Tampere University

Santeri Röning

ATTACKING RSA WITH SIDE CHANNELS

A review for the attacks against RSA

Bachelor of Science Thesis
Faculty of Information Technology and Communication Sciences
Examiners: Tiina Schafeitel-Tähtinen
August 2021

ABSTRACT

Santeri Röning: Attacking RSA with side channels
Bachelor of Science Thesis
Tampere University
Computing Sciences
August 2021

The current world requires a safe and secure method of transporting data through the internet. One of the oldest, still secure and most used methods currently in use is the public-key cryptographic system concocted by Rivest, Shamir and Adleman (RSA). As the systems methods of encrypting data are known, there have been a lot of attacks to it and this thesis aims to review the attacks on RSA via side-channel attacks. Side-channel attacks are attacks that focus not on cracking the encryption itself but on collecting data from other sources. These sources might include but are not limited to sound, timing, electricity draw and electromagnetic radiation. From this leaked data researchers have been able to extract the required decryption keys to decrypt the RSA protection.

This work is divided into three parts. First is the general information about RSA, how it works, why it works and the math behind the encryption. The purpose is to give the reader an introduction to RSA and a bit of background knowledge behind the attacks. The second part is about side-channel attacks. This includes how they work in general, how these attacks can be classified and as examples recent side-channel attacks, spectre and meltdown. The third part aims to gather these first two parts together and create a review about what attacks were successful, how these attacks worked and how to defend against these kinds of attacks.

This thesis concludes that the side-channel attacks are here to stay on the threat radar and a generalization of how we can defend against them. This defending is opened via examples and what it means to the user using these defences against side-channel attacks, as usually, these require additional resources to defend against.

Keywords: RSA, side-channel

TIIVISTELMÄ

Santeri Röning: Sivukanavahyökkäykset RSA:ta vastaan
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Elokuu 2021

Yhteiskuntamme vaatii toimiakseen tietoturvallisen keinon siirtää dataa käyttäjältä toiselle internetin yli. Yksi vanhimmista, yhä turvallisista ja eniten käytetyistä metodeista tähän on julkisen avaimen salausalgoritmi RSA. RSA:n nimi tulee keksijöiltään Rivestiltä, Shamirilta ja Adlemanilta. Salausalgoritmin ollessa tunnettu sitä vastaan on ollut monia hyökkäyksiä ja tämä työ keskittyy tuomaan katselmuksen sivukanavahyökkäyksiin RSA:n algoritmia vastaan. Sivukanavahyökkäykset ovat hyökkäyksiä, jotka eivät keskity itse algoritmin hajoittamiseen, vaan keskittyvät keräämään tarvittavaa dataa salauksen oikeaoppiseen purkamiseen. Erilaisia lähteitä ovat esimerkiksi aika, äänet, sähkönkulutus tai sähkömagneettinen säteily, jota syntyy tietokoneen purkaessa algoritmiä. Näistä lähteistä saadulla tiedoilla tutkijat ovat onnistuneesti purkaneet RSA:n salauksen.

Tämä työ on jaoteltu kolmeen osaan. Ensimmäisenä on yleiskatsaus RSA:han, kuinka se toimii, miksi sitä pidetään vielä turvallisena ja matematiikkaan itse algoritmissä. Yleiskatsauksen tarkoituksena on tuoda lukijan tietoon RSA:n eri osat ja antaa hieman taustatietoja miksi sivukanavahyökkäykset toimivat. Toisena osana ovat itse sivukanavahyökkäykset. Tämä pitää sisällään miten ne toimivat, kuinka näitä hyökkäyksiä voidaan luokitella sen mukaan mihin hyökkäys kohdistuu ja esimerkkinä työ tuo esille vuonna 2017 julki tulleet hyökkäykset spectre ja meltdown. Kolmannen osan tarkoituksena on yhdistää kaksi aiemmin mainittua osaa ja tehdä selostus siitä, miten hyökkäykset RSA:ta vastaan ovat toimineet ja kuinka puolustautua sivukanavahyökkäyksiä vastaan.

