PERC Double Media Encryption for WebRTC 1.0 Sender Simulcast

Boris Grozev*†, Emil Ivov*, Arnaud Budkiewicz‡, Ludovic Roux§ and Alexandre Gouaillard§‡
*Atlassian PLC, Video Engineering Dept., Austin, Texas, USA
†University of Strasbourg, ICube Laboratory, Illkirch-Graffenstaden, France
‡Symphony Communication Services LLC, Palo Alto, California, USA
§CoSMo Software Consulting, Singapore

***Abstract*—Finance institutions have always needed enhanced security to protect their assets. Today they want to enjoy access to the cloud as well as new communication technologies like WebRTC. In parallel, the IETF created a new working group called Privacy Enhanced RTP Conferencing (PERC) that propose solutions to allow usage of media servers in the public cloud without compromising the security.**

This work is about implementing PERC's double encryption specifications in conjunction with WebRTC 1.0 sender simulcast. We are showing that the implementation of a few additional specifications, and an enhancement of the proposed Headers Extension mechanism allows for a viable double encryption for WebRTC 1.0 sender simulcast.

***Index Terms*—Media, Encryption, SRTP, WebRTC, Communications**

I. Introduction

Finance institutions need enhanced security to protect their assets. They traditionally achieved that through internal communication channels, or physical dedicated lines with trusted partners. Internally, high level encryption, together with smart management of security keys is adding the last touch to as-secure-as-you-can-be systems.

Nowadays, not only do they want to be able to communicate among each other, but they would love to leverage the power of cloud infrastructures, PaaSes and new standards of communication like WebRTC [1]. How to interoperate with browsers using WebRTC, still managing the security keys internally, and having part of the infrastructure in the open, without compromising the security?

The W3C and IETF standard committees, following the Snowden revelations, have put more emphasis on the security of all transactions over the Internet. The IETF created a new working group called Privacy Enhanced RTP Conferencing (PERC) [2] to address that concern, and during the meeting in Seoul in November 2016, made a call for implementation, particularly in conjunction with either WebRTC or SIP.

In this paper we will investigate the feasibility of an implementation of PERC's double encryption specifications [3] in conjunction with WebRTC. Modifications needed to the current draft will be presented along with their security implications.

978-1-5386-1322-1/17/$31.00

II. PERC Double Encryption Implementation

While WebRTC was designed to be encrypted end-to-end, it was also designed to be peer-to-peer. In the most recent version of the WebRTC 1.0 specifications [1], one can see specific APIs and features to support the sender simulcast use case [4]. In that use case, a peer (sender) will stream multiple copies of a media from the same source, but at different resolutions, to a media server. That media server will then selectively forward one of those incoming media streams to a remote peer (receiver) depending on the receiver's capacity (bandwidth, CPU, display size, etc.) or user interface configuration. In a multiparty conference, this allows a media server to forward different streams to different peers. This kind of media server is called a Selective Forward Unit or SFU [5].

Current implementations of simulcast for WebRTC (including the Jitsi media bridge, now part of Atlassian, and written by some of the authors) manage incoming simulcast, allowing receiving clients to be completely unaware that this technique is used on the sender side.

In these implementations the SFU switches the stream which is forwarded to a given receiver, while modifying certain RTP header fields in order to mask the operation for the receiver and output one consistent RTP stream. It might, for example, switch from sending a high-resolution stream to a lower resolution stream (and back) as the available bandwidth fluctuates. Specifically, SFUs modify the Sequence Number (from now on `SEQ`), the Synchronization Source (`SSRC`) and the Timestamp (`TS`) fields from the RTP header.

In this configuration, the media stream is only encrypted between the sender and the SFU, and between the SFU and the receiver, but not within the server itself. Anyone tampering with the media server can thus have access to unencrypted media content. At the system level, the PERC working group calls this an Hop-by-Hop (HBH) encryption, as opposed to an End-to-End (E2E) all the way between sender and receiver. We are interested in providing systemic end-to-end encryption for video conferencing systems based on WebRTC 1.0 simulcast.

