

Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 25, 2012

O. Ohlsson
Ericsson
January 22, 2012

Support of SDES in WebRTC
draft-ohlsson-rtcweb-sdes-support-00

Abstract

Which key management protocols to support has been lively debated in WebRTC on several occasions. This document explains the benefits of SDES and argues why allowing it as an alternative option has little impact on security.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 25, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
2.	Benefits of Supporting SDES	3
2.1.	Reduced Complexity of WebRTC-SIP Gateway	3
2.2.	Reduced Processing (Less SRTP Terminations)	3
2.3.	Reduced Call Setup Time	4
3.	Security Considerations	4
3.1.	SDES vs DTLS-SRTP in Case of Inside Attacker	5
3.1.1.	Extraction of Log Data	5
3.1.2.	Script Injection	5
3.2.	SDES vs DTLS-SRTP in Case of Inside Attacker	6
3.2.1.	Downgrade Attack	6
3.2.2.	Difficulties with Key Continuity	7
3.2.3.	3rd Party Identity Assertion	8
4.	Discussion and Conclusion	9
5.	Informative References	9
	Author's Address	10

1. Introduction

Which key management protocols to support has been lively debated in WebRTC on several occasions. The main question is the following: Should applications be restricted to DTLS-SRTP or could SDES be allowed as an alternative option?

In this document we identify and address the issues that have been raised. We explain the benefits of SDES and argue why allowing it as an alternative option has little impact on security.

2. Benefits of Supporting SDES

Being able to communicate from WebRTC applications to existing SIP/RTP endpoints is a highly desirable use case. The SIP installed base is huge and contains millions of devices and a large number of applications (e.g. conferencing and voicemail). Even more important, nearly all mobile phones and landlines are reachable through SIP/RTP gateways deployed in service provider networks. The same can also be said for other signaling protocols, such as XMPP or H.323. As a sidenote, the recent work on the DTMF tone API in WebRTC proves that many members consider legacy interworking to be important.

2.1. Reduced Complexity of WebRTC-SIP Gateway

Communication between the Browser and SIP/RTP endpoint will most likely require some form of media-plane gateway (due to the need to terminate ICE). The development and testing costs for such gateways are typically very high since they need to handle a large number of users and often contain special purpose hardware. It is definitely worthwhile to try to reduce costs by lowering the complexity and removing functionality that is not strictly required. This would result in lower prices which will lead to a higher degree of interconnectivity between WebRTC and existing SIP deployments.

Already today there are SBCs that perform SRTP termination on behalf of endpoints with SDES based keying (there are SBCs that support DTLS-SRTP but this is uncommon). If the browser also supported SDES, the WebRTC gateway could simply forward all SRTP packets to the SBC and let it decide whether to terminate encryption or not (depending on the capabilities of the receiving endpoint).

2.2. Reduced Processing (Less SRTP Terminations)

A large part of existing SIP/RTP devices support SRTP and most of them that do, use SDES based keying. This is confirmed in the report from the latest SIPit event which stated that:

- o 80 percent of the tested implementations supported SRTP
- o 100 percent of the SRTP implementations supported SDES
- o 0 percent of the SRTP implementations supported DTLS-SRTP

Although these figures may not be entirely accurate, they at least provide an indication of the current situation. Assuming that SDES is supported by browsers, a major part of all calls (80 percent if the above figures were correct) would not need to be encrypted/decrypted by an intermediate gateway. This is a substantial reduction in processing cost for the gateway. Another benefit is that for those endpoints that support SDES the call will be protected end-to-end for free. Achieving this with DTLS-SRTP would require the gateway to first decrypt and then re-encrypt traffic.

Note that the important question is whether the gateway needs to terminate SRTP at all. Processing wise there is probably not that much difference in terminating an SRTP + SDES or an SRTP + DTLS-SRTP call.

2.3. Reduced Call Setup Time

One common argument against SDES is its inability to handle early media (i.e. media that arrives at the SDP offerer before the SDP answer arrives). However, since at least one full signaling roundtrip is required to conclude ICE, this argument is not applicable in WebRTC. In fact, media starts to flow later with DTLS-SRTP than with SDES since additional time is required for the DTLS handshake to complete.

Note that what is said above only applies to the first offer/answer exchange. If an additional media stream is added later in time and multiplexing is used (i.e. single 5-tuple carrying all flows established with a single run of ICE), then the problem with early media could arise when SDES is used (but never with DTLS-SRTP).

