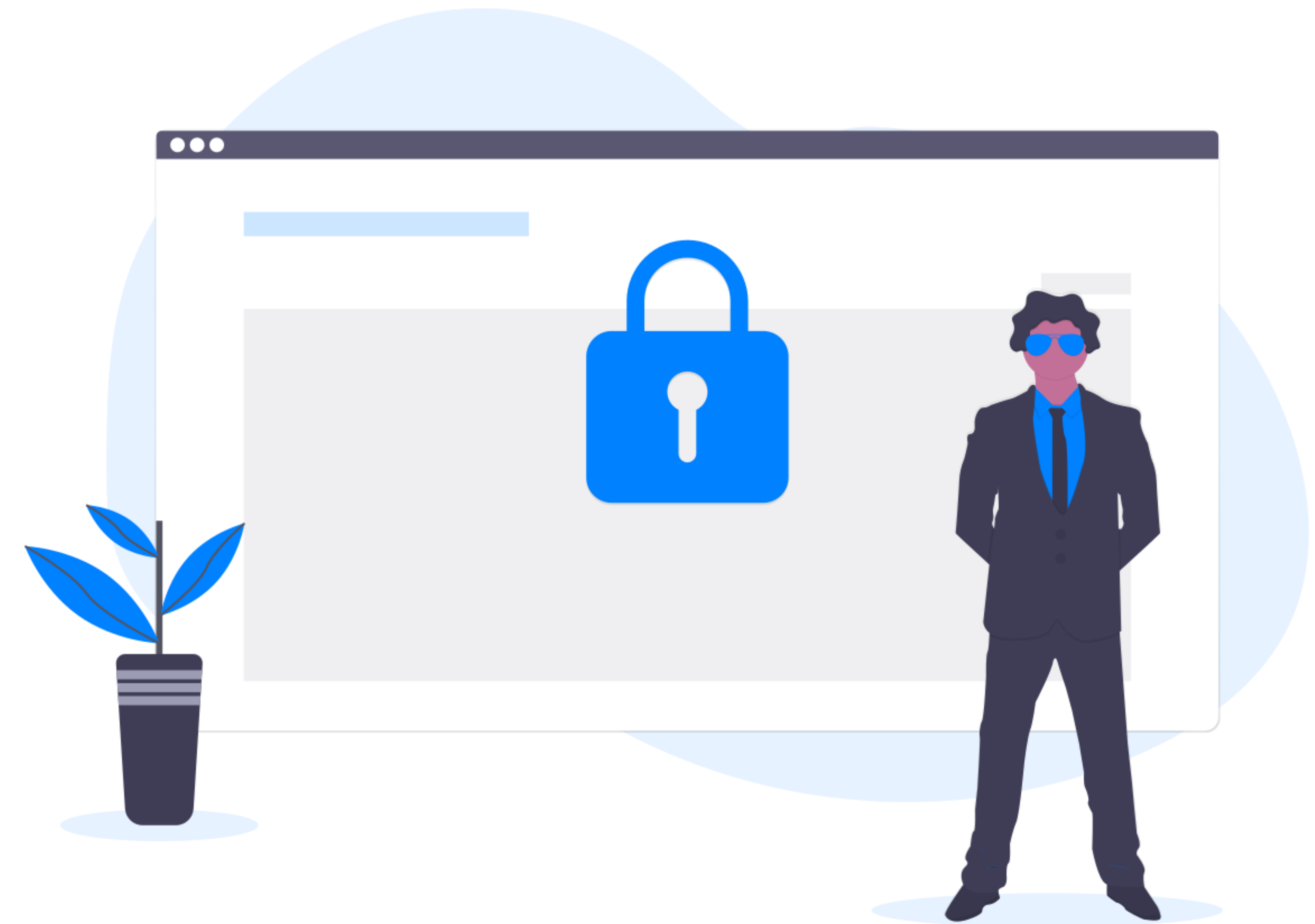
The road to End-to-End Encryption in Jitsi Meet

How we did it, and how you can do it too!

