Web媒体处理与实时传输标准实践

Practices of Web Media Processing and Real-time Communication Standards
1. Background
2. Case1: E2E Encryption
3. Case2: Digital Rights Management
4. Case3: H265 Supporting for RTC
5. Case4: Alpha Video Transmission
**Background**

**New Trends in the RTC Industry**

- Expanding overseas business
- Web is the most important platform overseas
- Security and compliance by design is required by foreign laws
- Diverse application scenarios create diverse requirements on web infrastructure
Case 1: E2E Encryption

Providing reliable transmission network

- Encrypt and decrypt media by end device to protect privacy
- User data could not be decrypted by RTC service provider
Case 1: E2E Encryption

Requirements:
- Provide interface to grab encoded media data
- Provide high performance encryption/decryption component
- Provide interface to write back transformed media data
**Case 1: E2E Encryption**

**Secure Frame (SFrame)**

- End-to-end encryption and authentication mechanism for media frames
- Compatible with RTP & non-RTP media transport
- Reduce bandwidth overhead by adding encryption overhead only once per media frame, instead of once per packet.

---

**Secure Frame (SFrame)**

**Abstract**

This document describes the Secure Frame (SFrame) end-to-end encryption and authentication mechanism for media frames in a multiparty conference call, in which central media servers (selective forwarding units or SFUs) can access the media metadata needed to make forwarding decisions without having access to the actual media.

The proposed mechanism differs from the Secure Real-Time Protocol (SRTP) in that it is independent of RTP (thus compatible with non-RTP media transport) and can be applied to whole media frames in order to be more bandwidth efficient.
**Case 1: E2E Encryption**

**SFrameTransform**

Spec: [https://www.w3.org/TR/webrtc-encoded-transform/](https://www.w3.org/TR/webrtc-encoded-transform/)

```javascript
typedef (SFrameTransform or RTCrtpScriptTransform) RTCrtpTransform;

// New methods for RTCrtpSender and RTCrtpReceiver
partial interface RTCrtpSender {
    attribute RTCrtpTransform? transform;
}

partial interface RTCrtpReceiver {
    attribute RTCrtpTransform? transform;
}
```

```javascript
enum SFrameTransformRole {
    "encrypt",
    "decrypt"
};
dictionary SFrameTransformOptions {
    SFrameTransformRole role = "encrypt";
};
typedef [EnforceRange] unsigned long long SmallCryptoKeyID;
typedef (SmallCryptoKeyID or bigint) CryptoKeyID;

[Exposed=(Window,DedicatedWorker)]
interface SFrameTransform : EventTarget {
    constructor(optional SFrameTransformOptions options = {});
    Promise<undefined> setEncryptionKey(CryptoKey key, optional CryptoKeyID keyID);
    attribute EventHandler onerror;
};
```

---

**Diagram**

[Diagram showing the flow of encryption and decryption through SFrameTransform and RTCrtpSender/Receiver]
Case 1: E2E Encryption

web-platform-tests dashboard
Latest Run  Recent Runs  Interop 2024  Insights  Processor  About

wpt / webrtc-encoded-transform

partly

For information on the search syntax, view the search documentation

Showing 5 tests (11 subtests) in webrtc-encoded-transform from the latest master test runs for chrome[experimental], edge[experimental], firefox[experimental], safari[experimental]

Path


sframe-keys.https.html
sframe-transform-buffer-source.html
sframe-transform-in-worker.https.html
sframe-transform-readable.html
sframe-transform.html

Subtest Total

0 / 11  0 / 11  0 / 11  0 / 11
Case 1: E2E Encryption

Partly testable on Safari with feature flag enabled

Functionalities not ready

WebKit Bugzilla
Bug 218752: Add a WebRTC SFrame transform

Summary: Add a WebRTC SFrame transform

Status: RESOLVED FIXED
Product: WebKit
Component: WebRTC (other bugs)
Version: WebRTC Local Build
Hardware: Unspecified Unspecified
Importance: P2 Normal
Assignee: youenn fablet
URL: Key Radar
Depends on: 218751
Blocks: Show dependency tree / wash

Summary
Harness status: OK

Test name:

<table>
<thead>
<tr>
<th>Result</th>
<th>Test Name</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass</td>
<td>Cannot reuse attached transforms</td>
<td>• Asserts run</td>
</tr>
<tr>
<td>Pass</td>
<td>SFrameTransform express readable and writable</td>
<td>• Asserts run</td>
</tr>
<tr>
<td>Fail</td>
<td>readable/writable are locked when attached and after being attached</td>
<td>• Asserts run</td>
</tr>
<tr>
<td>Fail</td>
<td>SFrame with array buffer - authentication size 10</td>
<td>• Asserts run</td>
</tr>
<tr>
<td>Fail</td>
<td>SFrame decryption with array buffer that is too small</td>
<td>• Asserts run</td>
</tr>
<tr>
<td>Fail</td>
<td>SFrame transform gets around if trying to process unexpected value types</td>
<td>• Asserts run</td>
</tr>
</tbody>
</table>

