions were finished in 3.6 s. The maximum size of the server queue was lower

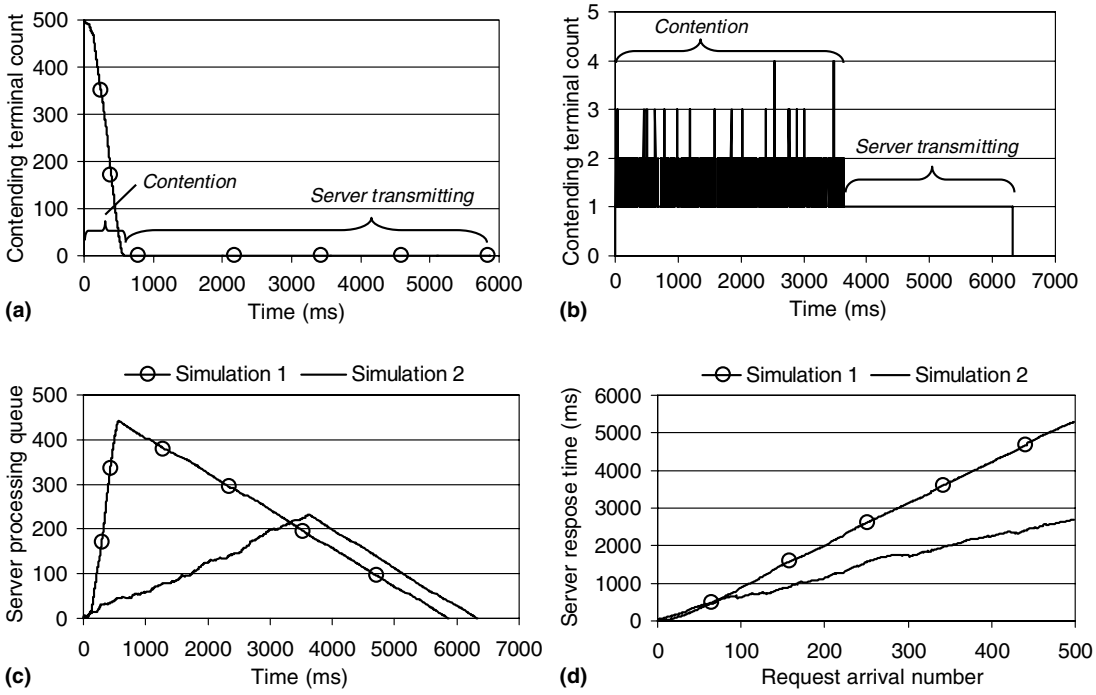


Fig. 3. Online betting simulations in WLAN: (a) the number of contending terminals as a function of time in Simulation 1 and (b) in Simulation 2, (c) server processing queue lengths as functions of time, (d) server response times as functions of the arrival number of a terminal request.

but, due to the randomization, the total processing time was about 0.5 s longer. Similarly to Simulation 1, the response time gradually increased as more bets arrived. At some points the server was able to process the requests faster than new ones arrived. Hence, the queue and response time did not grow as linearly as in Simulation 1. The maximum response delay was 2.7 s.

The WLAN simulations show that the most significant part in the response delay comes from the processing time. The terminals were able to transmit much faster than the server processed, even when all the 500 terminals started competing of the channel at the same time. The simulations illustrate that an online system with the used setup is not capable of providing RTB even for a reasonably small number of players. For example, if a bet was open for 5 s, the placements from the last 67 terminals in Simulation 1 and the last 112 terminals in Simulation 2 would be rejected. The tests with real audience in Section 8 will show that this is in fact the case: a decision to place a bet is often made during the final 10 s even if the bet is open for 30 s.

In Simulation 2, the terminals which placed their bets last did not get them through. As an opposite, in Simulation 1 the success was randomized by the WLAN contention. Thus, the later a bet is placed at a terminal or the more WLAN terminals are competing for the channel access, the higher the rejection probability is. These remarks on an online system break the fairness and real-time requirements earlier set for RTB.

The simulations did not take into account possible retransmission due to acknowledgement timer expirations. Because most terminals wait several seconds to receive a response, it is likely that their timers expire several times resulting in unnecessary traffic. This would significantly increase the network load as well as keep the server processing queue growing. By decreasing the processing time it would be possible to decrease the response time. However, increasing terminal count would eventually cancel the gain of increased processing power. Consequently, the online solution is not easily and inexpensively scalable.

For comparison with other common wireless networks, GSM and GPRS data transfer delays were measured. The connection was created through a modem pool to a destination computer in the same LAN as the pool. In the test a 100-byte packet was transmitted 1000 times. The average delay was 999 ms for GSM and 742 ms for GPRS. The delay in a WLAN via an access point was only 3 ms on average.

5. Offline architecture for RTB

In order to overcome the real-time processing problems of online betting systems, the novel *offline RTB* system has been developed. The key idea is to time-stamp and store bet records locally at terminals instead of transmitting them to a server. The server collects the records after the target event is finished and incident outcomes already revealed. The offline RTB avoids high processing peaks and supports scalable, fair electronic betting in short time cycles regardless of the number of participants and betting frequency.

The architecture is illustrated in Fig. 4. The data and broadcast networks can be separate or the same. The two-way data network is utilized for user registrations and bet record collections whereas the broadcast transmissions convey bet announcements. The term *offline* refers to the bet placement phase in which terminals only receive broadcast downlink data and locally store bet records generated from user inputs.

Participating betting in the offline RTB comprises of three stages: *registration*, *betting session*, and *bet record collection*. This section presents the general principles of the stages. Section 6 describes the security and timing related processing at a terminal in more detail.

5.1. Security challenges

The offline solution adds new constraints especially for terminal security and timing. Reliable methods are required for ensuring that accepted placements are made before outcomes are known and that they have not been changed afterwards. Due to the application, the operation environment of a terminal is considered hostile. It is required that the most critical parts of security and timing

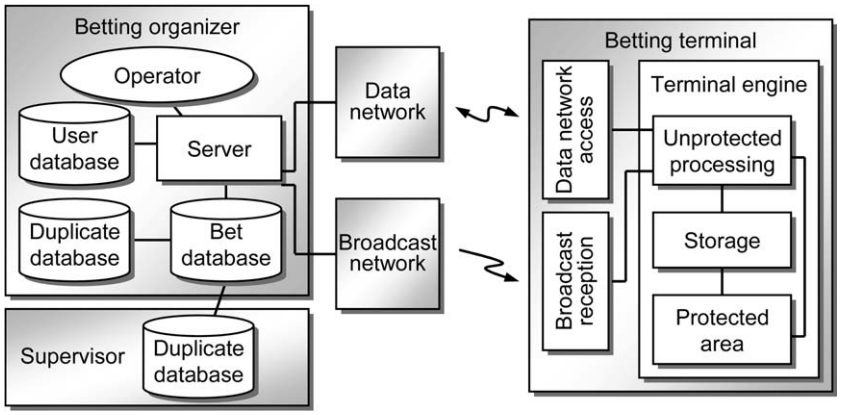


Fig. 4. Offline RTB architecture.

operations are protected from external parties, including users. Thus, the RTB terminal includes a *protected area* as depicted in Fig. 4. Only an authentic server can access it through the data and broadcast links. Unauthorized access must require much more effort and expenses than it can pay off.

Local time-stamping requires that the terminal time is synchronized to the server time and that the synchronization is maintained throughout a betting session. The synchronization has to be protected from the intervention of external parties. Also, since bets are opened and closed via the broadcast link, a terminal has to receive the broadcast in time. A user should not be able to postpone the closing time of a bet or modify the terminal time. These would allow betting on an incident after its outcome is already known. The server and/or the terminal must be able to recognize if the synchronization or the timing of the broadcast is tampered with. On the other hand, reasonably fixed, standard network delays should not prevent operation.

In order to guarantee that only an authorized user has access to the betting account and can attend betting, the user has to be reliable identified at the server during the registration. In addition, the two-way data link must be protected from outsiders. Similarly, it must be possible to verify the origin of the broadcast to disable packets from a fraudulent third party. A user should not be able to falsely deny placing bets.

5.2. Registration

Registration is possible before an event begins and also any time during the event. The two-way communication of the registration is depicted in Fig. 5.

In order to create a connection, the user is authenticated with the server (*user authentication*). After the authentication the user gets access to her account and the lists of upcoming events. Event information includes an event identifier, description, start time, types of target incidents, and a limit for acceptable stakes. Once an event has been chosen, the terminal transmits the server a *registration request*. The request consists of the event identifier and the amount of money to be reserved. The monetary transactions are carried out as in online systems. The reserved sum is deducted from the account and stored in the terminal with the rest of the registration information.

