Open-source RTP Library for End-to-End Encrypted Real-Time Video Streaming Applications

Joni Räsänen, Aaro Altonen, Alexandre Mercat, and Jarno Vanne

Tampere University, Tampere, Finland
{joni.rasanen, alexandre.mercat, jarno.vanne}@tuni.fi

***Abstract*— Information security has become a key success factor for streaming media applications that are increasingly vulnerable to wiretapping, message forgery, data tampering, hacking, and other possible cyberattacks. This paper addresses the existing security risks in real-time video streaming by introducing a new security extension to our uvgRTP open-source Real-time Transport Protocol (RTP) library. The proposed solution improves content integrity and privacy by adopting Secure RTP (SRTP) and Zimmermann RTP (ZRTP) for media End-to-End Encryption (E2EE). These new security mechanisms make uvgRTP the first open-source library that supports on-the-fly encrypted AVC, HEVC, and VVC video streaming. Our performance results on Intel Core i7-4770 processor show that uvgRTP is able to transport encrypted 8K VVC video at up to 187 fps and 8K HEVC video at up to 120 fps over a 10 Gbps Local Area Network (LAN). The achieved transfer rate for encrypted HEVC video is 50% higher and latency 86% lower than the respective performance values of FFmpeg in unencrypted HEVC streaming. These top streaming speed results with state-of-the-art video codec support, advanced encryption mechanisms, and the permissive BSD license make uvgRTP an attractive solution for a broad range of commercial and academic streaming media applications.**

Keywords—Real-time Transport Protocol (RTP), Versatile Video Coding (VVC), High Efficiency Video Coding (HEVC), Secure RTP (SRTP), Zimmermann RTP (ZRTP)

I. Introduction

Recent advances in video transport and coding technologies have led to a proliferation of interactive and live streaming media applications. In the existing solutions, real-time video streaming is typically implemented with *Real-time Transport Protocol* (*RTP*) [1] using standard video coding formats such as *Advanced Video Coding* (*AVC/H.264*) [2] or *High-Efficiency Video Coding* (*HEVC/H.265*) [3]. In addition, the latest video coding standard, *Versatile Video Coding* (*VVC/H.266*) [4], will gain ground in the future streaming applications.

High video quality, high frame rate, low frame loss, and low end-to-end latency are the key performance indicators of user experience, but without any protection, the video content is vulnerable to attackers when it involves sensitive or private data. Media stream encryption enables keeping personal or business information out of the hands of undesirable actors.

Secure RTP (*SRTP*) [5] is an extension to RTP and the most common approach for transporting encrypted media in *Real-Time Communication (RTC)* applications [6]. However, SRTP does not specify methods for encryption key management. Therefore, it is often used with *Zimmermann RTP* (*ZRTP*) [7] that fulfils the requirements for media *End-to-End Encryption* (*E2EE*) by providing key negotiation and management capabilities for SRTP. In E2EE, only the stream sender and receiver are able to decrypt the traffic. This reduces the attack surface on the encryption by preventing the security key exchange provider from decrypting the media stream.

Multiple open-source SRTP libraries [8]–[14] have been released over the last decades, but none of them support the state-of-the-art VVC standard. The full-fledged *GStreamer* [13] and *FFmpeg* [14] multimedia frameworks have built-in support for HEVC and AVC, but they are not appropriate for lightweight applications striving for maximum performance. Furthermore, only *ccRTP* library [12] comes with ZRTP, but the GPL license makes it less attractive for commercial applications.

Our *uvgRTP* library [15][16] has built-in support for VVC, HEVC, and AVC video codecs and *Opus* audio codec [17] that can be seen as key enablers of economic video and audio transmission. It also provides an easy-to-use *Application Programming Interface* (*API*) for introducing other user-defined RTP payload formats.

In this work, we implemented SRTP and ZRTP extensions into *uvgRTP* and made it compatible with end-to-end encrypted media streaming. As of now, applications can utilize *uvgRTP* to stream encrypted VVC, HEVC, and AVC videos in real time. *uvgRTP* is available online at

https://github.com/ultravideo/uvgrtp.