The PERC working group produced a draft [3] which describes a mechanism for systemic end-to-end encryption in conferences which use a Media Distributor (MD). In this context the role of an MD can be served by an SFU and we will henceforth use the two terms interchangeably. The PERC

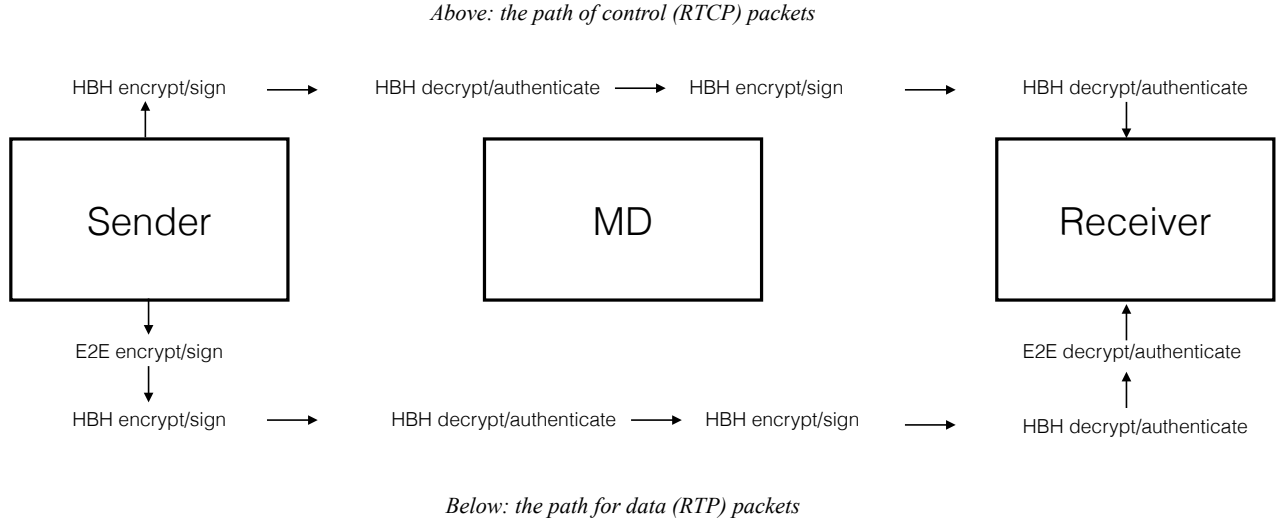

Fig. 1. End-to-End and Hop-By-Hop encryption of data (below) and control (above) packets in PERC.

system consists of endpoints using two separate cryptographic contexts, one end-to-end (E2E, or "inner") and one hop-by-hop (HBH, or "outer"). The MD has the HBH context, but not the E2E context. Media packets are encrypted twice, once with the E2E context, and once with the HBH context, and the MD re-applies the HBH encryption with the correct context for each receiver. Control packets (RTCP) are not E2E encrypted. See Figure 1.

The PERC draft uses two RTP header extensions [6] to allow an MD to perform selective forwarding. The Frame Marking extension [7] is added by media senders, and includes additional information about an RTP packet, which an MD needs in order to make a forwarding decision. The Original Header Block (OHB) extension [3] is added by the MD whenever it modifies any of the RTP header fields. This allows the receiver to reconstruct an exact copy of the original E2E-encrypted RTP packet, and thus decrypt and verify it.

In this paper we discuss an implementation of the scheme proposed in the "double" draft, with some modifications. We do not examine the problem of distributing the keys needed for E2E encryption, or that of identity verification.

The following two sections (II-A and II-B) describe the RTP Header Extensions and the modifications which we propose. The rest of the sections (II-C through II-G) discuss the main problems which we identified while implementing, and the solutions that we propose.

A. Extended OHB Format

The OHB header extension is defined in the "double" draft [3]. Its purpose is to allow an MD to modify a subset of the RTP header fields, while still making it possible for receivers to perform packet authentication end-to-end. This is accomplished by having the MD add the original values of the modified fields in an OHB extension, and having the receiver remove the extension and restore the original values before performing authentication. The current version of the draft defines a structure which can contain at most the original RTP Payload Type (from now on `PT`) and `SEQ` fields, and this restricts the MD to only modifying these two fields, while, as explained before, for the simulcast use case the MD may also need to modify the `SSRC` and `TS`.