Työn loppupäätelmänä on, että sivukanavahyökkäykset ovat jo osa ja pysyvät osana hyökkäyskenttää. Tämän lisäksi työ tuo esille yleistyksen miten näitä hyökkäyksiä vastaan voidaan puolustautua. Puolustautumista myös avataan hieman esimerkkien avulla ja käydään läpi miten sivukanavahyökkäyksiä vastaan puolustautuminen näkyy käyttäjälle, sillä usemmiten puolustautuminen vaatii lisäresursseja tietokoneelta.

Avainsanat: RSA, side-channel

CONTENTS

1. INTRODUCTION

From the start of the internet, there has been a need to transmit data without other users knowing the contents of the data. One of the oldest and still secure ways of transmitting this data is the public key algorithm Rivest–Shamir–Adleman (RSA). Even as this encryption is said to be secure, there still have been some successful attempts in breaking it. These have been mostly via so-called side-channels. These side-channels are a way to gather data from the cryptographic process via different means to get information on what is happening in it. These means can be time, power draw, noise or any other of multiple different things. This data is usually some sort of difference, for example, time difference. From this difference, the attacker can deduce the right decryption key.

This thesis aims to be a review of the different styles of successful attacks against RSA, how these attacks came to be, how they worked and how these side-channels were and can be defended against.

This thesis will go through three different chapters. Chapter two being the first one focusing on the basic principle of using RSA, how RSA is encrypted, why and how RSA works and how RSA is decrypted. After this, in chapter 3 this paper delves into side-channels, what are side-channels, how side-channels are created and will introduce the three different styles of side-channel attacks, timing-based, access-based and trace-based. And for chapter 4 these two previous topics will be brought together and will cover how the existence of side-channels have made the attacks possible against RSA and how these side-channels helped researchers to crack the RSA encryption. This will be viewed from said three different styles of attack. After the attacks have been covered, this paper will consider the defences against these three styles of attacks before concluding with conclusions.

2. RIVEST–SHAMIR–ADLEMAN

Rivest–Shamir–Adleman shortened to RSA is a widely used public-key cryptosystem for secure data transmission and it is one of the oldest currently in use. The name for this cryptosystem comes from the surnames of Ron Rivest, Adi Shamir and Leonard Adleman the inventors of the algorithm. They described the algorithm publicly in 1977 at the Communications of the ACM [1].

RSA is a public-key cryptosystem. Public-key meaning that one part of the key, usually the encryption key is public information and can be sent to all without the fear of losing security. The other key, called the private key is kept secret. One can decrypt messages encrypted with the public key with the private key and vice versa. This is if a sender would send a message to a receiver encrypted with the receivers public key the receiver can decrypt the message with their private key. This interchangeability of the keys gives us the possibility of sending secure data and the important ability to verify the origin of data if it is encrypted with a private key and if the issuer of the public key is known. This is called signing or signatures. [2]

RSA being one of the oldest and one of the first still secure public-key cryptosystems in use is an important backbone in our current interconnected world. RSA has a part in nearly all current activities online, from credit-card transactions, securing e-mail and authenticating phone calls. [3] With this interconnectivity to our virtual world, it is easy to see why breaking this algorithm is appeasing, especially if one could find a way to break it consistently and within a reasonable time frame. If a proven method would be found of breaking this algorithm, then most of our internet backbone should be reassessed on its security and probably would need at least a change of algorithm. Interestingly enough, we are currently searching for an alternative for RSA or addition to RSA to make it less susceptible against quantum computing as quantum computing gives us numerous different tools against the core of RSA, its math [4].

The algorithm itself is a continuation of Diffie and Hellmans [2] work with their public-key cryptosystem, which presented with an idea of the function that would be a "trap-door one-way function". The "one-way" offers us the knowledge that the function is efficient and easy to compute in one direction, but hard to compute in the other direction. The "trap-door" gives us the knowledge that the message will be easy to compute if the receiver knows certain private information, which is the key to either decrypt or encrypt the message. The idea is not to have a purely secure algorithm, but an algorithm that is computationally infeasible.