Saúl Ibarra Corretgé (saghul) | FOSDEM 2021

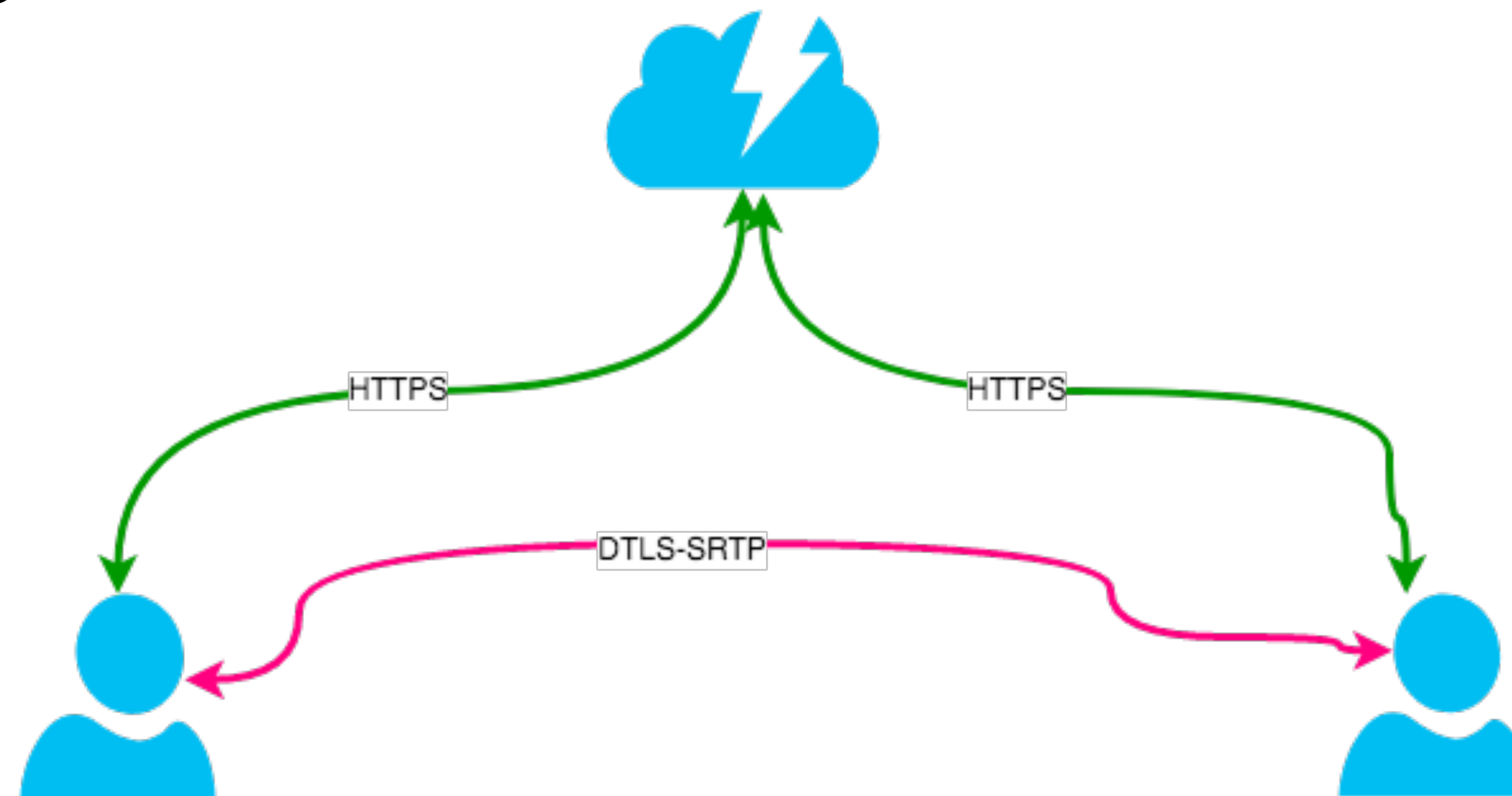
WebRTC Security model

A quick look



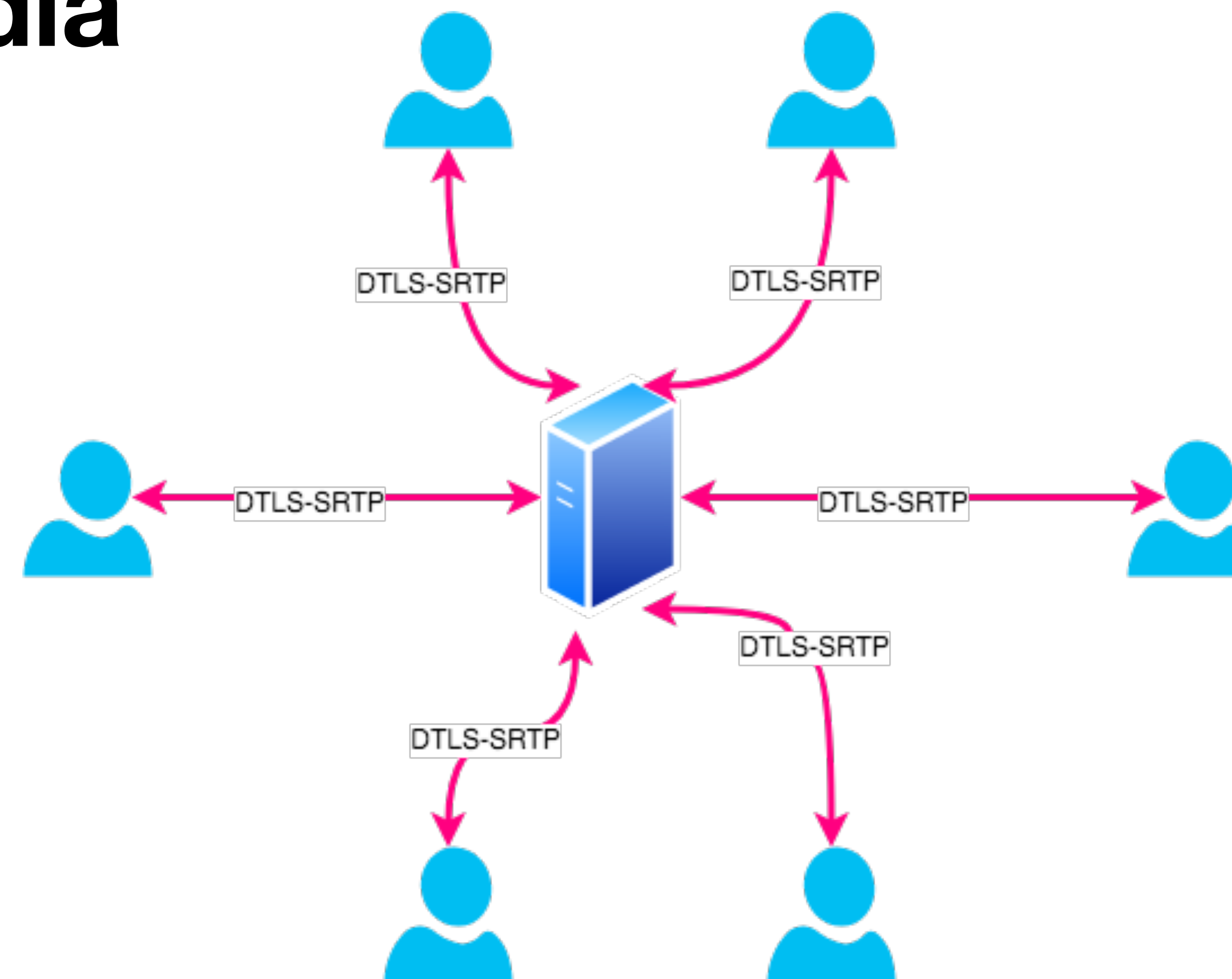
WebRTC security refresher

- Restricted to “safe origins” in browsers
- DTLS-SRTP is mandatory (RFC8829, sec 5.1.1)
- **Encrypted by design**