We propose to keep the general scheme, and preserve the semantics of the OHB extension, but extend its structure so that it can contain the additional fields needed for the WebRTC 1.0 simulcast use case. In order to optimize the overhead we still use the 4-bit length field of the header extension header [6] to encode the structure of the remaining bytes. Specifically, we use it to encode which of these four fields will be present in the following bytes: Payload Type (`PT`, 1 byte, including an extra R-bit), Sequence Number (`SEQ`, 2 bytes), `SSRC` (4 bytes), Timestamp (`TS`, 4 bytes).

If the length field has a value of 0, 1 or 2 (indicating that there are respectively 1, 2 or 3 more bytes), then the format is exactly the same as in the current draft. This means that an endpoint which implements the extended format can work with the current format without modification.

Table I lists the fields which are encoded (and the order in which they are included) for the different values of the length field and the R-bit. The R-bit is the most significant bit (MSB) of the `PT` field, and is present if and only if the `PT` field is present. If the extension includes the `PT` field, it consists of an R-bit (MSB) and 7 bits for the original RTP Payload Type. The R-bit is used to encode which fields are included, when there is ambiguity.

B. Frame Marking

To achieve selective forwarding, the MD needs to know whether the packet is part of a keyframe or not. This infor-

TABLE I
THE FIELDS ENCODED IN THE ORIGINAL HEADER BLOCK HEADER EXTENSION, FOR DIFFERENT VALUES OF THE LENGTH FIELD AND THE R-BIT.

len	R-bit set	R-bit not set
0	PT	
1	SEQ	
2	PT, SEQ	
3	SSRC	
4	PT, SSRC	PT, TS
5	SEQ, SSRC	
6	PT, SEQ, SSRC	PT, SEQ, TS
7	SSRC, TS	
8	PT, SSRC, TS	
9	SEQ, SSRC, TS	
10	PT, SEQ, SSRC, TS	

mation is not included in the RTP header of a packet, and is usually part of the payload. Existing implementations extract this from the payload (in a codec-specific way). With PERC this is not possible anymore, because the payload is E2E encrypted and therefor unavailable to the MD. The framemarking draft [7], from the IETF's *avtext* working group, has a solution to this problem. It defines an RTP Header Extension which contains additional information about a packet (such as whether the packet is part of an independent frame). The RTP Header Extensions are not E2E encrypted, and thus always remain available to the MD.

The details of the Header Extensions are as follows, and were implemented in our solution.

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  ID=? |  L=0  |S|E|I|D|0 0 0 0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Note that we use the first of the remaining 4 bits to signal padding packets, see section II-E.

C. Injecting Video Frames

Obviously one of the goals of the PERC architecture is to not allow an MD to modify or inject media content in a conference that it hosts. However, there are certain situations in which injecting media may be useful.

For example, due to a long standing bug in the Chrome / Chromium browser[1], the audio from a `MediaStream` which contains both audio and video tracks is never played back until some video packets are received. In order to work around this in the case where a participant sends no video, our SFU injects empty video frames in the beginning of a stream. We had to disable this behaviour and fall back to using two separate `MediaStream` objects for the audio track and the video track on the receiver side. This removes the audio playback problem, but disables synchronization between the playback of audio and video on the receiver side.

[1]https://bugs.chromium.org/p/webrtc/issues/detail?id=5254

D. RTX

One of the mechanisms WebRTC uses to handle packet loss is packet retransmissions [8], and its effectiveness depends highly on the delay [9]. Thus handling retransmissions in the SFU is desirable, because it can reduce the delay of retransmitted packets by a half on average (since the SFU is in between any two peers on the transport path). Usually, the SFU would implement two mechanisms. It would request missing packets from the senders if it itself failed to receive them. It would also buffer/cache outgoing packets for each receiver, and respond to retransmission requests from receivers.

The RTX format [10] is used for RTP packet retransmission. It uses its own RTP payload type with its own SSRC and thus has its own sequence number space. When a packet is retransmitted, it inherits the original packet headers with the exception of these three fields (`PT`, `SSRC` and `SEQ`). The payload type and SSRC are paired via signaling with those of the corresponding media stream, so the original values can be reconstructed on the receiving end. The sequence number, however, is encoded in the Original Sequence Number (OSN) field, which is the first two bytes of the RTP payload. And as part of the payload, it is encrypted by SRTP.