2.1 The math of RSA

RSA fundamentally works with two chosen large, distinct prime numbers, that should be chosen with random and a third number that is a combination of these two. The specifics of how this combination is made will be explained below. The security of the two randomly chosen prime numbers comes from the observation that it is very hard and usually not feasible with the current computing capabilities to find these two randomly chosen large prime numbers and is, like Diffie and Hellman [2] suggested infeasible to compute. [1]

Below is the basic idea of the RSA algorithm. Here we have the public key represented by the integers n and e, the private key with the integer d and the message with m.

$(m^e)^d \equiv m$ (mod n) [1] The equivalence sign is denoting modular congruence.

Here knowing all of the other parts of this equation, except for d, it can be extremely hard to find the correct answer for d thus extremely hard to crack the message.

Due to the operation of modulo, we can also exchange the order of the exponentiations and so the following also applies:

$(m^d)^e \equiv m$ (mod n)

That is to say, if the message is encrypted with the public key, it can be easily decrypted with the private key and vice versa.

2.2 Key generation

The key generation starts with the user choosing two distinct large prime numbers, let them be p and q. To create their keys first, the user calculates $n = pq$.

The n that one calculates here is used as a modulus for both the private key and the public key. The length of the n is also the indication of the key length when expressed in bits. This key length is typically between 2048 to 4096 bits. As the key length increases, so does the safety against attacks but so does the time to encrypt and decrypt the message.

After this the user picks the integer d to be a large, random integer which is relatively prime to $(p-1)\cdot(q-1)$, that is that d satisfies the following function:

gcd$(d, (p-1)\cdot(q-1)) = 1$. [1]

Here the "gcd" means the greatest common divisor.

After these have been computed the integer e is to be computed from p, q and d to be the "multiplicative inverse" of d, modulo $(p-1)\cdot(q-1)$. Thus we have

$e \cdot d \equiv 1(\text{mod}(p-1)\cdot(q-1))$.

In this example, our public key would be a pair of positive integers (e, n) and our decryption key the pair of positive integers (d, n). Here even as the n is released as a part of the public key, the factors p and q will remain hidden as the factoring of n is enormously difficult. The same factoring problem also hides the way that d can be derived from e. And as the private key is purely (d, n) it is easy to see why d is to be kept a secret. [1]

2.3 Encryption and Decryption

To encrypt a message M first one must represent the message as an integer between 0 and $n-1$ using standard representation, that is not to encrypt the message, but only to get it into the numeric form necessary for the encryption. This form should be uniform for both the encrypter and the decrypter as if they use different representations the encrypted message will come out as scrambled text.

Then one must encrypt the message by raising it to eth power modulo n. That is, the result (the ciphertext C) is the remainder when M^e is divided by n. [1]

To decrypt the ciphertext, it is to be raised to another power d, again modulo n.

$C \equiv E(M) \equiv M^e(\text{mod } n)$, for the message M.

$D(C) \equiv C^d(\text{mod } n)$, for the ciphertext C.

These operations can be done reasonably quick via different modular exponentiation methods.

3. SIDE-CHANNEL ATTACKS

This work will condense the side attacks to the parts that conclude inside the chosen scope, that is within the decryption of wanted secrets that were encrypted with RSA. This will remove some of the available and possible attacks and such shouldn't be taken as a list of all attacks possible on a system.

As computers are physical machines, these will have physical properties. These properties include but are not limited to time, sound, electricity draw and electromagnetic radiation. All of these properties can be used by an attacker on a system to reveal the internal states of algorithms and so they may leak critical information about the system and the data that it is using and storing. In addition to this, the processor may leak critical information about the cache that it is in charge of, more about this later. Side-channel attacks generally do not focus on compromising the chosen algorithm itself, but the leakage of information that can be gathered from the algorithm via means that aren't usual for the hardware or software itself. This information then can be used to crack the secured algorithms. As the attacks get only pieces of information, for example, the timing of a part of the algorithm, the attackers require multiple probes to gather all the required information for exploiting the system.