To guarantee valid operation during the betting session, the protected area and the server have to agree on certain parameters secretly from the user. Upon the reception of the registration request, the server initiates an authentication procedure with the protected area (*terminal authentication*). Next, the server transmits parameters for verifying the validity of the broadcast transmission. The server also requests the protected area to synchronize its wall clock time to the server time and to initialize its local storage for storing data during the betting session. Finally, the terminal closes the two-way link and starts waiting for the event to begin.

5.3. Betting session

The betting session utilizes only the broadcast link. Each broadcast packet contains data for both the protected and unprotected areas. The unprotected area contains the betting application and the protected area ensures the validity of the betting session. The RTB session communication is illustrated in Fig. 6. Upon the reception of a broadcast packet, the unprotected area copies and forwards it unchanged to the protected area. The protected area verifies the authenticity and timing of the packet. The unprotected area is informed about the result with a *status* message. If the packet is valid, the unprotected area continues processing it.

The broadcast packet types and contents area shown in Fig. 7. All of them contain protocol, event, and packet identifiers (ID), size, type, and a server time stamp and signature. The event, packet, bet, and tick IDs are sequence numbers. The server ensures that each event has a unique event ID and each packet, bet, and tick ID is unique during an event. The open, close, and result packets of a bet share

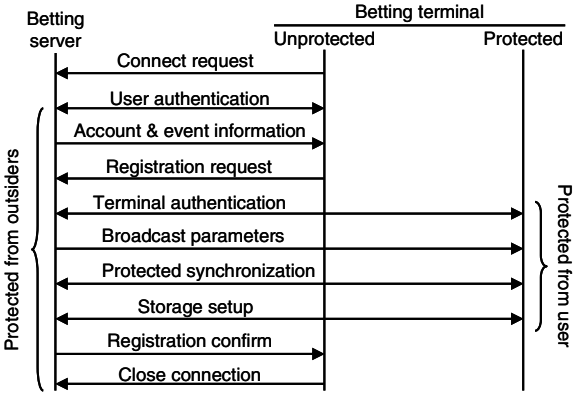


Fig. 5. Offline RTB registration.

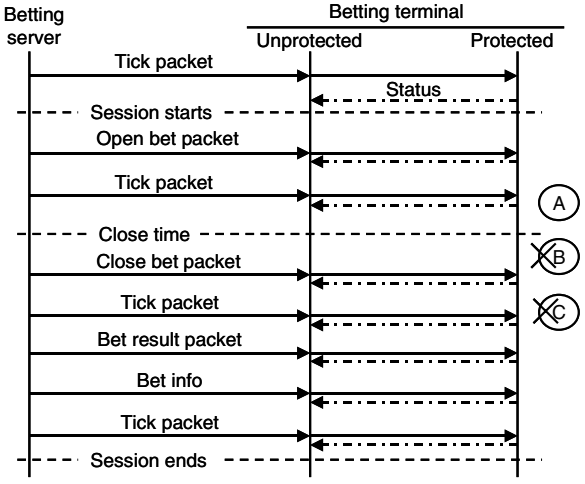


Fig. 6. Offline RTB session.

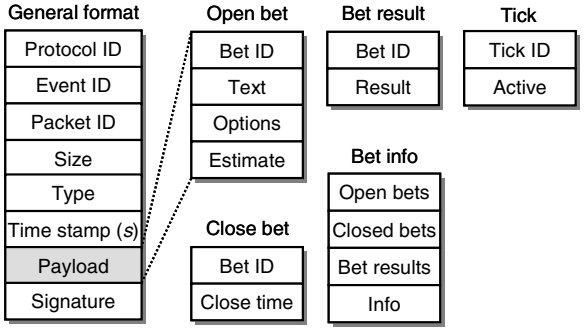


Fig. 7. Broadcast packet formats.

the same bet ID. Before a target event begins, the server starts transmitting periodic *tick packets*, which are used for timing. The *active* field defines whether the event has started or not.

As a suitable incident emerges, the operator opens a bet at the server and an *open bet packet* is broadcast. It contains a bet ID, textual target information, result options, odds, and an *estimate* of available betting time. The user may choose a stake and place a bet. The placement information is given to the protected area for time-stamping.

Before the outcome is settled, the operator sends a *close bet packet.* It contains *close time*, a time instant after which placing a bet for the incident was not allowed. Already the close bet packet disables placing the bet at a terminal (placement C in Fig. 6). In addition, the protected area verifies that the bet was placed before the close time (placement A). This *guard period* ensures that the broadcast delay, the slow reaction of the betting organizer operator, and synchronization inaccuracies do not allow placing bets after the outcome of the incident is already known (placement B). The protected area informs the unprotected area with a status message whether a placement was accepted.

When the outcome is clear, the terminal receives a *bet result packet*. The unprotected area compares the received result with the user's prediction. If they match, it adds winnings to the local balance. *Bet info packet* is a periodic information packet. The *open bets list* of the packet contains the bet IDs of last few open bets. Similarly, the *closed bets list* includes the bet IDs of last few closed bets and their close times. The *bet results list* carries the results of last few bets. The lists allow users to stay up-to-date even if they miss bet packets. An operator may transmit additional information to users in the *info* field.

The betting continues until the active field in tick packets is cleared or the user decides to stop betting. A terminal continues receiving tick packets until betting is stopped.

5.4. Bet record collection

The bet record collection utilizes the two-way data link. At a suitable time, the server reuses the previously negotiated data link session or reinitiates the user authentication. It requests the terminal to transmit its stored betting session data consisting of the bet records and the trace of the session. The session data are stored in the unprotected storage shown in Fig. 4. As described in Section 6, they are cryptographically protected with the keys of the protected area. Hence, the session data are not compromised in the storage or during the transmission to the server.

The server checks that the data are valid and computes the final outcome for the session. Winnings are added to the betting account and the user is informed about the final result.

6. Offline RTB terminal design

The betting terminal plays the key role in ensuring the authenticity of the offline RTB session. In this section the design of the offline RTB terminal is presented. The security rationale for the design choices is given in each subsection. Many of the utilized components, such as the cryptographic algorithms, are well-known as well as widely used and trusted in other contexts. According to the authors' knowledge, in this work they are combined and utilized in a novel way and for a new application purpose. In addition, the methods for ensuring the secure timing at the terminals using the broadcast and combining them with the time-stamping of the protected storage are novel.

Compared to existing distributed systems, the time synchronization problem is different in the offline RTB. Usually, the user wishes the synchronization to be as accurate as possible. On the contrary, in the offline RTB a malicious user wants the synchronization to be inaccurate so that bets could be placed after the results of target incidents are available. The offline RTB also changes the synchronization requirements and possibilities from existing systems. The absolute long-term synchronization accuracy does not have to be very high since the synchronization has to be maintained for a fairly short period of time. In addition, the timing resolution can be moderate as only the timing of actions performed by human users is considered. However, the synchronization should still be rigid and respect certain limits for the trustworthiness of betting sessions.

The broadcast synchronization methods of common technologies, such as Global Positioning System (GPS), the National Institute of Standards and Technology (NIST) time service [18], and the de facto computer network synchronization standard Network Time Protocol (NTP) [19], are not suitable for the offline RTB betting sessions, although their accuracy can be very high. Typically, in those systems the synchronization is a continuous process that lasts throughout the operation time of a device. Contrarily, in the offline RTB the synchronization can only be performed before the target event begins, as explained in Section 6.2.4.

The functional entities of the offline RTB terminal are shown in Fig. 8. The protected area contains components for broadcast security, terminal data security, storage management, protected time control, tick control, and a counter. The unprotected components are network and user interfaces, user data security, game engine, and data storage.

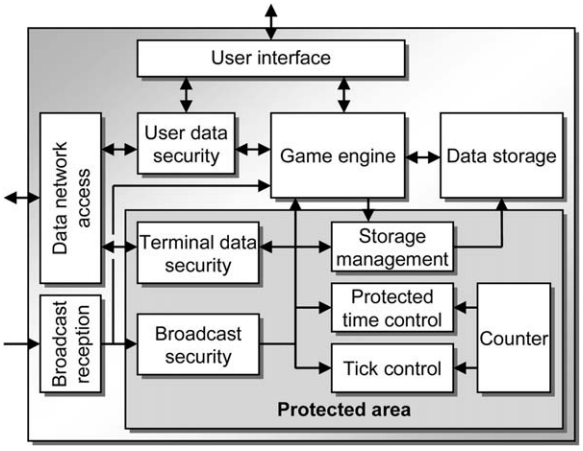


Fig. 8. Offline RTB terminal architecture.

The game engine in the unprotected area contains the functionality of the betting application. It communicates with the user data security entity, User Interface (UI), data storage, broadcast interface, and protected area. The game engine generates bet records from user inputs, computes outcomes, updates the balance, and returns the session data to the server. The protected area makes decisions about bet placement acceptance, verifies the validity of terminal operations, controls the authenticity and timing of broadcast, time-stamps bet records, and creates a complete, protected session trace.