It is written in *C++* and distributed under the permissive BSD 2-Clause license. This cross-platform library can be run on both Linux and Windows operating systems.

The rest of the paper is organized as follows. Section 2 provides an overview of end-to-end encrypted video streaming with SRTP and ZRTP protocols. Section 3 introduces the ZRTP and SRTP extensions to our *uvgRTP* library. Section 4 evaluates the encryption performance in comparison with *FFmpeg* and Section 5 concludes the paper.

II. End-to-End Encrypted Media Streaming

RTP, specified in *RFC 3550* [1], defines the general-purpose RTP packet format for real-time media transfer and an associated *RTP Control Protocol* (*RTCP*) for *Quality-of-Service* (*QoS*) monitoring and RTP session management. Besides *RFC 3550,* video E2EE calls for other specifications that define RTP payload formats for different video coding formats, bitstream encryption, and encryption key management.

A. VVC, HEVC, and AVC Streaming over RTP

The draft specification of the RTP packet format for VVC [18] describes rules for VVC bitstream packetization and de-packetization. When streaming VVC over RTP, the VVC bitstream is divided into *Network Abstraction Layer* (*NAL*) units. A single VVC frame may contain multiple NAL units and each NAL unit begins with a start code. The RTP packet format specifications for HEVC [19] and AVC [20] follow the same principles.

The NAL units are further split into *Fragmentation Units* (*FUs*) having a size of an ethernet frame payload. This approach removes IP level fragmentation and thereby

TABLE I
FEATURES OF EXISTING OPEN-SOURCE SRTP LIBRARIES

Ref. Library	ZRTP	VVC	HEVC	AVC	LoC	License
[8] libre	No	No	No	No	58k	BSD
[9] PJSIP	No	No	No	Yes	360k	GPL-2.0
[10] libsrtp	No	No	No	No	23k	BSD
[11] JRTPLIB	No	No	No	No	28k	MIT
[12] ccRTP	Yes	No	No	No	14k	GPLv2
[13] GStreamer	No	No	Yes	Yes	3062k	LGPLv2.1
[14] FFmpeg	No	No	Yes	Yes	1250k	LGPLv2.1
uvgRTP	**Yes**	**Yes**	**Yes**	**Yes**	**13k**	**BSD-2**

improves the reliability of transmission because any lost IP fragment would result in a lost frame.

B. SRTP Media Streaming

SRTP, specified in *RFC 3711* [5], defines operating principles to cipher and authenticate RTP streams. It is also coupled with *Secure RTCP* (*SRTCP*), which is an encrypted and authenticated version of RTCP.

The use of SRTP requires implementing *Advanced Encryption Standard* (*AES*) [21] with at least 128-bit key length in *Counter Mode* (*CM*) for ciphering and *Secure Hash Algorithm 1* (*SHA-1*) [22] for *Hash-based Message Authentication Code* (*HMAC*) [23]. The ciphering is performed for the whole bitstream and is computationally intensive, but when used together with HMAC, the 128-bit AES key length offers sufficient level of encryption for any practical application to the foreseeable future.

For ciphering, SRTP needs the master key and salt, which can be generated and managed with the following protocols: ZRTP [7], *Multimedia Internet KEYing* (*MIKEY*) [24], *Session Description Protocol Security Descriptions* (*SDES*) [25], and *Datagram Transport Layer Security Extension to Establish Keys for the SRTP* (*DTLS-SRTP*) [26]. However, only ZRTP and DTLS-SRTP can be used to implement E2EE. DTLS-SRTP relies on public key infrastructure which makes it more vulnerable to *Man-in-the-middle* (*MITM*) attacks than ZRTP.

C. ZRTP Key Management

ZRTP [7] defines cryptographic key management to securely establish a shared cryptographic context from a cached secret or by performing a *Diffie-Hellman* key exchange. The shared context is used to derive the master key and salt, which are, in turn, applied to create the session key and salt for the stream encryption and decryption. ZRTP increases the possibility of detecting MITM attacks with a feature called *Short Authentication String* (*SAS*).