Side-channel attacks, in general, can be classified as Szefer [5] did in three main categories. These categories are timing-based, access-based and trace-based attacks.

3.1 Timing-based

Timing-based attacks rely on the timings and the differences of various operations to leak information. These timing differences may appear for various reasons, but especially in recent years, the processor manufacturers aim to optimize their products have given birth to various timing differences. One example of these different timings is the difference between execution trees of correct and incorrect inputs. This is if the correct tree has for example less timing intensive calculations and the incorrect one has more intensive, the attackers may gather information from this difference. One way of forcing these timings to appear to an attacker is by forcing the processor to operate multiple operations, for example, multiple memory accesses in a short amount of time. This will cause the other processes to be slowed down and the timings can be found out with better clarity. [5]

3.2 Access-based

Access-based attacks rely on accessing information directly. Even though some access is required for these attacks to be viable, usually these do not require the attacker to have full access to the machine. These attacks might require for example the access to run their code in a less privileged user and using this code in different ways to open the system or to gather vital information of the system that is running. One of the most recent and newsworthy of these kinds of access-based attacks was the Spectre [6] and Meltdown [7] attacks. These attacks will be explained at the end of this chapter. Usually, these use the latency of the cache to determine if the data was a hit or a miss in the cache, which then clues the attack in on what is being used and thus help with opening the attacked system [5].

3.3 Trace-based

Trace-based attacks rely on the exact execution of a program and the measuring of it. The attacker in question might obtain the sequence of memory accesses that the encryption or decryption uses and whether they are hits or misses in the cache based on the power usage, sound differences or other ways. [8] Usually these attacks require the attacker to be present on the attacked machine.

Even as these attacks are presented alone as attacks, one should notice that multiple side-channels can be combined into a single attack. One of these most recent and most powerful combined attacks would be the Meltdown [7] and Spectre [6] attacks that were introduced in 2017. These attacks used a combination of timing-based and access-based attacks to gather data from within the victims' machine to the data from the kernel.

3.4 Spectre and Meltdown

Spectre is a family of attacks that work on inducing the victims' machine to speculatively execute operations. This speculative execution is done to get more performance out of the CPU and it is done by every major manufacturer. These speculative executions in spectre aim to violate memory isolation boundaries and such read data that shouldn't be available for the attacker. Usually, speculative execution isn't visible to anyone and is discarded when this speculative execution is seen not to be needed. Here is where the spectre attack uses another side-channel attack to transmit the data from the cache to usable data. [6]

Meltdown is a related attack to the spectre family, except meltdown uses so-called out-of-order execution. Out-of-order execution is where the CPU predicted the branching execution and ran the predicted before verifying the execution. The out-of-branch execution was exploited by trying to access inaccessible kernel addresses causing an exception. This exception is raised and should stop the execution of the following commands. But as the CPU uses out-of-order execution, there is a possibility that the CPU might have already executed the following instructions. These shouldn't cause any harm as there are no architectural effects on the CPU, but as the commands have already been executed there will be microarchitectural effects. As the inaccessible kernel addresses have been read and so cached, there is a possibility to use another attack to detect what cache is used by using a timing difference on the cache read times. This data is then collected with another side-channel attack, also an access based attack. The meltdown attack used similar techniques to the "Flush+Reload" attack, where they successfully transferred the data from the cache to the accessible memory. [7]

4. ATTACKING AND DEFENDING

As we now have a general understanding of RSA and the side-channel attacks, three differing attacks will be examined. Firstly there will be a discussion of the attacks in general against RSA with side-channels, then specify the chosen three attacks which correlate to the three different classifications that were outlined in chapter 3.

After the information of the attack styles and the weakness that the attacks focused on, we'll move on to defending against these attacks, what can be done to migrate the potential of these side-channels, and quickly touch upon the similarities of these attacks, that is what was the culprit and how we might be able to see these before these happen.