In the following, the applied cryptographic algorithms and communication protocols are considered secure, meaning that either they have been proven secure or it is not known how to break them. For example, it is assumed that it is not possible to find a collision for the utilized one-way hash, fabricate a digital signature, or eavesdrop on message exchanges of a protected link. The assumption is typical in security designs in which a specific cryptographic algorithm is not important but rather how and for what purpose it is used. If the properties of an algorithm are not found sufficient, it can be switched to another one. A similar assumption is often made in theoretical cryptography, e.g., about the underlying algorithm when proving a security bound for a data authentication scheme [20].

6.1. Communications

The protected and unprotected areas share the two-way data link by creating their own logical connections to the server. It is required that both the connections are mutually authenticated and fresh session keys are generated between the communicating parties. In addition, the utilized security protocol must ensure the confidentiality, freshness, and authenticity of the communications following the authentication. Commonly trusted secure channels, such as the Transmission Layer Security (TLS) protocol [21], are suitable for the purpose.

The *user data security* entity performs the user authentication over the data link. The mutual authentication is required for preventing the users from revealing their login information to a fraudulent party and for ensuring that the server only allows authentic users to access their accounts and register to betting sessions. The authentication generates a shared master key K_m between the unprotected area (user) and the server. The session key K_s is computed using a cryptographic hash algorithm (H) for the key separation

$$K_s = H(\text{“SessionKey”}|K_m), \quad (1)$$

where | stands for concatenation. For instance, SHA-256 [22] is a suitable hash algorithm. After the authentication the connection is protected with K_s. Another key is also computed from K_m in Section 6.3.

The user authentication allows only communicating with the unprotected area. The authentication can be based on user name/password pairs, public key certificates, as well as smart cards. The TLS handshake [21] using both the server and client side certificates is a valid method. Another suitable technique is the Secure Remote Password (SRP) protocol applied for the TLS authentication [23]. SRP provides password-based mutual authentication and session key generation without requiring Public Key Infrastructure (PKI).

The communication between the protected area and the server is protected in the same way by the *terminal data security* entity. In order to decrease the processing requirements of the protected area, an authentication protocol that does not require public key operations is preferred. Kerberos [24], which has been widely employed and analyzed, is such an authentication system. The authentication material is pre-installed in the protected area. The fresh session key generated during the authentication must be long (128 bits) and it is used for protecting the protected area initialization from outsiders, including the user.

In contrast to the two-way link, broadcast packets presented in Fig. 7 carry data for both the protected and unprotected areas. As the packets do not contain any confidential information, it is enough to ensure only the authenticity of the data, which is provided with a digital signature in each packet [25].

The broadcast packets are processed by the *broadcast security* entity of the protected area. In the registration the server transmits its public key to the entity for verifying the authenticity of the broadcast. The signing key is only valid for a single betting session. Therefore, the signature does not have to be extremely strong, which results in lower processing requirements. During a betting session an invalid signature does not trigger any other actions except for dropping a packet. However, it may later disable the operation due to failing in the timing checks of the following sections.

Alternative, the signatures can be verified in the unprotected area, which further decreases the cost of the protected area. This is enough for convincing the user that the broadcast is coming from an authentic source. However, as the protected area and the server cannot trust the unprotected area, the protected area must store the

complete contents of each received broadcast packet into its betting session trace. When the trace is returned to the server, it can verify that the received broadcast data are the same as the transmitted data. The drawbacks of the solution are the increased storage size and the fact that broadcast tampering by the user is only detected after the end of the session.

The broadcast security entity also verifies that no packets have been missed or replayed. This is carried out by checking the event ID, packet ID, and tick ID fields. The event ID is unique and a packet or tick ID should always be one greater than the previous one. Packets with passed IDs are dropped. Packets with higher IDs are accepted but the actions of Section 6.5 are performed.

6.2. Timing

The procedures of this section address maintaining a reliable relationship between the time of the protected area and the time of the server and detecting attacks on timing. An attack against these procedures is considered successful if an attacker (a malicious user) manages to place a valid bet after the result of a target incident is known. For a successful attack, the human attacker must have enough time to first observe the result of the incident and then place the bet.

6.2.1. Protected counter

All the time-related processing in the protected area is based on the protected *counter*. The main purpose of the counter is to provide time stamps for bet records. The requirements are that the counter is monotonic, relatively stable and accurate (its drift rate from the server clock is small during a betting session), it does not roll over between the registration and the bet record collection, its precision is at sub-millisecond level, and its frequency cannot be significantly affected by changing the operating conditions (voltage, temperature, electromagnetic fields etc.). In order to transform counter cycles into seconds, a fixed mapping value (frequency) f is stored in the protected area. The server requests and stores it in the registration for the final verifications of the bet record collection.

The counter is implemented in hardware. Preferably, the protected area contains several parallel counters implemented in different technologies, which makes it extremely difficult to modify all of them in the same way. The rates of the counters must remain constant relative to each other. The drift rate requirement is not very strict as the counter is used for timing actions performed by human users and the length of a betting session is typically limited to a few hours. An attacker can only attempt exploiting the counter inaccuracy if its drift is negative from the server clock. A positive drift only means that the user has less time for placing a bet.

In the following sections the counter drift rate is referred to as u and the time the counter has been drifting as T (i.e. T is the time elapsed from the previous synchronization). For example, with a typical oscillator stability of ± 15 ppm (parts per million) the rate u is ± 15 ns per second. When T is 5 h, the total drift is at most ± 270 ms, which is definitely too little for a successful attack. Oscillators with much better stabilities (0.5 ppm) are generally available [26]. Because of the guard period, bets are also closed earlier, e.g. 1 s before, than exactly at the time of the result resolution and much larger errors can be tolerated.

The *tick control* entity employs the counter and the tick packets for controlling that broadcast transmission is actually received and it is not delayed. Each tick packet has a predefined time window, inside which it must be received. Even though it is difficult to tamper with the protected area, the tick packet verifications are used for detecting changes in the counter rate.

The counter is used by the *protected time control* entity for maintaining an estimate of the server wall clock time during the betting session. The estimate is required for time-stamping bet records and verifying the timing of broadcast packets, which prevents recording and replaying a complete session. The entity also controls that the error of the wall clock estimate does not exceed set limits. The timing parameters of a betting session are defined in Table 1.

6.2.2. Counter verifications

The relations of tick packet parameters are presented in Fig. 9. The server defines an interval b for successive ticks during the registration. When the protected area receives its first tick after the registration stage, it records the counter value of the reception time r_0. Then, the tick control computes an estimate for the reception time of the next tick packet

$$r_i = r_{i-1} + fb, \tag{2}$$

Table 1
Offline RTB timing parameters

Parameter	Unit	Description
b	seconds	Pre-defined reception interval for tick packets
c_i	cycles	Read counter value
d	seconds	Minimum broadcast delay
e_i	cycles	Measured error between r_i and c_i
f	Hz	Fixed protected counter mapping
g	seconds	Allowed deviation for the broadcast time stamps and the terminal wall clock estimate
h	seconds	Allowed error for the reception times of tick packets
m	seconds	Minimum data link round-trip delay
o	seconds	Offset of the wall clock estimate
p	seconds	Limit for the data link round-trip delay
r_i	cycles	Precomputed reception time for tick i
s	seconds	Wall clock time stamp of the server
t	seconds	Terminal estimate of the server wall clock
T	seconds	Time elapsed from the previous synchronization
$T_{\max}$	seconds	Maximum allowed value for T before betting session
u	ppm	Protected counter drift rate
v	seconds	Standard broadcast delay variation

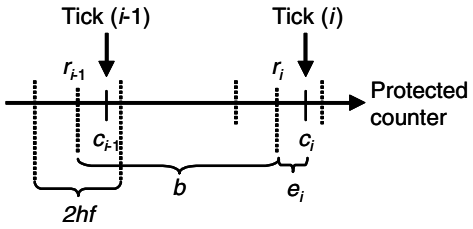


Fig. 9. Tick packet timing.

where $i = 1, 2, \ldots$ is the sequence number of the received tick. Thereafter, upon the reception of the next tick packet, an error value is computed

$$e_i = |r_i - c_i|, \tag{3}$$

where c_i is the real, read counter value at the reception time. The tick control compares

$$e_i < hf, \tag{4}$$

where h is the allowed error in seconds between the real and the estimated tick reception times defined in the registration by the server. If the condition does not hold during a betting session, it is deduced that either the counter rate has changed or part of the broadcast transmission is intentionally delayed. A violation is recorded (Section 6.5). If the condition of Eq. (4) fails a certain number of times before the event has started, resynchronization is triggered as described in Section 6.2.4, the value r_0 is refreshed, and computing Eq. (2) is restarted.