WebRTC with SFU architecture

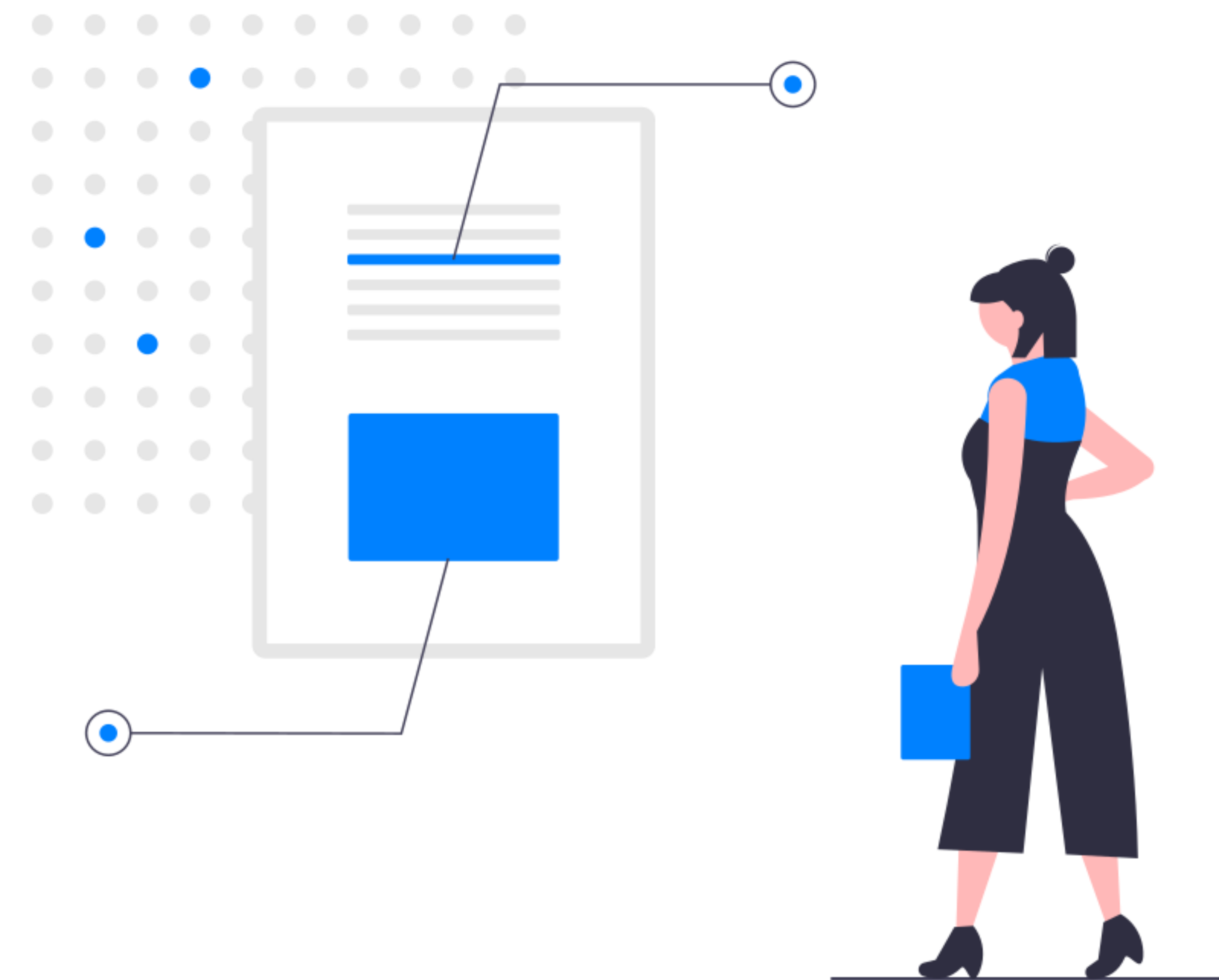
- Peer connections established with a server
- More scalable architecture
- **The server has access to the media**