In the case of the double encryption scheme, if RTX packets are handles like regular RTP packets, this means that the MD can not read the OSN in incoming RTX packets, and can not create its own RTX packets (as this involves modifying the payload by inserting the OSN).

We propose to solve this problem by using RTX hop-by-hop only. That is, the RTX transformation is applied after the E2E SRTP transform is applied, but before the HBH. This is the same way that PERC "double" handles RTCP packets, except of course that the input is an RTX packets which encapsulates and already E2E encrypted media packet.

Specifically, the procedure for a sender to retransmit a packet is the following: first it finds the unencrypted packet to be retransmitted and applies the E2E SRTP transformation to it. Then, it applies the RTX transformation, namely it replaces the Payload Type and SSRC with those of the RTX stream, inserts the OSN as the first two bytes of the payload, and generates a new sequence number. Then it applies the HBH SRTP transformation.

When an MD receives an RTX packet, it first applies the reverse HBH SRTP transformation. It then restores the original E2E-encrypted packet by replacing the Payload Type and SSRC fields with those of the associated media stream, and restores the sequence number from the OSN (first two bytes of the payload). It can then treat the restored packet as usual, optionally encapsulate it in RTX, and send it to other endpoints encrypted with the corresponding HBH context. The same procedure applies to receiving endpoints, except, of course, that after restoring the E2E-encrypted packet they need to verify and decrypt it.

The above scheme allows an MD to insert packets into an RTX stream. However, it does not impact an MD's inability to insert packets in a media stream, because packets transported in RTX are still protected with the E2E context.

E. Terminating Bandwidth Estimation

Part of the mechanism used in *webrtc.org* for bandwidth estimation is to actively probe for additional available bandwidth [11] by sending extra padding packets. An MD should terminate the bandwidth estimation, because the available bandwidth normally varies per endpoint. Thus, packets sent for bandwidth estimation only, which don't contain any actual media should not be forwarded to other endpoints.

The RTP padding mechanism uses the last byte of the payload to encode the padding length [12], and so the field is encrypted by SRTP as part of the payload [13]. This means that an MD is not able to read the padding length field for E2E encrypted packets, and it can not differentiate between padding-only packets and those that contain (some) media. Thus, in the context of PERC, if additional data needs to be sent for the purposes of bandwidth estimation, it should not be sent as RTP padding.

There are different ways in which this problem could be solved. We propose to use the RTX stream, together with a padding-only indication in a header extension in order to allow probing-only packets to be sent HBH (and HBH only). Just using the RTX format without other changes doesn't work for two reasons. First, the last byte of the RTX payload is still E2E encrypted, and second, the padding bit in the RTP header must match between the inner and outer SRTP transformations, because the "double" draft does not include a way for the original value to be recovered. Using a separate padding-only bit in the header also has the advantage that is allows padding-only packets with more than 256 bytes of data to be sent.

We propose to use the frame marking header extension to carry the padding-only bit. Clearly we don't intend for this bit to be used for packets containing actual media, so we define it only in the one-byte format used for non-scalable streams, and not in the 3-byte format for scalable streams (see section II-B and also Sections 3.1 and 3.2 from the frame marking draft [7]). We use the first of the remaining bits (originally defined as 0) as the "padding-only" bit. We do not reuse the "discardable frame" bit, because it has different semantics (that the packet contains media, but it may be discarded if necessary).

F. RTP Timestamps

The current implementation of simulcast in *webrtc.org* (used in turn in Chrome, Firefox and Safari) uses different initial offsets for the RTP time stamp for each simulcast stream. This forces an SFU to re-write the timestamps of the single RTP stream it outputs.

In the context of PERC, however, allowing an MD to modify the RTP time stamp is not desired, because it has security implications. Namely, the MD could perform replay attacks. These attacks are prevented by the verification of the RTP sequence number.

In our work we chose to allow rewriting of the time stamp field by the MD, as explained in sections II and II-A above. The reason is the ease of implementation using existing software, and the limited impact on security during local tests. This should not be allowed in real systems.

A possible viable solution for production would be for simulcast senders to use the same offset for all simulcast streams coming from a unique source. This would remove the need for timestamps modification by the MD. It would also allow the receiver to verify the timestamps of all streams end-to-end, which prevents a form of attack in which the MD does not initially forward one of the input streams, and then replays it at a later time.