4.1 Attacks against RSA

As RSA is seen as being secure with a long enough key length, the attacks on it are not focused on compromising the algorithm itself, but on the inherent faults of machines running the algorithm, the people that create the code, the optimizations that are made inside the machine and the fact that the current machines running have their computing units, memory and other components shared to multiple users. [9]

One of the most notable creators of side-channel attacks has been the optimization that is being done within the processors. This optimization has lead to more than one way of exploiting the processor. Firstly the processor manufacturers have unwittingly created so-called fast and slow execution paths, this is a simple addition that might take much less time to execute than a harder to run multiplication operation. [9] This simple timing difference will cause the attacker that can call the decryption algorithm to have a clue on what operations the defenders processor is running at that time. This will lead to the attacker being able to deduce the different paths that the processor is going through. Armed with a knowledge of the processor and the knowledge of the algorithm that is being run on it attackers can with multiple probing of the machine can piece together when they are close to the proper key.

Other optimizations that cause side-channel attacks to appear can be seen in the current predictive computing. As the requirement for faster computing has steadily risen, most manufacturers of processors have optimize their processors what operations usually follow other ones and what part of the cache is usually called after each operation. [10] This optimization has provided malicious actors with a way to see what the correct answer will be and so cause weaknesses in the encryption without attacking the algorithm itself [11].

4.1.1 Attack one: Timing-based attack

As described above, some attacks take the fact that different operations take different times to compute on the processor. The reasons include, but are not limited to optimizations, branching, conditional statements, cache and instruction differences. This difference can be used as described by Kocher [9] and demonstrated by Brumley and Boneh [12] to crack the encryption. In short, these kinds of attacks require the attacker to be aware of the information of the receiving system, that is the way that RSA is being used and what paths the encryption and decryption take and how they differ. The attackers start by guessing a key to the system and by monitoring the difference between the call and response can deduce how close the guessed key is to the correct key. With this the attackers can start deciphering the key from the gathered information from the most significant bit to the least. [9]

This was the way that OpenSSL calculated the same operation with two different algorithms, Karatsuba and normal multiplications depending on the input it received. And this caused it to be a viable target to side-channel attacks. A more specific explanation can be found from Brumleys [12] paper.

4.1.2 Attack two: Access-based attack

Access-based attacks such as cache attacks focus on the fact that most computers being used are used with multiple users and so the processing is happening simultaneously with all the users. This leads to the processor using its L1 cache, which will be filled with the knowledge of the operations called by the user that is being attacked and so the attackers can filter out what is and isn't being used by the user, as the used instructions will be in the cache [10]. This will create a disparity between uncalled instruction call time and a called instruction call time as the already called instruction will be in the cache and the uncalled will have to be loaded from memory.

If the attackers can then "flush" the cache, that is to clear the cache before the victim uses their system, the attackers can then decipher what was used. With a few decryptions the attackers can decode the wanted key from the user's system. [13] This was the way that researchers Zhou et. al. [13] got the RSA encryption key with their paper.

4.1.3 Attack three: Trace-based attack

For trace-based attacks, the example was chosen to be an acoustic attack on RSA as specified by Genkin et. al. [8]. This attack requires, as most trace-based attacks require, physical interaction or observation of the device. In this case, the observation of the device is done via sound waves.

In their attack, Genkin et al. [8] noticed that the computer they were attacking emitted constantly a high-pitched noise during operation, which was caused due to vibrations in the electronic components. The team then caused the attacked computer to cycle through the encryption and decryption process multiple times whilst listening via a microphone next to the computer. This microphone output was captured and analyzed to see where GnuPG implementation of RSA switched between exponentiation modulo of the secret p to the exponentiation modulo of the secret q. The differing noise was then attributed to the same differences in decryption and encryption as seen in the Brumleys and Bonehs [9] paper and so the same techniques on guessing and applying the leaked information can be used to crack the key.

4.2 Defending against side-channel attacks

Most of the defences against side-channel attacks of all classifications are the migration of the part that causes the differences, whatever the differences may be [5]. The defences might include standardizing the actions that the decrypting algorithm does whether it is given the correct key or the incorrect one [12], revoking other users access to the machine that is being attacked, flushing the cache itself after decrypting [13] or shielding the whole attacked machine within a sound-proof, electromagnetically sealed cage and causing the machine to draw constant load from the power line [8].