SFU media processing

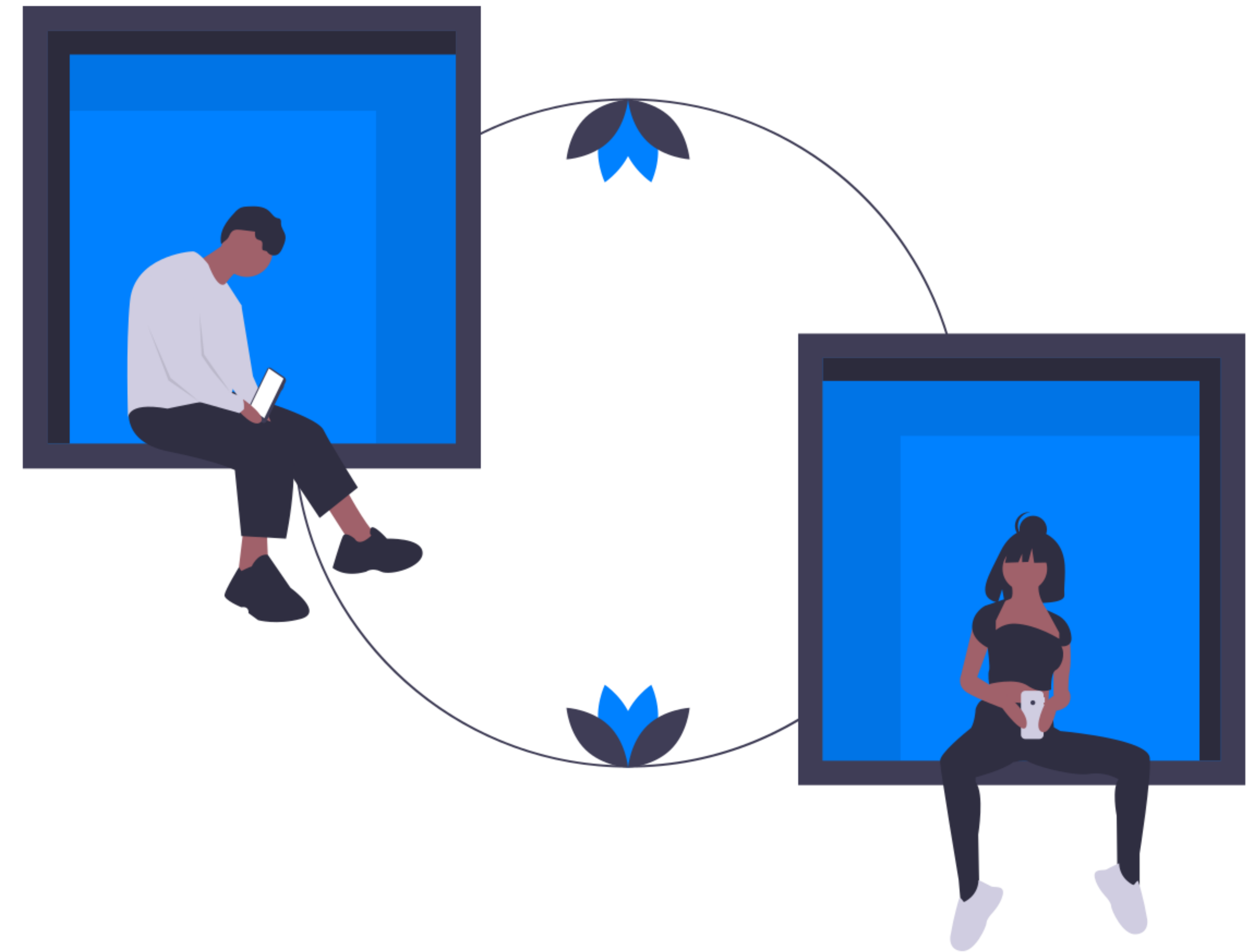
Why access to the media ~~is~~ was necessary

- Optimal video layer routing
- Keyframe detection
- Only the **packet header** is necessary



Wasn’t WebRTC end to end encrypted?