D. Existing SRTP Libraries

Table 1 tabulates the existing open-source RTP libraries that support SRTP encryption. Each library is characterized by its compatibility with ZRTP (*RFC 6189*) [7] protocol as well as its RTP payload format support for VVC (draft) [18], HEVC (*RFC 7798*) [19], and AVC (*RFC 6184*) [20]. In addition, the *Lines of Code* (*LoC*) in the library and the licensing information are given.

libre [8], *PJSIP* [9], *libsrtp* [10] and *JRTPLIB* [11] do not provide built-in support for VVC, HEVC, or ZRTP. *libre* has been developed by *Creytiv.com*, PJSIP by *Teluu Inc.*, and *libsrtp* by *Cisco*. All these company-led projects are under active development and *libsrtp* is currently one of the most popular SRTP implementations. *JRTPLIB* is no longer under active development.

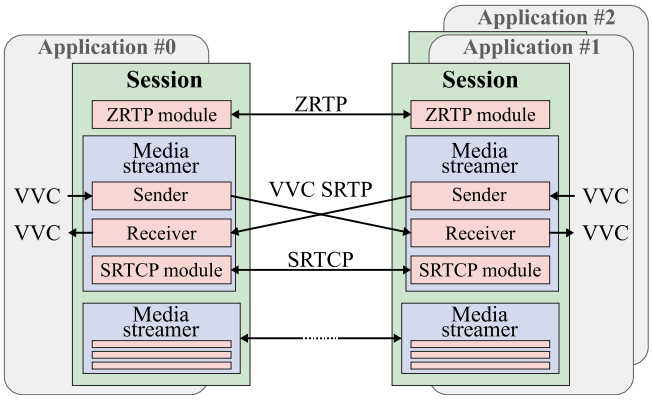

Fig. 1. uvgRTP usage scenario for streaming encrypted VVC.

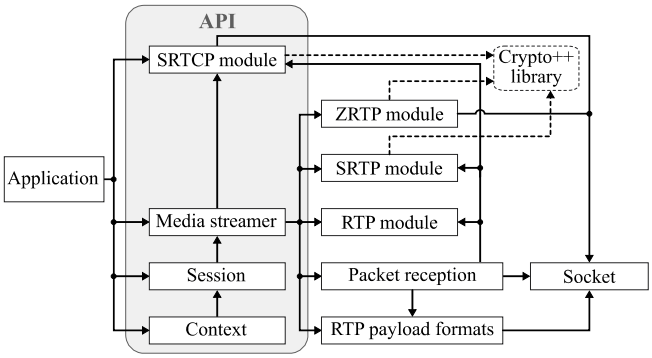

Fig. 2. Software architecture of uvgRTP.

ccRTP [12] is the only existing open-source library that has a built-in support for ZRTP. However, *ccRTP* only provides low-level control over its RTP functionality, which leaves more implementation work for the application designer. *ccRTP* is licensed under GPL v2 and endorsed by *Free Software Foundation*, but its last update was in 2015.

GStreamer [13] and *FFmpeg* [14] are widely used and actively developed multimedia frameworks with media streaming functionality. However, their broad spectrum of usage scenarios makes them less suitable for modest sized projects and for applications that have no need for extensive media processing capabilities.

III. SECURE UVGRTP LIBRARY

uvgRTP has a built-in support for VVC [18], HEVC [19], AVC [20], and Opus [17] payload formats for which it implements encryption via SRTP protocol and encryption key management via ZRTP protocol.

A. Architecture

Fig. 1 depicts the operating principle of *uvgRTP* when applied in two-way point-to-point VVC video communication between different streaming media applications. *uvgRTP* creates a new *session* module for each peer it exchanges media with, and each *session* applies a different *media streamer* module for every audio or video stream. The sending application uses *uvgRTP sender* to transmit video to corresponding *uvgRTP receiver* that passes it on to a receiving application. A single *media streamer* module does not support multithreading, but different *media streamers* can be used from separate threads.