These defences do come at a cost, as there might need to be the need to remove the optimization to reduce the differences, the defending against side-channels might cause a drop in performance [5]. But some may require extra security over the most optimal performance, especially when dealing with critical information such as currency or personal data.

4.2.1 Attack one: Timing-based attack

As Brumley and Boneh [12] wrote in their paper, the defence against timing-based attacks is a standardization of the timing or randomization of the timing. They suggest a technique called RSA blinding, which calculates $x = r^e g$ mod N before decryption, where r is random, e is the RSA encryption exponent, and g is the ciphertext to be decrypted, followed by division by r. As the r is randomly chosen here, this will cause the timing of the decryption not to be linked with the decryption key and so their chosen attack vector will be negated.

Suggestions from Brumley and Boneh [12] included the quantization of the RSA computations as suggested by Wenxue et. al. [14] in their paper, which will cause the calculations to take the same time. Other suggestion was the modification of existing code via causing it to calculate all the required operations, even if the given key is correct. Though Brumley and Boneh [12] speculated that this might be optimized away by the compiler.

As noted in section 4.2 all of the defences against different attacks will cause a drop in performance, with RSA blinding causing a calculated 2% - 10% drop in expected performance.

4.2.2 Attack two: Access-based attack

As access-based attacks require access to the machine that is being attacked to be able to read the cache and modify it to the needs of the attacker, the easiest way of defeating these attacks would be the virtualization of the machines with one user per machine. If multiple access to a machine is required Percival [11] suggested other ways of defending against this data leak, most including changes in the way that cache is handled. One is to cache eviction strategy, this is to make the cache eviction logic in the CPU be made thread-aware and only to allow the cache owned by the thread in question to evict it. Other ways are using the operating systems kernel scheduler to cause the cached data to be safe from tampering and the standardization of the code path and sequence of memory access, whether it uses the cache or not. Other suggested methods included the rewriting of some existing algorithms or in the case of the "sliding window" method of modular exponentiation to be thrown out.

All of the suggested defences for access-based attacks cause some performance drop, for example, the rewriting of the existing algorithms for RSA can be as little as 10%. But as this multi-threading is an integral part of computing, in the case of the kernel thread-aware suggestion, this might cause unwanted bugs and locking of the kernel data as the credentials of the thread can be changed during a system call. [11]

4.2.3 Attack three: Trace-based attack

Trace-based attacks can seem the most trivial to defend against. As these usually require physical access to the machines in question a simple lockdown of the machine, the power supply and emissions should be sufficient to close this side-channel. [8]

There is still some consideration to be had against attacks that target the power-draw as it can be monitored outside of this imaginary Faraday's cage. But the machines power-draw can be hidden within the general noise of a buildings power supply.

4.3 Defending against side-channels and future

As we have discovered, most of the attacks require a difference of some sort to be viable attacks. With this in mind, the easiest way of defeating these attacks would be the discovery and destruction of these differences, whether they might be sound, power draw, timing or other.

Even though the attacks were discussed one by one, as said in chapter 3, these attacks can be combined. Schindler [15] researched the possibility of combining a Timing-based and a Trace-based attack to create a sturdier, better attack with fewer measurements required to crack open the key. This idea of combination attacks seems to be the future of side-channels as we could see in chapter 3 with the attacks Meltdown and Spectre, which used multiple side-channels to create a more efficient attack against the whole system. These mentioned attacks were patched out with software updates and with later CPU's architectural reviews. Both of these brought with them a noticeable impact on performance. [6] As a note, the timing defence of quantization of the computations may not be enough against combination attacks as the different commands draw a differing amount of power and such will cause the data to appear to the attackers.

There is an interesting dilemma here to be seen, the constantly growing requirement of processing power and performance from the machines and the security that will cause the performance of the machine to drop. The requirement of this performance has caused these side-channels to appear to the defender, whether it may be a faster way to approximate large primes or the invention and use of predictions in the processor.