The size of the error limit h depends on the quality of the utilized broadcast channel. If the channel has a well-defined, constant standard delay, h can be small and already small violations are detected. However, because the reception time of the first tick packet r_0 is used as the reference point for the rest of the ticks, the limit h should be larger than the standard delay variation v of the broadcast channel. Otherwise, if the standard delay of the first tick packet is coincidently at the maximum level, the next packets may not satisfy Eq. (4), even if their delay was also inside the standard variation. By choosing $h = 2v$, this situation cannot occur, as clarified in Fig. 10.

After the first tick, the broadcast transmission can intentionally be delayed at most by $h + uT$. This means that in addition to h, an attacker can cumulatively delay the broadcast at most by uT. The term *cumulative delay* refers to the total delay that is constructed by delaying each packet with a small, non-violating amount. With the example of Section 6.2.1 the maximum cumulative delay after five hours is only 270 ms, which is not enough for a successful attack. The cumulative delay is limited to the counter drift because the reception time estimates (r_i) are computed from the reception time of the first tick after the synchronization. An exploitable delay could be caused if the next estimate was always calculated from the real reception time of the previous tick (c_{i-1}). This would enable postponing the broadcast by $nh + uT$, where n is the number of received tick packets. For example, if tick packets were transmitted in 30 s intervals and $h = 20$ ms, after 5 h the cumulative delay would be over 12 s, regardless of the counter stability.

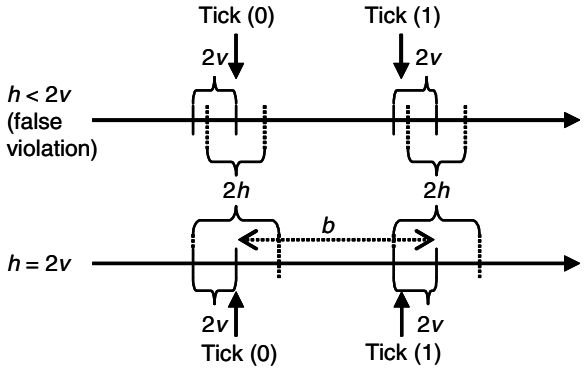


Fig. 10. Choosing the limit h for tick packets.

The verification of Eq. (4) does not trigger countermeasures if the complete broadcast is delayed with a fixed offset (shifted), because it only verifies the tick reception times relative to each other, starting from the tick received first. This type of a complete session replay is prevented with the wall clock estimate methods described in the next two sections.

6.2.3. Wall clock estimate

The local estimate of the server wall clock time maintained by the protected time control entity is

$$t_i = t_{i-1} + f(c_i - c_{i-1}), \tag{5}$$

where $i = 1, 2, \ldots$ is the sequence number of the estimate update operation, c_i is the value of the protected counter at the time of performing the update, and $t_0 = c_0 = 0$. The update procedure is performed every time the estimate is read.

The wall clock estimate is synchronized to the server wall clock time in the registration through the two-way data link. In the basic method [27] the wall clock offset at a terminal is

$$o = \frac{(s_r - t_t) + (s_t - t_r)}{2}, \tag{6}$$

where t_t is the terminal wall clock estimate when transmitting a synchronization request, s_r is the server wall clock when receiving the request, s_t is the server wall clock when replying, and t_r is the terminal wall clock estimate when receiving the reply. For better results, the message exchange is performed several times. The offset is added to the wall clock estimate

$$t_n = t_n + o, \tag{7}$$

which is the value of the first wall clock estimate after the synchronization. The value t_n on the right side of the equation is computed according to Eq. (5). The offset o is also recorded by the server for the final session verifications.

For better accuracy, the two-way wall clock synchronization can utilize more advanced methods, such as NTP and IEEE 1588 [28]. For example, NTP estimates the dispersion, frequency error, and stability of the local clock as well as handles discontinuations. The newest NTP version also

includes cryptographic methods for authenticating the synchronization source [29]. In the offline RTB registrations these cryptographic features are unnecessary as the same protection is already provided by the created two-way connection.

Regardless of the used synchronization method, it is required that the protected time control reasonably accurately knows the network round-trip delay of the message exchange and discards messages exceeding a certain tolerance level. Otherwise an attacker can try to delay the messages to make the wall clock estimate loose time. Before the offset computations, the protected time control compares

$$(t_r - t_t) - (s_t - s_r) < p, \tag{8}$$

where p is the limit for the data link round-trip delay defined by the server before beginning the synchronization procedure. If the condition is not satisfied, the values are discarded.

The smaller p is chosen the better synchronization is achieved. However, p cannot be smaller than the real minimum round-trip delay, since in that case none of the message exchanges satisfies Eq. (8). A similar delay tolerance has been introduced in [30] in order to ensure precise synchronization in distributed systems. NTP also sets limits for the round-trip delay in its clock selection and adjustment algorithms. In this work the limit serves for two purposes: ensuring precise synchronization for benign users and limiting the advantage (offset error) a malicious user can achieve by delaying the synchronization message exchange. Assuming that the protected counter drift rate compared to the server clock is negligible during the message exchange, the maximum offset error over a symmetric link is

$$\frac{p-m}{2}, \tag{9}$$

where m is the real minimum round-trip delay [30].

If the counter satisfies its requirements, resynchronization is not required after the registration since a betting session is fairly short, typically few hours (cf. Section 6.2.4). For example, when considering $p = 10$ ms and $m = 3$ ms, which was the average round-trip delay for WLAN in Section 4, with the typical counter stability of Section 6.2.1 the wall clock estimate error is at most ± 273.5 ms after 5 h. This is still too little for enabling a successful attack.

6.2.4. Wall clock verifications

The protected time control entity uses the wall clock estimate and the server time stamps of the broadcast packets for verifying that individual packets arrive at the right time, bets are placed before closing times, and the complete broadcast transmission is not intentionally delayed, including the first tick packet. The relations of the wall clock verification parameters at the protected area are illustrated in Fig. 11.

In the registration the server defines the estimate for the minimum broadcast delay d and the allowed deviation g for the server time stamps of broadcast packets and the local wall clock estimate. A similar broadcast delay estimate

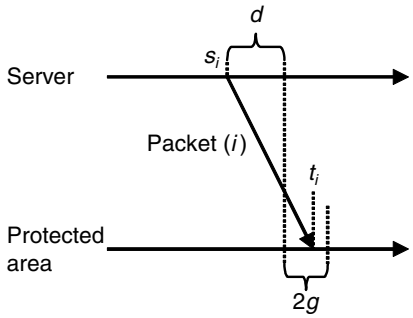


Fig. 11. Wall clock estimate timing.

can be used in NTP. The deviation limit g includes the standard broadcast delay variation v. Upon the reception of a packet, the protected time control compares

$$|t_i - (s_i + d)| < g, \tag{10}$$

where $i = 0, 1, 2, \ldots$ is the sequence number of the received packet, t_i is the value of the local wall clock estimate at the time of reception, and s_i is the server time stamp of the packet. If the condition is not satisfied during a betting session, the protected area performs the actions defined in Section 6.5. The verification detects packets that have been delayed more than $g + uT$.

In addition to limiting individual packet delays, the verification ensures that the broadcast transmission can intentionally be shifted at most by g. Here the coefficient uT is negligible because T is very small at the time of receiving the first tick. After the first tick the cumulative delay is limited as described in Section 6.2.2. Hence, the total intentional delay an attacker can cause without detection is limited to $g + h + uT$.

If the target event has not begun and Eq. (4) or Eq. (10) fails a server-defined number of times or T exceeds a server-defined limit T_{max} (e.g. 2 h), resynchronization is performed over the two-way data link. This ensures that betting is not denied from benign users during the upcoming session because of the instability of the protected counter. It also limits the advantage the counter instability can provide to a malicious user. The cumulative delay is limited to $u(T_{max} + w)$, where w is the elapsed time from the betting session start and the last synchronization was performed T_{max} before the start. Therefore, the maximum undetectable delay during a betting session is limited to $g + h + u(T_{max} + w)$. The resynchronization reuses the previously negotiated data link session or reinitiates the authentication procedures for a fresh connection and it is carried out as the original synchronization. The server records the new offset.

When a close bet packet is received, the protected time control entity verifies that the corresponding bet has been placed before the close time. The close time and the time stamp of the bet record are directly compared.

The wall clock verifications cover other timing attacks except the modification of the offset or the counter rate at the same time with the intentional delay of the broadcast. The offset modification with delayed broadcast is

detected by the counter verifications. The counter verifications can also be more precise since the limit h must only include the variation v. The limit g must contain both v and the allowed wall clock estimate error. In addition, the comparison of Eq. (10) includes the estimate d. The offset change is also detected in the bet record collection since both the counter value and the corresponding wall clock estimate are recorded with each entry in the bet storage and the server has recorded the negotiated offsets.