Fig. 2. describes the high-level architecture of our *uvgRTP* library with dependency relations between its components. The *context* is the top-level module, and it is used to allocate

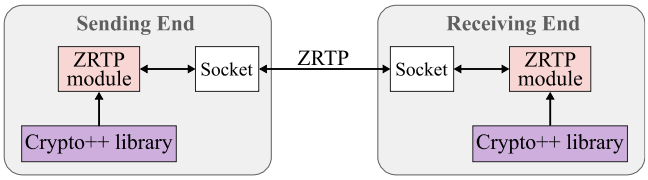

Fig. 3. ZRTP encryption key negotiation in uvgRTP.

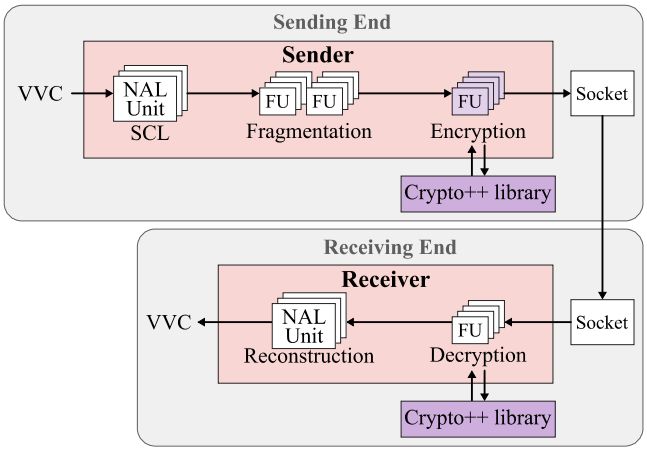

Fig. 4. The E2EE workflow for VVC video streaming in uvgRTP.

separate *session* modules for each IP address. The *session* module creates one *media streamer* module per encrypted SRTP stream for data sending, receiving, or both. The *SRTCP module* manages the SRTCP traffic of the corresponding SRTP stream.

The *RTP module* upholds the state of the SRTP stream whereas *SRTP module* takes care of stream encryption and decryption. The *packet reception* module controls the processing of received packets. The *RTP payload formats* module implements the format specific *Start Code Lookup* (*SCL*), fragmentation, and reconstruction of the stream. The *socket* module handles the transmission and reception operations on the system socket. The *Crypto++ library* [27] is used by the *SRTCP module*, *ZRTP module,* and *SRTP module* for encryption and decryption tasks.

B. ZRTP Implementation

Encrypted media streaming starts with the negotiation of the cryptographic context. For the negotiation, *uvgRTP* uses ZRTP that makes it possible for the application to implement E2EE without having to pay attention to the encryption key management.

When creating the first *media streamer* instance in a *uvgRTP session*, the *ZRTP module* (see Fig. 3) performs a handshake to establish a shared cryptographic context between the peers. The ZRTP protocol is used in the *Diffie-Hellman* mode, where the *ZRTP module* makes use of *Crypto++ library* [23] to generate a private/public key pair and then performs the *Diffie-Hellman* key exchange. All *media streamer* instances within the same *session* are in the *Multistream* mode, in which the *ZRTP module* uses the already negotiated context to re-negotiate a new context for each new *media streamer* instance.

C. SRTP Implementation

The *media streamer* module can start sending and/or receiving encrypted media after the *ZRTP module* has established the cryptographic context for it. Each *media streamer* instance uses the same *socket* module as the corresponding *ZRTP module*.

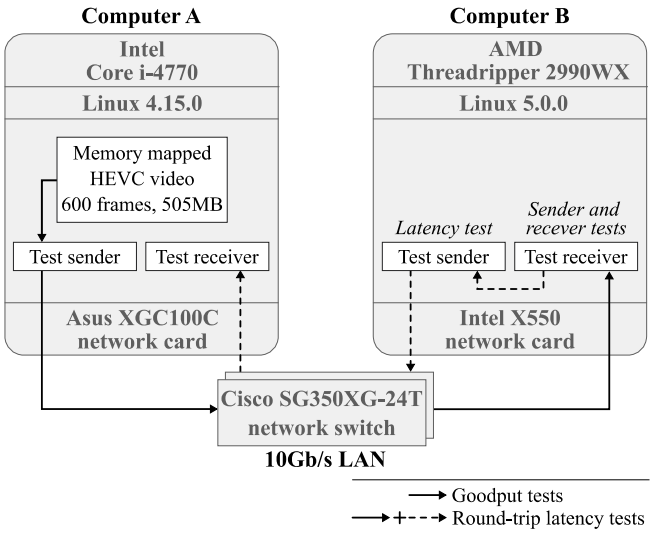

Fig. 5. Experimental setup.