Since RTP timestamps use a 32-bit integer field, which wraps around, some form of replay attacks may still be possible. To prevent this we suggest that the conference is re-keyed often enough. In the case of the usual $90kHz$ RTP clock rate, it takes around 13 hours for the RTP timestamps to cycle, so re-keying should happen at least that often.

G. Signaling Double Encryption

An SFU needs two pieces of information in order to function in the double encryption scheme described above. These are the ID number for the two header extensions, frame marking and OHB.

If an SFU needs to differentiate between the cases where double encryption is used, it can do so solely based on the presense of the OHB header extension. That is, if OHB has been configured, it will always add the modified fields in an OHB extensions, and will follow the semantics defined in the double draft for handling other RTP header extensions. Otherwise, it should not add an OHB extension, and is free to add, remove or modify other header extensions.

III. Analysis of the Overhead

The "double" SRTP encryption scheme used in PERC adds an additional overhead to each packet in two ways. One comes from the additional SRTP [13] transformation, and the fields added by the EKT format [14]. This adds at least 4 bytes to each packet. It might be very large when a new master key is included, but this is not expected to happen often. Thus we assume an overhead of 4 bytes for most packets.

The other comes from the Original Header Block RTP Header Extension added by the MD. This adds between 0 (if no headers were modified) and 8 bytes (if both the Payload Type and Sequence Number were modified, and the original packet contained no header extensions).

Our extended OHB format adds between 0 and 16 bytes (if all of `PT`, `SEQ`, `TS` and `SSRC` were modified and the original packet had no header extensions). However, it does not change the overhead compared to the current OHB unless the extra features (modifying the `SSRC` and/or `TS` fields) are used.

Together, the overhead is between 4 and 20 bytes, and in our implementation it is 4 for audio packets and usually 16 for

video packets. This is because header extensions are already in use, the MD does not need to modify any headers for the audio streams, and usually modifies the SEQ, SSRC and TS fields for video.

In a typical WebRTC call scenario, using the Opus codec with $20ms$ frames and the default bitrate of $40kbps$ we observe average packet sizes of around 100 bytes. Thus the overhead between sending endpoints and the MD is 4%, and between the MD and receiving endpoints it can be as high as 20%, but in most cases it is 4%.

For video, the average packet size is usually between 600 and 1200 bytes, depending on the bitrate. Thus the overhead between sending endpoints and the MD is less than 0.7%, while the overhead between the MD and receiving endpoints is up to 4% (and 2.7% on average on our system).

For a combined audio and video call, the overhead is closer to the one for video, since video comprises most of the data.

IV. Related Work

The PERC working group at the IETF continues to be active and there have been other proposals addressing some of the same problems that we discuss (some of them submitted after our work was originally written).

One idea is to replace the complex OHB mechanism, and instead encapsulate the whole end-to-end encrypted RTP packet, headers and authentication block included, when applying the hop-by-hop encryption. This results in some additional overhead, but it is probably not significant.

The question of handling RTX and other stream repair mechanisms such as Forward Error Correction (FEC) [15] has seen a lot of discussion. There is agreement that the need exists for such mechanisms to be used hop-by-hop, however there doesn't seem to be a consensus on what the best way to achieve it is.

One proposal was to simply send packets for stream repair without additional protection. This is not viable, because lacking the E2E context an MD can not authenticate incoming packets, and also in the case of FEC, because receivers would have to try all different combinations of packets when authentication of a recovered packet fails.

Another proposal, referred to as "triple", consists of applying the hop-by-hop encryption twice to repair packets. This brings extra complexity in handling the encryption of packets in a special way, different from the way the encryption of RTP and RTCP packets is already handled by PERC. The second encryption round, since it uses the same context, does not have any useful cryptographic properties. It also requires modifications to the way SRTP is performed, because with normal operation the same context can not be used twice for a packet. This is because SRTP's replay protection is based on the RTP sequence number of the packet, which doesn't change. This proposal is now included in the latest version of the double draft [3].