If broadcast time stamps are available, they are usually utilized for adjusting local clocks in other systems. However, it is not secure to use the broadcast time stamps as they are typically used, e.g. in GPS, the NIST time service [18], and NTP, instead of resynchronizing over the two-way data link prior to the session. Because the limit h is stricter than g, Eq. (10) only fails before Eq. (4) if the complete broadcast transmission is shifted more than the amount of g or if a broadcast packet other than a tick packet is delayed outside $g + uT$. Therefore, if the protected counter, which is used for maintaining the wall clock estimate, has drifted too far from the server clock after the synchronization, it is already detected as a failure in Eq. (4). Without the protection against the cumulative delay, the broadcast time-stamps could be utilized for maintaining the wall clock synchronization for a benign user. Unfortunately, at the same time a malicious user could utilize the cumulative delay for postponing the end of the betting session. The cumulative delay would automatically make the adjusted wall clock estimate loose time by $g + uT$ per packet without detection, and thus, allow a successful attack.

The combined counter and wall clock verification methods detect all the possible attacks when only the protected counter rate or only the broadcast delay can be tampered with. The detection accuracy depends on the chosen timing parameters, the stability of the counter, and the accuracy of the knowledge of the utilized network technologies. Successful attacks can only be implemented if the attacker is able to modify the counter rate unlimitedly and delay the broadcast transmission simultaneously. Such an attack can technically be prevented by randomly reopening the two-way data link during the betting session, integrating the wireless network transceiver or, e.g., a reference GPS receiver inside the protected area. In that case, tampering with the counter rate also prevents the synchronization to the wireless medium, which is detected as a connection failure of the betting terminal. As a non-technical protection, the betting organizer should keep a log on winnings so that if a user starts succeeding suspiciously often, the authenticity of her activities can be verified.

6.3. Storage management

The accuracy and correctness of timing alone is not enough for ensuring the security of a betting session. It is also required that the stored bet records are authentic and their information, including the time stamps, has not been tampered with. An attack against the storage procedures is successful if a malicious user manages to fabricate, modify, or delete a bet record without detection.

In order to reduce the resource requirements and cost of the protected area, the storage for the session data is located in the unprotected area of the terminal. The storage entries are cryptographically chained in the protected area by the *storage management* entity. The chaining is based on [31] in which a method for creating protected audit logs in untrusted environments is designed. In this work the design is simplified as the stored data does not have to be kept secret (encrypted) from the user and there is no need for supporting partial access or verification of the stored data. In addition, in this work it is defined how the chain is bound to both the user and the protected area. The format of the storage contents are defined as well.

The chain is computed using a keyed Message Authentication Code (MAC) algorithm [32]. The chaining could also utilize digital signatures. However, a MAC algorithm requires less processing power, and thus, it is favored to decrease the processing requirements and cost of the protected area. The widely utilized HMAC [33] using SHA-256 [22] or Advanced Encryption Standard (AES) [34] in the CMAC mode [35][20] are well-suited for the purpose.

A storage entry consists of data and an appended chain value. The chain value is computed from the data, the chain value of the previous entry, and the current chain key A_i with the MAC algorithm

$$\text{chain}_i = \text{MAC}(A_i, \text{chain}_{i-1} | \text{data}_i). \tag{11}$$

The previous chain value for the first entry is chosen to be zero [31].

The initial chain key is generated from the master key K_m of the user authentication and from a unique seed q, which the protected area receives from the server in the storage setup of the registration. The parameter q can also be derived from the session key between the protected area and the server using the key separation as in Eq. (1). It is required that q is long, at least 128 bits, in order to thwart brute force searching for the initial chain key. Using the key separation, the initial chain key is

$$A_0 = H(H(\text{“StorageKey”} | K_m) | q). \tag{12}$$

K_m of the inner hash ties the identity of the user to the chain. On the contrary, since q is defined secretly from the user, it protects the chain against user modifications. After this, q is irretrievably destroyed at the terminal. The server computes the same initial key and stores it with the user registration data for the final verifications.

Thereafter, a different key is used for each stored entry [31]. After storing an entry, a new key is computed

$$A_{i+1} = H(\text{“UpdateKey”} | A_i). \tag{13}$$

The previous key is always irretrievably destroyed. Hence, even compromising a terminal will not allow changing entries that have been generated with a used key. Resetting or shutting down the protected area must also destroy

the chain key. This prevents a malicious user from attempting to turn back the protected counter through a restart, e.g., if the counter starts every power cycle from zero. If the chain key is lost, the user must reregister and resynchronize for the event to be able to continue the session.

The chain ensures the authenticity of the data, since altering or deleting an entry makes also the rest of the storage invalid. Since each entry is protected with a different key and the keys are generated with a cryptographically secure one-way hash function, an attacker can only attack a single entry alone. The amount of available data per key is minimized. Furthermore, since each key is used only once and each storage entry has a strictly defined format (next section), the attacker cannot exploit the protected area for requesting additional MACs for a key or for freely chosen data.

6.4. Stored data

To create a complete, verifiable trace for a betting session, a new entry is written to the data storage for each received broadcast packet and placed bet. Each entry generated by the storage management contains event ID, entry ID, size, type, protected counter value at the time of entry creation, protected wall clock estimate at the time of the entry creation, entry data, and chain value. The entry ID is a per-event sequence number for the storage entries maintained in the protected area. The value of the type field is *initial entry*, *broadcast packet*, *bet record*, or *final entry*. The contents of the entry types are presented in Fig. 12.

At the end of the registration, the storage management opens the storage by creating an initial entry with the initial chain key [31]. The entry data are user ID, terminal ID, and initial balance. The user and terminal ID are the same that were utilized in the user and terminal authentication. When the terminal receives a valid broadcast packet, the entry data copied from the packet are event ID, packet ID, packet type, and tick or bet ID (zero for a bet info packet). For a placed bet the entry data are bet ID, identifier of the chosen option, stake, and current balance. The identifiers are copied from an open bet packet. The stake is given by the user and the current balance is computed by the game engine. When a betting session ends or when the user decides to stop betting, the storage is sealed with the final entry.

Each stored entry has unique contents (unique event ID under which the entry IDs are unique) and defined format. In addition to the unique keys, this further restricts an attacker not knowing the chain key to attempting the modification of very limited number of entry fields. When a secure MAC algorithm is used, it is not possible to change those fields – especially into anything meaningful – without also changing the MAC value. The purpose of the initial and final entries is to prevent the user from restarting the storage or wiping out the complete session data [31]. After the initial entry has been created, data can only be added to the storage. After the final entry, the storage management destroys the chain key to disable further additions.

After collecting the session data, the server verifies that the storage chain is valid, based on the recorded registration data. It also checks that the time stamps and counter values of the entries are consistent with each other and with the broadcast timing information stored at the server during the session.

6.5. Violations

Since during an offline RTB session terminals only receive data over an unreliable broadcast link, it is probable that some packets are missed. Casually missing an open, close, result, or info packet is not crucial for the security of the organizer. If a bet packet is missed, a user misses the information related to a bet target. The missed information is received in the next bet info packet. The user only has less time for betting or receives the placement acceptance notification or the bet result later. A tick packet miss or failing in tick timing verifications at a terminal is considered more severe.

The protected area maintains two separate counters, *violation counter* and *severe violation counter*, for logging failures during a betting session. A bet or info packet miss or

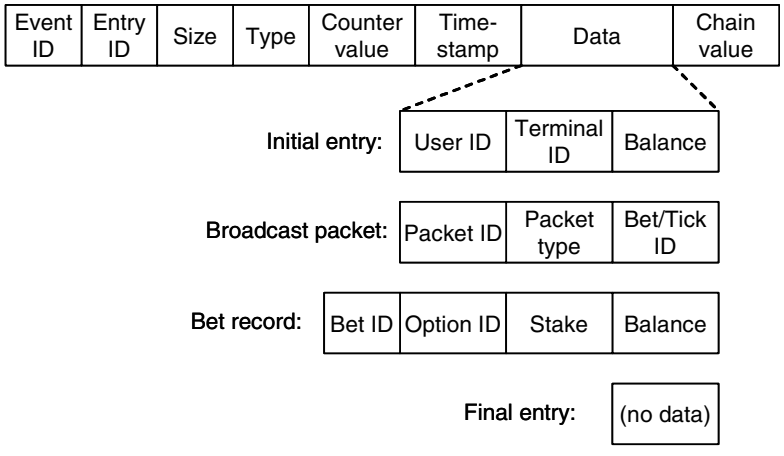


Fig. 12. Contents of stored entries.

timing error increment the violation counter. A tick packet miss or timing error increment the severe violation counter. The server defines thresholds for both the counters in the registration. If a threshold is reached, the data storage is sealed with the final entry and betting is disabled. The bets placed so far are valid. By defining the limits to one, any violation disables continuing betting.