Fig. 4 illustrates the E2EE workflows of the SRTP *sender* and *receiver* for VVC streaming. First, the application delivers the VVC input frame to the *media streamer*. The *SCL* stage splits the frame into NAL units, and the *fragmentation* stage divides each NAL unit into FUs. When all NAL units have been fragmented, the *encryption* stage uses *Crypto++* to encrypt the FUs using 128-bit AES in CM. In the receiving end, the FUs are read from the *socket* module and decrypted using *Crypto++ library*. Finally, the NAL units are reconstructed from the FUs before they are sent to the user.

uvgRTP can further improve protection by calculating an RTP authentication tag with *Crypto++*. The *sender* appends the tag at the end of each FU, and the *receiver* verifies the tags and discards all invalid FUs.

IV. Performance Analysis

In our experiments, *uvgRTP* was benchmarked against *FFmpeg* version 4.3 that represents the state-of-the-art in RTP streaming and media processing. However, SRTP functionality of *FFmpeg* was excluded from our evaluations because it does not include key negotiation. In addition, *GStreamer* was completely omitted because there was no straightforward way to integrate its closely-knit media processing filters into our benchmark setup. The other considered libraries [8]–[12] were also left out because none of them support HEVC or VVC video streaming. The benchmarking framework used in our performance analysis is available online[1].

A. Experimental Setup

Fig 5. illustrates the experimental setup. The sender computer (*computer A*) was equipped with an *Intel Core i7-4770,* Linux kernel version 4.15.0, and *Asus XCG100C* network card. The receiver computer (*computer B*) had an *AMD Threadripper 2990WX*, Linux kernel version 5.0.0, and *Intel X550* network card. Both computers were connected over a 10 Gbps *Local Area Network* (*LAN*) with two *Cisco SG350XG* network switches in between them.

[1] https://github.com/ultravideo/rtp-benchmarks

TABLE II
SPEED, FRAME LOSS, AND SENDER CPU USAGE OF UVGRTP AND FFMPEG

Library	SRTP	Format	Frame rate	Frame loss	CPU utilization
uvgRTP	Yes	VVC	749	0.04%	98.4%
	Yes	HEVC	500	0.03%	99.3%
	No	VVC	1200	0.03%	98.1%
	No	HEVC	840	0.04%	98.8%
FFmpeg	No	HEVC	324	2.70%	62.7%

TABLE III
ROUND-TRIP LATENCIES OF UVGRTP AND FFMPEG

Library	SRTP	Format	Failed runs	Intra (ms)	Inter (ms)	Average (ms)
uvgRTP	Yes	VVC	14%	23.9	6.3	7.4
	Yes	HEVC	20%	22.1	11.0	11.2
	No	VVC	11%	14.5	3.8	4.4
	No	HEVC	7%	15.1	7.1	7.3
FFmpeg	No	HEVC	52%	108.3	80.6	81.0

The tests were performed with one 4K120p (3840×2160 pixels) test video sequence that was encoded into HEVC and VVC bitstreams. The encoded bit rates were 660 and 442 Mbit/s for HEVC and VVC bitstreams, respectively. To mitigate the additional latency caused by file I/O operations, the sequences were memory-mapped to the address space. In addition, the Linux network stack parameters were adjusted according to the amount of data as in our previous work [15].

Two separate test cases were evaluated. In both test cases, a single thread was used to send the same test video 100 times in a row and average performance results were reported. If the tests consisted of multiple simultaneous streams run on separate cores, we would expect the per stream results to be similar if the network capacity is not exceeded.