V. Conclusion and Future Work

This paper illustrated for the first time a real-world implementation of the proposed PERC standard. Out of this work, it is clear that a lot of what is implemented today in the browser is still quite proprietary, and supporting Chrome in our case required extra work. However, solutions to almost all the problems we faced already existed in some specifications, e.g. frame marking [7] or "diet"-EKT [14] to name a few and this paper provides the details needed to achieve an implementation. We also showed that the current OHB format is insufficient, but can be easily extended to achieve what is needed. Finally, a study of the bandwidth cost also shows that the overhead is almost negligible. In our case, where E2E encryption was needed for compliance reasons, the additional bandwidth is completely acceptable.

References

[1] A. Bergkvist, D. Burnett, C. Jennings, A. Narayanan and B. Aboba, "WebRTC 1.0: Real-time Communication Between Browsers," November 2016, W3C Editor's Draft. [Online]. Available: https://w3c.github.io/webrtc-pc/archives/20161123/webrtc.html

[2] R. Barnes and S. Nandakumar, "Privacy Enhanced RTP Conferencing," 2016, IETF Working group. [Online]. Available: https://datatracker.ietf.org/wg/perc/charter/

[3] C. Jennings, P. Jones, R. Barnes, and A. Roach, "SRTP Double Encryption Procedures," 2017, IETF Internet Draft. [Online]. Available: https://tools.ietf.org/html/draft-ietf-perc-double-06

[4] B. Grozev, G. Politis, E. Ivov, T. Noel, and V. Singh, "Experimental evaluation of simulcast for webrtc," *IEEE Communications Standards Magazine*, vol. 1, no. 2, pp. 52–59, 2017.

[5] M. Westerlund and S. Wenger, "RTP Topologies," RFC 7667 (Informational), Internet Engineering Task Force, Nov. 2015. [Online]. Available: http://www.ietf.org/rfc/rfc7667.txt

[6] D. Singer and H. Desineni, "A General Mechanism for RTP Header Extensions," RFC 5285 (Proposed Standard), Internet Engineering Task Force, Jul. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5285.txt

[7] E. Berger, S. Nandakumar, and M. Zanaty, "Frame Marking RTP Header Extension," 2017, IETF Internet Draft. [Online]. Available: https://tools.ietf.org/html/draft-ietf-avtext-framemarking-05

[8] S. Holmer, M. Shemer, and M. Paniconi, "Handling packet loss in webrtc," in *International Conference on Image Processing (ICIP 2013)*, 2013, pp. 1860–1864.

[9] M. Nagy, V. Singh, J. Ott, and L. Eggert, "Congestion control using fec for conversational multimedia communication," in *Proceedings of the 5th ACM Multimedia Systems Conference*, ser. MMSys '14. ACM, 2014, pp. 191–202.

[10] J. Rey, D. Leon, A. Miyazaki, V. Varsa, and R. Hakenberg, "RTP Retransmission Payload Format," RFC 4588 (Proposed Standard), Internet Engineering Task Force, Jul. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4588.txt

[11] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Analysis and design of the google congestion control for web real-time communication (webrtc)," in *Proceedings of the 7th International Conference on Multimedia Systems*, ser. MMSys '16. New York, NY, USA: ACM, 2016, pp. 13:1–13:12. [Online]. Available: http://doi.acm.org/10.1145/2910017.2910605

[12] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 3550 (INTERNET STANDARD), Internet Engineering Task Force, Jul. 2003, updated by RFCs 5506, 5761, 6051, 6222, 7022, 7160, 7164. [Online]. Available: http://www.ietf.org/rfc/rfc3550.txt

[13] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)," RFC 3711 (Proposed Standard), Internet Engineering Task Force, Mar. 2004, updated by RFCs 5506, 6904. [Online]. Available: http://www.ietf.org/rfc/rfc3711.txt

[14] C. Jennings, J. Mattsson, D. McGrew, D. Wing, and F. Andreasen, "Encrypted Key Transport for Secure RTP," 2016, IETF Internet Draft. [Online]. Available: https://tools.ietf.org/html/draft-ietf-perc-srtp-ekt-diet-02

[15] V. Singh, A. Begen, M. Zanaty, and G. Mandyam, "RTP Payload Format for Flexible Forward Error Correction (FEC)," 2017, IETF Internet Draft. [Online]. Available: https://tools.ietf.org/html/draft-ietf-payload-flexible-fec-scheme-05