7. Technologies for offline RTB

There are several alternatives for offline RTB implementation technologies. In set-top boxes DVB can be utilized for the broadcast data delivery whereas the two-way communications utilize an available return channel (e.g. IP network, analog modem). The set-top box carries out the unprotected processing and the protected area is implemented in an additional expansion module. Currently, a new standard, DVB-H, for handheld devices is emerging. It expands the DVB coverage from set-top boxes to small, portable devices with less processing capacity. For instance, a PDA or a mobile phone with GPRS/UMTS and DVB-H support is a suitable platform for the offline RTB. Most mobile phone producers have already built DVB-H terminals. Another solution for advanced multimedia terminals as well as small hand-held devices can utilize Digital Audio Broadcasting (DAB) as the broadcast technology.

As supporting both the broadcast and the two-way traffic, WLAN technology is suitable for localized RTB services, e.g. in a stadium. When WLAN terminals only passively receive broadcast traffic during a betting session, the limit for the maximum number of terminals in one network is unrestricted. Depending on the terminal type, the protected processing can be implemented on a PC-card or some other dedicated expansion card.

Currently, smart cards are utilized in a wide range of e-commerce applications for authentication and storing private information. Due to their tamper-resistance and support for cryptographic operations, they have suitable characteristics for the implementation of the protected area of the offline RTB terminal. The protected area control can be implemented in the smart card processor and the rest of the processing as special functional units. However, current smart cards do not contain fixed, internal counters for time-keeping. Instead, they support variable operating frequencies supplied by an external oscillator. It is also recommended that smart cards randomly vary their internal clock cycle length during the operation in order to prevent timing and power attacks against cryptographic operations [36]. These features make smart cards in their current form unsuitable for the offline RTB.

In addition to dedicated solutions, the technology developed by Trusted Computing Group (TCG) [37] has potential for the protected area implementation. TCG is defining hardware-based solutions in order to increase the level of security and trust in various computing platforms, such as PCs and mobile terminals. The devices are specified to include a tamper-resistant Trusted Platform Module (TPM) that provides cryptographic operations and verifies the integrity of the computing environment of the main platform. The TPM specification also includes a hardware counter that can be used for time-stamping with digital signatures. Of existing technologies, TPM is the closest match with the protected area design of this work. However, TPM does not define any method for ensuring the relationship (synchronization) between its counter and an external clock ([38], p. 114), which makes it inapplicable as such for offline RTB terminals. For the offline RTB, TPM should also provide a method for securely chaining the data (i.e. bet records) it has time-stamped and signed to prevent erasing them afterwards. The main purpose of TPM is to protect user data against software attacks. It has also been criticized for supporting digital rights management.

If smart cards or TPM included all the features required by the protected area, they would be perfect solutions for the offline RTB. Smart cards are already widely deployed and the TPM technology is emerging in the newest PCs and laptops. The authors believe that the features required by the offline RTB would be useful in other applications as well. They are especially beneficial in applications in which a choice made by a user and the time instant at which it was made are important.

8. Offline RTB prototype

In order to test offline RTB in practice, prototypes for WLAN and DVB environments have been implemented [39]. They enable evaluating design alternatives, network technologies, UIs, and end-user behavior in real environments. The prototypes have been experimented by several test groups. Valuable information on the attractiveness of the concept as well as its usability and applicability has been collected. Main aspects have been to investigate how near the closing time bets are placed, what kinds of bets are suitable, and what a suitable interval for announcing bets is. From the technical point of view, user behavior places requirements on the system performance. The current WLAN implementation can be directly applied for providing localized betting services. It has been tested in an ice-hockey stadium.

8.1. Implementation

The offline RTB prototype developed for WLAN environment is illustrated in Fig. 13. The main functional parts of the terminals and the betting server have been implemented in SDL. SDL is connected to the environment through user and network interfaces. The implementation does not limit the number of terminals accessing the service and it supports several operators and parallel events as well as parallel bets.

The prototype consists of a betting server, one or more operator terminals, an odds server, and several betting terminals. The server can be located in a place with the best network capacity and physical security whereas the

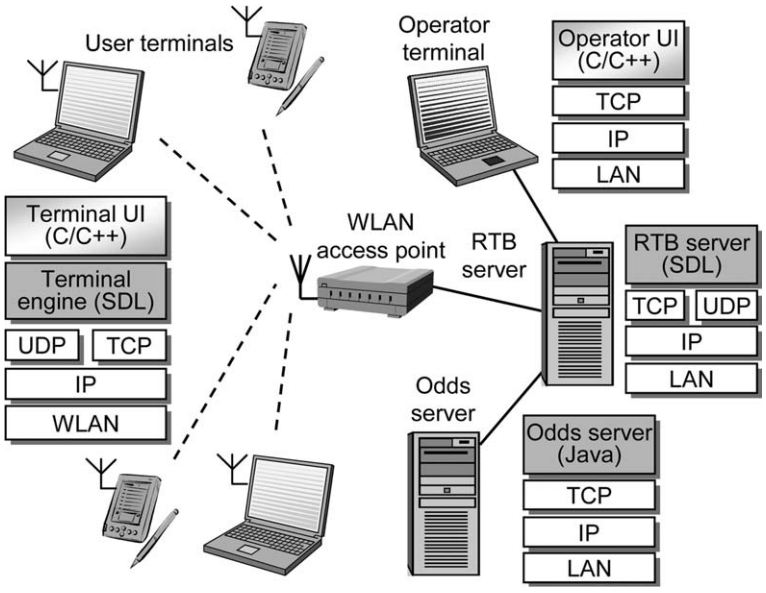


Fig. 13. Offline RTB prototype in WLAN environment.

operator in a suitable place for following the event. The server and operator software run in Windows XP environment. The betting terminal is implemented for Windows XP, Pocket PC, and DVB set-top boxes. The prototype utilizes UDP/IP for the broadcast and TCP/IP for the two-way data connection. The broadcast link for set-top boxes is DVB. An additional PC is used for receiving the WLAN broadcast and converting it into the DVB broadcast.

The operator uses predefined bet templates, which can be used as such or the texts and odds can be modified. The operator can also employ a separate server for dynamic odds. The odds are automatically adjusted throughout an event, utilizing game time, current scores, and expected final scores. When an operator announces a bet with dynamic odds, the offline RTB server transmits a request to the odds server. After the reply, the server replaces the odds provided in the template and broadcasts the announcement to the terminals.

8.2. User interfaces

Suitable UIs for both the RTB users and operators have been evaluated using the prototype. The UIs have gone through several generations and various improvements as well as optional features have been developed, based on the feedback of real end-users.

In general, the usability tests with different implementations have shown that the displayed information must be very clear and concise. Otherwise betting disturbs following the event instead of adding entertaining value to it. Also, because the time for placing a bet is strictly limited, compact and comprehensible information leaves more time for making the actual decision and increases the usability. Incidents should preferably have only few (two or three) outcome alternatives in order to make the information effortlessly manageable for the users as well as for the operators. It has also turned out that during a betting session the users prefer making all selections with the minimal effort, i.e., with a single keystroke or tap. According to the feedback, the RTB service has been found inspiring and entertaining.

Fig. 14a shows the betting UI of the RTB prototype for the Pocket PC touch screen and Fig. 14b for a set-top box in their current forms. Because current set-top box remote controls are fairly slow to use compared to a touch screen, the set-top box UI has been made simpler. The set-top box UI is semi-transparent and it can be hidden to free the full screen for the video. In addition to making a choice, the UIs allow the user to change the stake, discard a bet, view previous bets and their results, and finish the betting session.

When a bet is opened or a bet result received, the UIs show the information immediately on the screen. If the set-top box UI is hidden, only a small notification symbol is shown on a corner of the screen. An optional sound notification can be generated in both UIs so that the user does not frequently have to check the screen while waiting for announcements. Vibration can also be used as another way to notify the user in portable betting terminals, especially in noisy environments (e.g. stadiums). As the bet text is large, the user can quickly decide if the shown information is interesting. In the Pocket PC a bet is placed by simply tapping on one of the provided options. In the set-top box the choice is made with the remote control. The same stake is used for the bets as long as the user changes it. The user can also turn on optional confirmations to prevent accidental placements.