In the first test case, the maximum frame rate, frame loss, and CPU utilization of *uvgRTP* and *FFmpeg* were measured as a function of transfer speed by increasing the data transfer rates (i.e., frame rates) of VVC and HEVC test videos from 100 *frames per second* (*fps*) to 1200 fps in steps of 100 fps. In the second test case, the round-trip latency of the system was evaluated. Altogether, *uvgRTP* was benchmarked with four payload formats: 1) encrypted VVC; 2) encrypted HEVC; 3) unencrypted VVC; and 4) unencrypted HEVC. The obtained results were compared with those of unencrypted HEVC streaming with *FFmpeg*.

B. Frame Rate, Frame Loss, and CPU Utilization

Table 2 shows maximum attainable frame rates of the *uvgRTP* and *FFmpeg* libraries for 4K HEVC and VVC test videos. The frame rate values show that encryption of brings around 60-70% overhead compared to streaming unencrypted video. In addition, the respective percentages for frame loss and sender CPU usage are reported. The frame loss of uvgRTP stems from the UDP packet loss. With lower frame rates, the CPU usage was proportional to the frame rate but the frame loss stays the same at all frame rates.

The data transfer rate of *uvgRTP* approximately corresponds to streaming encrypted 8K VVC at 187 fps and HEVC at 125 fps. Since the encrypted streaming performance of *uvgRTP* is greater than the unencrypted streaming speed of *FFmpeg*, we can conclude that *uvgRTP* would also achieve higher data transfer rate than encrypted *FFmpeg* since the encryption comes with significant overhead.

One reason behind the superiority of *uvgRTP* is the higher CPU utilization. In addition, *uvgRTP* is solely designed for high-performance streaming, e.g., by minimizing data copies whereas *FFmpeg* provides an entire framework for multimedia processing.

C. Latency Evaluation

The second test case measured the round-trip latencies of *uvgRTP* and *FFmpeg*. The tests were performed at a frame rate of 120 fps by sending the HEVC and VVC test videos from the *computer A* to *computer B* and back. Measuring the latency of entire frames rather than individual packets better reflects the actual latency of the video seen by the user.

Table 3 reports the round-trip latencies (in ms) of *uvgRTP* and *FFmpeg*. They were measured for intra and inter frames separately as well as averaged. Only test runs without any lost packets were included in the results, because larger frames are more likely to have dropped packets leading to their exclusion from the result averages. The *Failed runs*-column stands for percentage of excluded test runs because of lost packets.

The results show that SRTP increases the latency by 50–70% compared with unencrypted RTP, but it remains relatively low at under 12 ms. The average latency of *FFmpeg* is significantly higher even without encryption.

V. CONCLUSION

This paper presented an E2EE extension to our *uvgRTP* open-source RTP library. Unlike the other open-source approaches, the latest version of *uvgRTP* supports SRTP protocol for encrypted AVC, HEVC, and VVC video streaming and ZRTP protocol for the cryptographic key exchange. Hence, it is the first streaming-oriented RTP library that supports encryption along with video coding formats. The small library size, permissive BSD license, and built-in payload formats make uvgRTP a convenient option for any commercial and academic projects dealing with state-of-the-art encrypted video transmission.

According to our experiments, *uvgRTP* is capable of transferring encrypted 8K VVC and HEVC videos in real time at up to 187 fps and 120 fps, respectively. Even with encryption, it achieves 54% higher frame rate and 86% lower latency than *FFmpeg* does for unencrypted HEVC video. These performance results indicate that *uvgRTP* is a potential candidate for practically any kind of current and future secure real-time streaming media solutions. The potential applications include interactive video communication platforms, remote presence services, and multimedia-oriented Internet of Things, where high speed, low latency, and security are of the essence.

In the future, *uvgRTP* will be extended with RTP header extension encryption. In addition, Elliptic-curve *Diffie-Hellman* key negotiation modes will be implemented to make the key exchange computationally less demanding.