3. Security Considerations

At this point most readers should agree that SDES is favourable from an interworking point of view. It is also clear that implementing SDES in WebRTC is a relatively straight forward task. What remains to be considered are its impacts on security.

We distinguish between the following two types of attackers:

Outside Attacker	An external party attempts to intercept a call (e.g. a host located on the same WLAN as the user)
Inside Attacker	The web application itself (or the signaling server, in case the web server and signaling server are separated) attempts to intercept a call

3.1. SDES vs DTLS-SRTP in Case of Inside Attacker

By requiring that signaling is secured using TLS, an outside attacker that monitors network traffic will not be able to extract the SDES keys. Therefore, in this scenario both SDES and DTLS-SRTP provide a sufficient level of protection.

The two other types of attacks that have been mentioned in this context are extraction of log data and code injection, each of which are considered below.

3.1.1. Extraction of Log Data

In this scenario the attacker manages to decrypt a previously recorded call by attacking the signaling server and extracting the SDES keys from the server log.

First of all, if the attacker gets as far as reading the logging data then eavesdropping of past calls is probably not the only problem. The effort required to break into the server is also related to the amount of trust the user assigns to the web application: well trusted sites often have well protected servers.

Secondly, it can be questioned how common this type of extensive logging really is. For example, user login via HTML forms is extremely common yet one seldom hear of passwords being extracted from server logs.

Finally, SDES will primarily be used when interworking with existing SIP systems deployed within enterprises or service providers. These have been using SDES for a long time and know that it is critical to protect the plain text keys.

3.1.2. Script Injection

In this scenario the attacker manages to inject his own piece of JavaScript into the WebRTC application. The next time a user downloads the application and places a call, the script will execute and start eavesdropping on the conversation.

There are three major ways in which code can be injected into a web application:

- o The page itself or one of its included JavaScript files is downloaded over a non-HTTPS link and is modified en route
- o The web application intentionally includes JavaScript supplied by the attacker (e.g. a third-party library or advertisement)
- o HTML form input or URL parameters are not properly sanitized (i.e. classical XSS vulnerability)

Modification en route can be ignored by requiring HTTPS to be used for all content. Whether the two other injection techniques are feasible or not largely depends on the application.

If script injection occurs then there are other methods to intercept a call, like establishing additional PeerConnection objects or use a recording interface and send the data using WebSocket. As long as these methods are available it does not matter much whether the application uses SDES or DTLS-SRTP.

In general, if an attacker manages to execute even a small piece of JavaScript then he has effectively gained full control of the application (additional code can be included and HTML elements removed/inserted). Since this situation is exactly the same as the situation with an inside attacker, script injection will not be discussed further.

3.2. SDES vs DTLS-SRTP in Case of Inside Attacker

First of all, it can be questioned if we really want to protect ourselves against an inside attacker. If consent is required every time the application wants to record or forward media then the user experience will suffer. One could also imagine future applications that want to use their own codecs or filters (for example a voice scrambler or face detection software), something which is difficult to achieve without access to the underlying bitstreams.

We ignore this problem for now and simply assume that the application cannot access the media from within the browser. In other words, we only consider protection of the media during transport.

3.2.1. Downgrade Attack

The major argument against SDES is that it would make it trivial for the application to perform interception. Let us compare what would be required in both cases.

Interception of SDES call:

1. Copy and store the 'a=crypto:' lines in the offer/answer SDP
2. Force media to pass through TURN server by deleting all candidates except the relayed one
3. Store all SRTP packets that pass through the TURN server and decrypt them later on (using the keys from step 1)

Interception of DTLS-SRTP call:

1. Replace the 'a=fingerprint:' lines in the offer/answer SDP with the fingerprint of a public key generated by the application
2. Force the media to go through the TURN server by deleting all ICE candidates except the relayed one
3. Modify an existing TURN server implementation so that it decrypts and re-encrypts the DTLS traffic (using the public-private key pair from step 1)

Putting the modified TURN server into place is the hardest part of intercepting a DTLS-SRTP call. Once this is done however, the remaining steps are fairly straightforward. This shows that neither DTLS-SRTP nor SDES provides any significant protection against an inside attacker.