Partly testable on Safari with feature flag enabled

Functionalities not ready
Case 1: E2E Encryption

RTCRtpScriptTransform

Spec: [https://www.w3.org/TR/webRTC-encoded-transform/](https://www.w3.org/TR/webRTC-encoded-transform/)

typedef (SFrameTransform or RTCRtpScriptTransform) RTCRtpTransform;

// New methods for RTCRtpSender and RTCRtpReceiver
partial interface RTCRtpSender {
    attribute RTCRtpTransform? transform;
};

partial interface RTCRtpReceiver {
    attribute RTCRtpTransform? transform;
};

[Exposed=DedicatedWorker]
interface RTCPTransformEvent : Event {
    readonly attribute RTCRtpScriptTransformer transformer;
};

[Exposed=DedicatedWorker]
interface RTCRtpScriptTransformer : EventTarget {
    // Attributes and methods related to the transformer source
    readonly attribute ReadableStream readable;
    Promise<UnsignedLong> generateKeyFrame(optional DOMString rid);
    Promise<undefined> sendKeyFrameRequest();
    // Attributes and methods related to the transformer sink
    readonly attribute WritableStream writable;
    attribute EventHandler onkeyframerequest;
    // Attributes for configuring the Javascript code
    readonly attribute any options;
};

[Exposed=Window]
interface RTCRtpScriptTransform {
    constructor(Worker worker, optional any options, optional sequence<object> transfer);
}
Case 1: E2E Encryption

E2E encryption with RTCRtpScriptTransform and Web Crypto API
Case 2: Digital Rights Management

Traditional DRM for CDN one-way media
**Case 2: Digital Rights Management**

**WebRTC Extended Use Cases**

### Case: Live encoded non-WebRTC media

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N40</td>
<td>An application can create an outgoing WebRTC connection without activating an encoder.</td>
</tr>
<tr>
<td>N41</td>
<td>An application can create encoded video frames from encoded data and metadata, and enqueue them on an outgoing WebRTC connection.</td>
</tr>
<tr>
<td>N42</td>
<td>The WebRTC connection can generate signals indicating the desired bandwidth, and surface those to the application.</td>
</tr>
</tbody>
</table>

### Case: Transmitting stored encoded media

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N41</td>
<td>An application can create encoded video frames from encoded data and metadata, and enqueue them on an outgoing WebRTC connection.</td>
</tr>
<tr>
<td>N42</td>
<td>The WebRTC connection can generate signals indicating the desired bandwidth, and surface those to the application.</td>
</tr>
<tr>
<td>N43</td>
<td>The application can modify metadata on outgoing frames so that they fit smoothly within the expected sequence of timestamps and sequence numbers.</td>
</tr>
<tr>
<td>N44</td>
<td>The application can signal the WebRTC encoder when resuming live transmission in such a way that generated frames fit smoothly within the expected sequence of timestamps and sequence numbers.</td>
</tr>
</tbody>
</table>

### Case: Decoding pre-encoded media

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N45</td>
<td>An application can create an incoming WebRTC connection to accept frames as if they were coming in over RTP, without creating an RTP transport.</td>
</tr>
<tr>
<td>N46</td>
<td>An application can create encoded video frames from encoded data and metadata, and enqueue them on an incoming WebRTC connection.</td>
</tr>
<tr>
<td>N47</td>
<td>The WebRTC connection can generate signals indicating demands for keyframes, and surface those to the application.</td>
</tr>
</tbody>
</table>
Case 2: Digital Rights Management

DRM requirements for WebRTC one-way media:

- Transmitting stored pre-encoded media as part of the WebRTC RTP session
- Decrypting media with CDM, and decoding pre-encoded media with MSE.
Case 2: Digital Rights Management

Current DRM solution for WebRTC

- RTCRtpSender
  - RTCRtpScriptTransform
  - postMessage
- MediaSource
- <video>
  - setMediaKeys()
- Web Workers
  - ReadableStream
  - WritableStream
  - TransformStream
  - Source Buffer
  - Encrypted Frames
  - Fake Frames
### Case 3: H265 RTC Supporting