ACKNOWLEDGMENT

This paper is part of the PRYSTINE project that has received funding within the ECSEL JU in collaboration with the European Union's H2020 Framework Programme (H2020/2014-2020) and National Authorities, under grant agreement 783190.

REFERENCES

[1] IETF. *IETF RFC 3550 RTP: a Transport Protocol for Real-Time Applications.* (2003). [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3550

[2] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H. 264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.

[3] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol*., vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[4] International Telecommunication Union (ITU). *New Versatile Video Coding Standard to Enable Next-Generation Video Compression*. [Online]. Available: https://www.itu.int/en/mediacentre/Pages/pr13-2020-New-Versatile-Video-coding-standard-video-compression.aspx

[5] IETF. *IETF RFC 3711 the Secure Real-Time Transport Protocol (SRTP)*. (2004). [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3711.

[6] A. Nisticò, D. Markudova, M. Trevisan, M. Meo, and G. Carofiglio, "A comparative study of RTC applications," in *Proc. IEEE Int. Symp. Multimedia*, Naples, Italy, Dec. 2020.

[7] IETF. *IETF RFC 6189 ZRTP: Media Path Key Agreement for Unicast Secure RTP*. (2011). [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6189.

[8] Creytiv.com. *libre*. [Online]. Available: https://github.com/creytiv/re.

[9] Teluu Ltd. *PJSIP*. [Online]. Available: https://www.pjsip.org/.

[10] Cisco Systems Inc. *libsrtp*. [Online]. Available: https://github.com/cisco/libsrtp.

[11] J. Liesenborgs. *JRTPLIB - End of Development*. [Online]. Available: https://research.edm.uhasselt.be/jori/page/CS/Jrtplib.html.

[12] GNU. *GNU ccRTP*. [Online]. Available: https://www.gnu.org/software/ccrtp/.

[13] *GStreamer*. [Online]. Available: https://gstreamer.freedesktop.org/.

[14] *FFmpeg*. [Online]. Available: https://www.ffmpeg.org/.

[15] A. Altonen, J. Räsänen, J. Laitinen, M. Viitanen, and J. Vanne, "Open-source RTP library for high-speed 4K HEVC video streaming," *in Proc. IEEE Int. Workshop Multimedia Signal Process.*, Tampere, Finland, Sept. 2020.

[16] A. Altonen, J. Räsänen, A. Mercat, and J. Vanne, "uvgRTP 2.0: open-source RTP library for real-time VVC/HEVC streaming," *in Proc. IEEE Int. Conf. Multimedia and Expo Workshops*, Shenzhen, China, Jul. 2021.

[17] IETF. *IETF RFC 6716 Definition of the Opus Audio Codec.* (2012). [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6716.

[18] IETF. *RTP Payload for Versatile Video Coding (VVC)*. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-avtcore-rtp-vvc/.

[19] IETF. *IETF RFC 7798 RTP Payload Format for High Efficiency Video Coding (HEVC).* (2016). [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7798.

[20] IETF. *IETF RFC 6184 RTP Payload Format for H.264*. (2011). [Online]. Available: https://datatracker.ietf.org/doc/html//rfc6184.

[21] J. Daemen and V. Rijmen. *AES Proposal: Rijandael*. (1999). [Online]. Available: https://www.cs.miami.edu/home/burt/learning/Csc688.012/rijndael/rijndael_doc_V2.pdf.

[22] Defense Technical Information Center. *Secure Hash Standard*. (1995). [Online]. Available: https://apps.dtic.mil/sti/citations/ADA406543.

[23] IETF. *IETF RFC 2104 HMAC: Keyed-Hashing for Message Authentication*. (1997). [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2104.

[24] IETF. *IETF RFC 3830 MIKEY: Multimedia Internet KEYing*. (2004). [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3830.

[25] IETF. *IETF RFC 4568 Session Description Protocol (SDP) Security Descriptions for Media Streams*. (2006). [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4568.

[26] IETF. *IETF RFC 5764 Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP).* (2010). [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5764.

[27] *Crypto++ Library*. [Online]. Available: https://www.cryptopp.com/.