According to the usability experiments, even though the betting screen must be concise by default, the users should

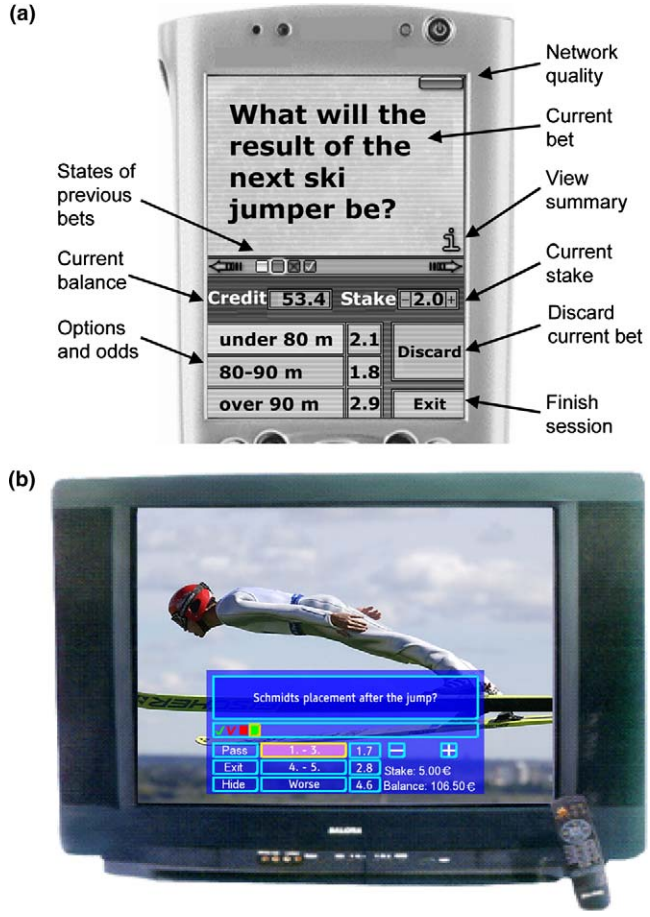


Fig. 14. Offline RTB user interfaces for: (a) Pocket PC and (b) set-top box.

also be able to get more information when they desire. Specifically, it should be easy to browse through previous bets. For the purpose, the UIs maintain lists of passed bets. The user always sees the states of previous bets (open, closed, placed, waiting result, won, or lost) represented as symbols below the bet text. She can check the information of a specific bet by selecting its symbol. Uninteresting bets can be removed from the list. In addition, the user can view a session summary and more detailed information on the bets by choosing the summary view.

In addition to the Pocket PC and set-top box UIs, a stadium screen interface has been developed and tested with the RTB prototype. The bet announcements are shown on the stadium screen, which releases the users from frequently checking their personal betting terminals in the noisy and crowded environment. The operator can choose whether the bet information occupies the whole screen or only parts of it. Similarly to the set-top box UI, the notification can be a small blinking logo or a sound and the detailed bet information is shown on the user terminals. On the other hand, when the stadium screen is utilized for providing the full information, the terminal UI can be made simpler. For example, the UI can consist of a small screen and few buttons just for making a choice.

The main screen of the operator UI is presented in Fig. 15. On the top of the window the operator starts and finishes betting sessions. The bets for a session are managed using the left half of the UI. As the states of the bets are changed, the bets flow from the top to the bottom of the UI. A new bet is defined using empty or prefilled templates as well as static or dynamic odds. The right side of the UI is utilized for reporting real-time session data (number of bets, registered users etc.). The operator can also broadcast textual information to the user terminals and choose the view for the terminals supporting multiple views (e.g. the stadium screen).

The UI tests have shown that the operator should not announce bets too frequently, again, not to interrupt following the event itself. On the other hand, limiting the announcement frequency for all the users in the same way is too restricting. Some users want to have more intense betting sessions than others. Thus, the RTB prototype is being further developed to support different bet announcement patterns. The users will be able to choose the intensity of their betting sessions and also to change their preferences during the sessions. The intensity level affects the announcement frequency and the time to make a decision. The bets are categorized according to their characteristics and the UI chooses which announcements to display based on the chosen level.

8.3. User behavior

In addition to the usability tests, the offline RTB prototype has been utilized for collecting data related to the timing of bet placements during real betting sessions. The target events for the experiments have been ski jumping and ice-hockey, which have different characteristics related to RTB. Ski jumping is strictly periodic with predictable closing times whereas in ice-hockey it is difficult to distinguish discrete incidents. The tests presented here were arranged for two small groups of volunteers, the first one had 21 members and the second one seven. The ski jumping test was performed for both the groups and the ice-hockey test for the first group only.

The results of the timing behavior experiments are shown in Fig. 16. The graphs present how long a bet was open and how near the closing times users placed their bets on average. At the beginning of the ski jumping session, the behavior of new users follows closely the opening times. However, as the users become more familiar with the betting, they start delaying placements nearer the predictable closing time. On the contrary, in ice-hockey the placement times follow the opening times throughout the session because of the unpredictable closing times. It was also verified that defining an estimate for the closing times changes the user behavior in the second test closer to the first one.

In Fig. 16a, the bets 5–11 are especially interesting. The users made their decisions about 5 s before the closing time on average, even though bets were open up to 27 s. This supports the performance simulations and analysis pre-

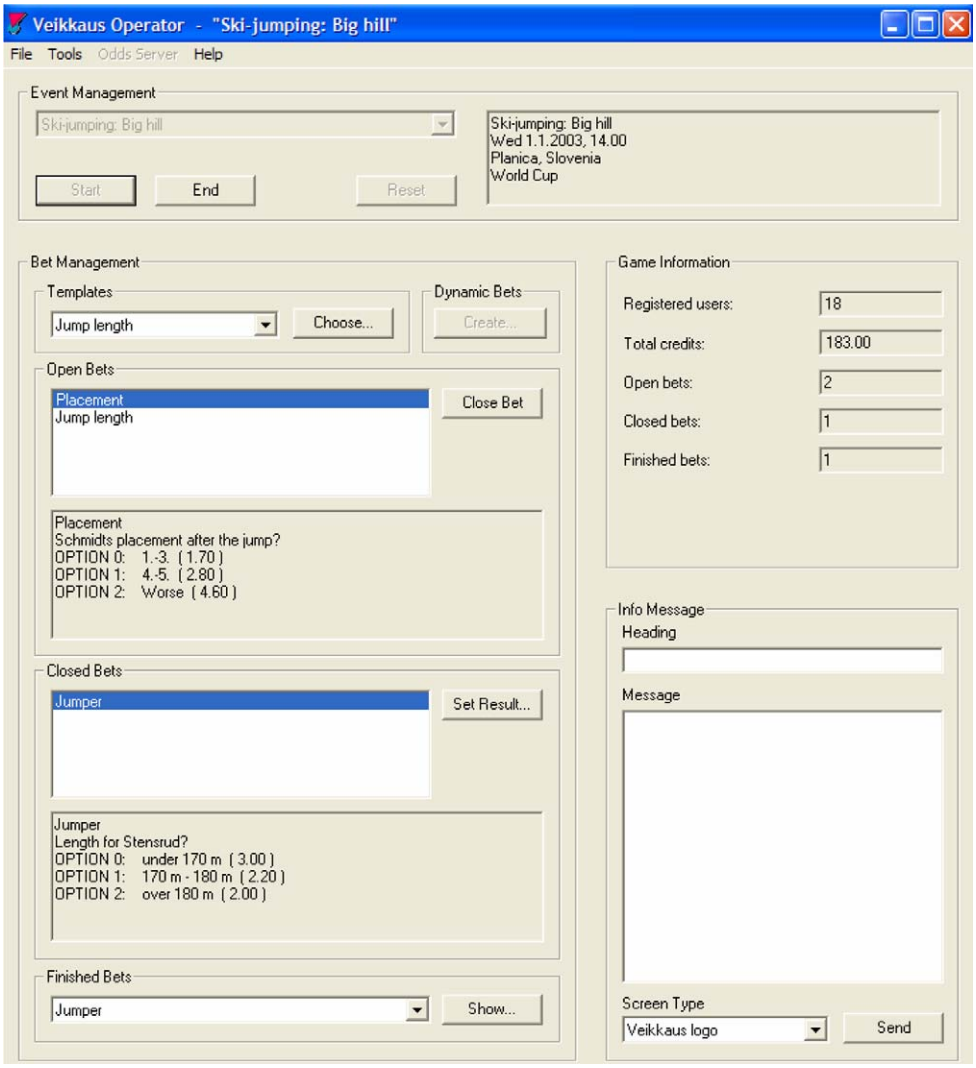


Fig. 15. Offline RTB prototype user interface for operator.