**HEVC/H.265 decoding support on Web**

<table>
<thead>
<tr>
<th></th>
<th>Chrome</th>
<th>Edge</th>
<th>Safari</th>
<th>Firefox</th>
<th>Opera</th>
<th>IE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-106</td>
<td>12-18</td>
<td>11-12.1</td>
<td>120</td>
<td>10-93</td>
<td>2-119</td>
<td></td>
</tr>
<tr>
<td>107-124</td>
<td>79-123</td>
<td>13-17.4</td>
<td>21-125</td>
<td>94-108</td>
<td>6-10</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>124</td>
<td>17.5</td>
<td>126</td>
<td>109</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>126-128</td>
<td>17.6-TP</td>
<td>127-129</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Chrome for Android</th>
<th>Safari on iOS</th>
<th>Samsung Internet</th>
<th>Opera Mini</th>
<th>Opera Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2-10.3</td>
<td>4</td>
<td>2</td>
<td>5-20</td>
<td>12-12.1</td>
<td></td>
</tr>
<tr>
<td>11-17.4</td>
<td>21-23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17.5</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>80</td>
</tr>
</tbody>
</table>

1. Supported only for devices with hardware support.
2. Reported to work in certain Android devices with hardware support.
3. Supported only on macOS High Sierra or later.
4. Supported for all devices on macOS (>= Big Sur 11.0) and Android (>= 5.0) if Edge >= 107, for devices with hardware support on Windows (>= Windows 10 1709) when HEVC video extensions from the Microsoft Store is installed.
5. Supported for all devices on macOS (>= Big Sur 11.0) and Android (>= 5.0), for devices with hardware support (the range is the same as Edge) on Windows in Nightly only. 10-bit or higher colors are not supported.
6. Supported for devices with hardware support powered by VAAPI on Linux and ChromeOS.
7. Supported for devices with hardware support (the range is the same as Edge) on Windows only. Enabled by default in Nightly and can be enabled via the `media.wmf.hevc.enabled` pref in `about:config`. 10-bit or higher colors are not supported.
**Case 3: H265 RTC Supporting**

### HEVC Encoding support on Web

- **Safari 13+ on Mac and iOS**

- **Chrome for Windows / Mac M109+; Chrome for Android M117+;**
  Experimental feature with switch: `--enable-features=PlatformHEVCEncoderSupport`

- **Not supported**

### WebRTC HEVC Support

- **Experimental feature on Safari 14+**
  Not compatible with RFC 7798 Packetization

- **Chrome Canary 127**

- **Not supported**
Case 3: H265 RTC Supporting

Solution 1: Remuxing RTP packets to HLS on Server

Pros
- The architecture is relatively simple
- The system has good compatibility

Cons
- Not a real-time system
- Poor resistance to weak network

Scenario
Good network quality, do not have strict real-time requirements.
Case 3: H265 RTC Supporting

Solution 2: Customized Browser with WebRTC H265 Support
**Case 3: H265 RTC Supporting**

**Solution 2: Customized Browser with WebRTC H265 Support**

**Pros**
- Pure web technical stack for developer
- Compatible with existed WebRTC apps
- Forward compatibility with future official Chrome browser

**Cons**
- Higher technical maintenance costs
- Extra efforts for client distribution

**Scenario**
Application environment under control in organization like company, bureau, etc…
Case 3: H265 RTC Supporting

Solution 3: Port RTC components to WebAssembly

Features

- Implement full downlink pipeline with WebAssembly. Including bandwidth estimation, jitter buffer, netEQ, video packetizer/depacketizer, media codecs, media renderer.
- Media is transmitted with tuned WebRTC DataChannel
Case 3: H265 RTC Supporting

Potential Future Solution: WebRTC-RtpTransport API
A new proposal being discussed in WebRTC WG

Problem & Motivation
WebRTC APIs is not sufficient, due to:

- Lack of support for custom metadata
- Lack of codec support
- Lack of custom rate control
- Inability to support custom RTCP messages

Goal
- Custom rate control (with built-in bandwidth estimate)
  - Custom bitrate allocation
- Custom metadata (header extensions)
  - Custom RTCP messages
- Custom RTCP message timing
  - RTP forwarding
- Custom payloads (ML-based audio codecs)
  - Custom packetization
    - Custom FEC
    - Custom RTX
  - Custom Jitter Buffer
- Custom bandwidth estimate
Case 4: Alpha Video Transmission

Scenario: Rendering characters onto virtual background
Case 4: Alpha Video Transmission

In WebRTC pipeline, alpha plane is ignored even if the encoder support alpha channel.
**Case 4: Alpha Video Transmission**

**Solution 1: Send alpha data with H.264 SEI**

- **H264 Encoder**
  - Insert Alpha data to SEI
- **Internet**
- **H264 Decoder**
  - Extract alpha data from SEI

*Not strict synchronized*
Solution 1: Store alpha data to the expanded area

- Need extra metadata to record video type (normal or expanded)
- Cost about 20% extra bandwidth
Thanks