There is one benefit of DTLS-SRTP that is not directly apparent from the above description. If both users read their respective fingerprint values over the voice channel then they can detect if the conversation is being intercepted. However, it is very unlikely that the average user would bother doing this.

3.2.2. Difficulties with Key Continuity

The comparison in the previous section is somewhat simplified since it does not consider DTLS-SRTP key continuity. The way this mechanism works is that the browser will notify the user whenever it receives a certificate which has not previously been seen (i.e. not present in the browser cache). Since the user will receive this notification every time he calls someone new and whenever someone changes browser, it is very likely that he/she will simply ignore it.

Reuse of public keys also has privacy implications as it enables user tracking. A user that wants to remain anonymous towards a service provider would need to generate a fresh key for each interaction. Furthermore, in order to avoid colluding service providers (e.g.

medical clinics and insurance agencies) from linking a user's activities, separate certificates are needed for different domains. However, storing domain names together with the certificates would allow the next browser user (e.g. a family member) to see which sites the previous user visited. All of this leads to more certificates being generated which in turn results in even more "new key" notifications.

It is also important to understand that the cached certificates are not bound to any identity (the certificates are simple containers for the public key without any additional information). This means that if just one of the cached keys is compromised any user call can be intercepted without causing the "new key" notification to be displayed. Note that the risk of this happening is directly related to the size of the cache, which grows over time.

3.2.3. 3rd Party Identity Assertion

[I-D.ietf-rtcweb-security] suggests a way to strengthen the security of DTLS-SRTP by validating the received fingerprint via an identity provider. At the time of writing the proposal is still a bit vague (for example, it is not clear how the identity provider is selected in practice) but it definitely seems promising. Such a mechanism (including the necessary browser chrome) would make it significantly harder for the application to act as man-in-the-middle.

The question is whether the identity mechanism is optional or not, i.e. will it be possible for an application to use "plain" DTLS-SRTP. The answer is most likely "yes" due to the following reasons:

- o Many applications are already trusted by the user
- o Some applications do not want to depend on third parties
- o Some users do not have any identity provider account
- o Users may not always want to reveal their identity
- o Working out all the details of the identity mechanism will take time (and if it is not mandatory from start there are backward compatibility issues)

Note that allowing an application to be its own identity provider is effectively the same as allowing plain DTLS-SRTP (the user trusts the application) only more complicated.

4. Discussion and Conclusion

We are not looking to replace DTLS-SRTP with SDES. The 20-line WebRTC developer will continue to use the default option which is DTLS-SRTP, while others who are interested in interworking will select SDES. The latter group will be required to use HTTPS for all content and can be informed of the necessary precautions (secure storage of log files or otherwise no extensive logging).

The main issue that appears to concern members is the application's ability to downgrade security. But as we have seen it is not significantly harder for the application to attack DTLS-SRTP. The main advantage of DTLS-SRTP is the possibility to detect when a call is being intercepted. However, doing so requires an effort from the user and a certain degree of technical skill. It can also be questioned to what extent the application should be restricted from accessing media since this limits usability and innovativity.

Finally, it has been suggested that additional identity mechanisms could prevent the application from listening in on calls. While this is certainly true, any such mechanism would most likely be made optional. If that is the case or if an application can be its own identity provider, then we are back at the situation where the user has to decide which sites to trust.

5. Informative References

[I-D.ietf-rtcweb-overview]

Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-02 (work in progress), September 2011.

[I-D.ietf-rtcweb-security]

Rescorla, E., "Security Considerations for RTC-Web", draft-ietf-rtcweb-security-01 (work in progress), October 2011.

[I-D.ietf-rtcweb-use-cases-and-requirements]

Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use-cases and Requirements", draft-ietf-rtcweb-use-cases-and-requirements-06 (work in progress), October 2011.

[I-D.kaplan-rtcweb-sip-interworking-requirements]

Kaplan, H., "Requirements for Interworking WebRTC with Current SIP Deployments", draft-kaplan-rtcweb-sip-interworking-requirements-02 (work

in progress), November 2011.

- [RFC4568] Andreassen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, July 2006.
- [RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, May 2010.
- [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, May 2010.
- [SIPit] "SIPit27 Summary",
<https://www.sipit.net/SIPit27_Summary>.

Author's Address

Oscar Ohlsson
Ericsson
Farogatan 6
SE-164 80 Kista
Sweden

Email: oscar.ohlsson@ericsson.com