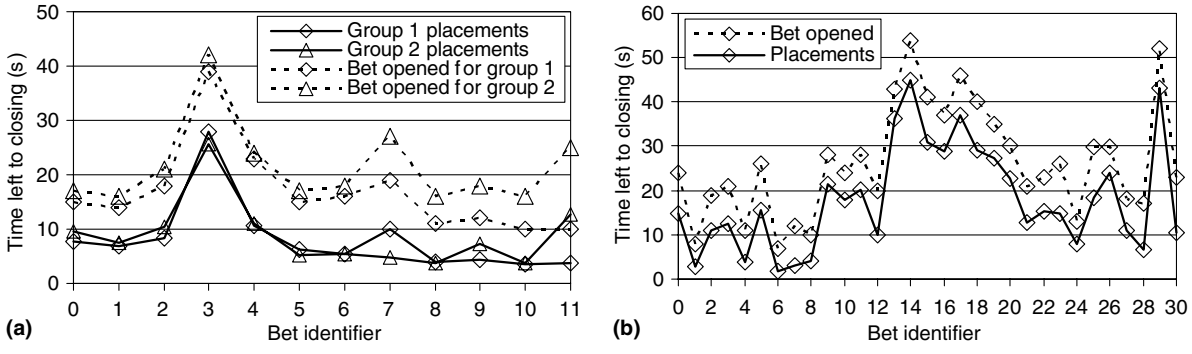


Fig. 16. Results of the timing experiments with the offline RTB prototype: (a) ski jumping and (b) ice-hockey.

sented in Section 4. Another interesting aspect is that even though both the test groups attended the RTB sessions for the first time and without any prior knowledge or guidance, the anticipated timing behavior started to show already after the first four bets.

9. Conclusions

The rapid development of communication systems as well as e-commerce technologies has enabled electronic betting to become an important field of e-commerce and

entertainment. However, the fundamental nature of betting itself has not changed. There are not enough processing resources in electronic betting systems for changing the services into interactive sessions with full scalability support. This was also shown in the simulations and experiments of this work.

In order to provide large-scale electronic betting as a new type of interactive, real-time entertainment, a new approach was presented in this paper, referred to as offline RTB. The offline architecture enables betting in short time cycles while keeping processing requirements low. It results in new challenges for security and timing as the operation environment of a betting terminal has to be considered hostile. Thus, reliable methods for verifying the timing and the authenticity of the terminal operations were developed. In addition to the reliable protocols and algorithms, the critical operations are implemented in a protected area which the user of the betting terminal cannot access.

Regardless of the dependability of the technical protection, the offline RTB still requires agreed rules, which can be applied if malpractices are detected. A trusted supervisor is required for ensuring the authenticity of the organizer operations. Limiting the stakes makes the service less appealing for break attempts with financial goals. Also, instead of providing bets on which large sums would be invested, the goal of the offline RTB is to offer frequent, short bets with small sums. This way the service can be applied for adding entertaining value to various events. In addition, the design is applicable for other types of contests without monetary goals.

References

[1] European Game and Entertainment Technology Ltd Ab (EGET) website, 2005. Available from: <http://www.eget.fi>.

[2] Global Interactive Gaming website, 2005. Available from: <http://www.gigltd.com>.

[3] E. Kushilevitz, T. Rabin, Fair e-lotteries and e-casinos, in: Proceedings of the Cryptographer's Track at RSA Conference 2001 (CT-RSA 2001), San Francisco, USA, Apr., 2001, pp. 100–109.

[4] C. Hall, B. Schneier, Remote electronic gambling, in: Proceedings of the 13th Annual Computer Security Applications Conference (ACSAC '97), San Diego, USA, Dec., 1997, pp. 26–29.

[5] M. Jakobsson, D. Pointcheval, A. Young, Secure mobile gambling, in: Proceedings of the Cryptographer's Track at RSA Conference 2001 (CT-RSA 2001), San Francisco, USA, Apr., 2001, pp. 110–125.

[6] W. Aiello, A. Rubin, M. Strauss, Using smartcards to secure a personalized gambling device, in: Proceedings of the 6th ACM Conference on Computer and Communications Security (CCS '99), Singapore, Nov., 1999, pp. 8–12.

[7] W. Zhao, V. Varadharajan, Y. Mu, Fair on-line gambling, in: Proceedings of the Annual Computer Security Applications Conference (ACSAC 2000), New Orleans, USA, Dec., 2000, pp. 394–400.

[8] K. Sako, Implementation of a digital lottery server on WWW, in: Proceedings of Secure Networking – CQRE (Secure) '99, Düsseldorf, Germany, Nov.-Dec., 1999, pp. 101–108.

[9] J. Zhou, C. Tan, Playing lottery on the Internet, in: Proceedings of the 3rd International Conference on Information and Communications Security (ICICS 2001), Xian, China, Nov., 2001, pp. 189–201.

[10] P.A. Fouque, G. Poupard, J. Stern, Sharing decryption in the context of voting and lotteries, in: Proceedings of the 4th International Conference on Financial Cryptography (FC 2000), Anguilla, British West Indies, Feb., 2000, pp. 90–104.

[11] P. Syverson, Weakly secret bit commitment: applications to lotteries and fair exchange, in: Proceedings of Computer Security Foundations Workshop (CSFW'98), Rockport, USA, Jun., 1998, pp. 2–13.

[12] S. Haber, W.S. Stornetta, How to time-stamp a digital document, J. Cryptol. 3 (4) (1991) 99–111.

[13] Symmetricom, Inc. website, 2003. Available from: <http://www.symmetricom.com>.

[14] SDL Forum Society website, 2005. Available from: <http://www.sdl-forum.org>.

[15] IEEE Std 802.11b-1999, Wireless LAN medium access control (MAC) and physical layer (PHY) specifications: higher-speed physical layer extension in the 2.4 GHz band, 1999.

[16] H.S. Wang, N. Moayeri, Finite state Markov channel – a useful model for radio communication channels, IEEE Trans. Vehicular Technol. 44 (1) (1995) 163–171.

[17] S. Karande, S.A. Khayam, M. Krappel, H. Radha, Analysis and modeling of errors at the 802.11b link layers, in: Proceedings of 2003 IEEE International Conference on Multimedia and Expo (ICME 2003), Jul., 2003, Baltimore, USA, vol. I, pp. 673–676.

[18] National Institute of Standard and Technology (NIST) Time & Frequency Division website, 2005. Available from: <http://tf.nist.gov>.

[19] D. Mills, Network time protocol (version 3) specification, implementation and analysis, RFC 1305, 1992.

[20] T. Iwata, K. Kurosawa, Stronger security bounds for OMAC, TMAC and XCBC, National Institute of Standards and Security (NIST), 2005. Available from: <http://csrc.nist.gov/CryptoToolkit/modes/comments/>.

[21] T. Dierks, C. Allen, The TLS protocol 1.0, RFC 2246, 1999.

[22] Federal information processing standards publication (FIPS) 180-2, Secure hash standard, 1999.

[23] D. Taylor, T. Wu, N. Mavrogiannopoulos, T. Perrin, Using SRP for TLS authentication, TLS Working Group Internet Draft, 2005.

[24] Kerberos website, 2005. Available from: <http://web.mit.edu/kerberos/www/>.

[25] Federal information processing standards publication (FIPS) 186, Digital signature standard (DSS), 1994.

[26] Taitien Electronics website, 2005. Available from: <http://www.taitien.com>.

[27] D. Mills, Simple network time protocol (SNTP) version 4 for IPv4, IPv6 and OSI, RFC 2030, 1996.

[28] IEEE Std 1588-2002, IEEE standard for a precision clock synchronization protocol for networked measurement and control systems, 2002.

[29] Network Time Synchronization Project website, 2005. Available from: <http://www.eecis.udel.edu/~mills/ntp.html>.

[30] F. Cristian, A probabilistic approach to distributed clock synchronization, in: Proceedings of the 9th International Conference on Distributed Computing Systems, Newport Beach, USA, June, 1989, pp. 288-296.

[31] B. Schneier, J. Kelsey, Cryptographic support for secure logs on untrusted machines, in: Proceedings of the 7th USENIX Security Symposium, San Antonio, USA, Jan., 1998, pp. 53-62.

[32] A.J. Menezes, P.C. Oorschot, S.A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.

[33] Federal information processing standards publication (FIPS) 198, The keyed-hash message authentication code, 2002.

[34] Federal information processing standards publication (FIPS) 197, Advanced encryption standard (AES), 2001.

[35] M. Dworkin, Recommendation for block cipher modes of operations: the CMAC mode for authentication, National Institute of Standards and Technology (NIST) special publication 800-38B, 2005.

[36] O. Kömmerling and M.G. Kuhn, Design principles for tamper-resistant smartcard processors, in: Proceedings of USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, USA, May, 1999, pp. 9-20.

[37] Trusted Computing Group (TCG) website, 2005. Available from: <http://www.trustedcomputinggroup.org>.
[38] Trusted Computing Group (TCG), TPM main specification – Part 2: TPM Structures, version 1.2, level 2, revision 85, February 2005.
[39] P. Hämäläinen, M. Hännikäinen, T.D. Hämäläinen, and R. Soininen, Offline architecture for real-time betting, in: Proceedings of 2003 IEEE International Conference on Multimedia and Expo (ICME 2003), Baltimore, USA, Jul., 2003, vol. I, pp. 709–712.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O. Box 527
FIN-33101 Tampere, Finland