5. CONCLUSIONS

As the technologies for secure transfers of data come into use, so do the attackers methods of cracking these secure methods of data transfer. But as we saw in this thesis, most if not all of the methods that have been used to crack the RSA cryptosystem can be defeated with proper planning of the systems, algorithms and environments used. As is, we can say that the RSA is still and should be until the arrival of the quantum computers a safe and sound cryptosystem with sufficient key length. There is still the question of how much processing power should be reserved from the system for inefficient ways of handling data for the quest of better security. As we saw in chapter 4, each security defence will raise the requirements from the machine using it. Usually, this isn't much but as these add on, these might cause some performance deficiencies on different hardware.

This was a review of the different styles of successful side-channel attacks against RSA looked through the different research papers, how they managed to become into existence and how these attacks were dwarfed with new and improved defences. The opening of the side-channels were the results of either the optimization and the search for the best performance or the realities of the physical world. The attacks themselves focused on the differences that the system leaked with different inputs, whatever these differences might have been. The defence with RSA was and still seems to be eliminating these differences, wherever the differences appear.

The work itself would've enjoyed a more comprehensive look at the specifics of the exploits, such as the math behind the timing-based side-channel attacks and more examples of the different styles of side-channels attacks, with each examined thoroughly and compared to another. Future research should and is being done on deepening the understanding of different side-channels, uncovering more of them and there is some promising research being made with machine-learning for detecting possible side-channels before they can be exploited by malicious actors.

REFERENCES

[1] Rivest, R. L., Shamir, A. and Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM* 21.2 (1978), pp. 120–126.

[2] Diffie, W. and Hellman Martin, E. New Directions in Cryptography. *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

[3] Robinson, S. Still Guarding Secrets after Years of Attacks, RSA Earns Accolades for its Founders. *SIAM News* 36 (2003), pp. 1–4.

[4] Bernstein, D. et al. Post-quantum RSA. *PQCrypto 2017. Lecture Notes in Computer Science* 10346 (2017), pp. 311–329.

[5] Szefer, J. Survey of Microarchitectural Side and Covert Channels, Attacks, and Defenses. *Journal of Hardware and Systems Security* 3 (2018), pp. 219–234.

[6] *Spectre Attacks: Exploiting Speculative Execution*. 2018. URL: https://spectreattack.com/spectre.pdf.

[7] *Meltdown: Reading Kernel Memory from User Space*. 2018. URL: https://meltdownattack.com/meltdown.pdf.

[8] Genkin, D. et al. *RSA Key Extraction via Low-BandwidthAcoustic Cryptanalysis*. Tel Aviv University. Tel Aviv, Israel, 2013. URL: https://www.tau.ac.il/~tromer/papers/acoustic-20131218.pdf/.

[9] Kocker, P. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. *Koblitz N. (eds) Advances in Cryptology — CRYPTO '96. CRYPTO 1996. Lecture Notes in Computer Science* 1109 (1996), pp. 104–113.

[10] Malishevsky, A. et al. *Dynamic Branch Prediction*. Course material for ECE 570 High Performance Computer Architecture. Oregon State University. Oregon, 2019. URL: https://web.archive.org/web/20190717130447/http://web.engr.oregonstate.edu/~benl/Projects/branch_pred/.

[11] Percival, C. *Cache Missing for Fun and Profit*. 2005. URL: http://www.daemonology.net/hyperthreading-considered-harmful/.

[12] Brumley, D. Remote timing attacks are practical. *Computer Networks* 48.5 (2005), pp. 701–716.

[13] Zhou, P. et al. Analysis on the Parameter Selection Method for FLUSH+RELOAD Based Cache Timing Attack on RSA. *China Communications* 12.6 (2015), pp. 33–45.

[14] Wenxue, T. et al. A mechanism of quantitating the security strength of RSA key. *2010 Third International Symposium on Electronic Commerce and Security* (2010), pp. 357–361.

[15] Schindler, W. A Combined Timing and Power Attack. *Public Key Cryptography. PKC 2002. Lecture Notes in Computer Science* 2274 (2002), pp. 263–279.