Sort of



End-to-End Encrypted…

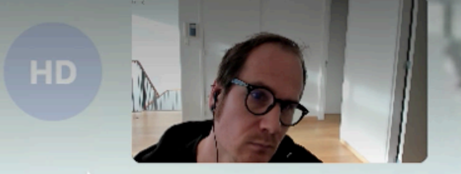
Sort of

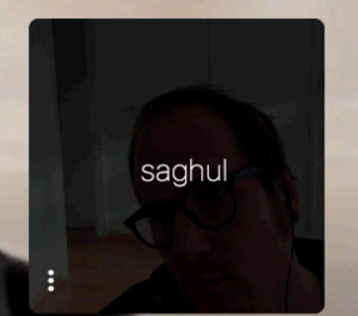
- When media is flowing Peer-to-Peer
  - But SFUs are needed for scaling
- Bad UX for certificate validation
- No indication if the tracks are swapped out

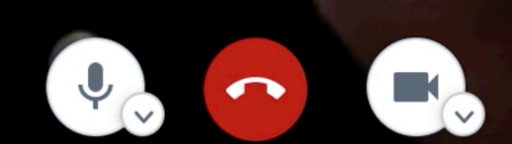
```
type: answer, sdp: v=0
o=- 1611234728580 2 IN IP4 0.0.0.0
s=-
t=0 0
a=group:BUNDLE audio video
m=audio 1 RTP/SAVPF 111 103 104 9 0 8 106 105 13 110 112 113 126
c=IN IP4 0.0.0.0
a=rtpmap:111 opus/48000/2
a=rtpmap:103 ISAC/16000
a=rtpmap:104 ISAC/32000
a=rtpmap:9 G722/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:106 CN/32000
a=rtpmap:105 CN/16000
a=rtpmap:13 CN/8000
a=rtpmap:110 telephone-event/48000
a=rtpmap:112 telephone-event/32000
a=rtpmap:113 telephone-event/16000
a=rtpmap:126 telephone-event/8000
a=fmtp:111 minptime=10; useinbandfec=1
a=rtcp:1 IN IP4 0.0.0.0
a=rtcp-fb:111 transport-cc
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=extmap:2 http://www.webrtc.org/experiments/rtp-hdrext/abs-send-time
a=extmap:3 http://www.ietf.org/id/draft-holmer-rmcat-transport-wide-cc-extensions-01
a=setup:active
a=mid:audio
a=sendrecv
a=ice-ufrag:XAiV
a=ice-pwd:RB7yTD33L6PFY83hjj0uSWjn
a=fingerprint:sha-256 A6:A2:B6:E3:E0:9D:6D:8B:6B:C2:EE:6F:47:10:34:F4:3C:0C:11:EF:DD:B6:95:7E:2A:A7:C3:87:92:5E:C8:9E
a=ssrc:1562203001 cname:PvTvS2sWMhUCiqO-2
a=ssrc:1562203001 msid:6ef4960a-88b6-4b7b-8303-944e11e0bc82-2 7742d881-027d-41bf-bc6a-7b67994769cc-2
a=ssrc:1562203001 mslabel:6ef4960a-88b6-4b7b-8303-944e11e0bc82-2
a=ssrc:1562203001 label:7742d881-027d-41bf-bc6a-7b67994769cc-2
a=rtcp-mux
m=video 1 RTP/SAVPF 96 97 98 99 100 101 114 115 116
c=IN IP4 0.0.0.0
a=rtpmap:96 VP8/90000
a=rtpmap:97 rtx/90000
a=rtpmap:98 VP9/90000
a=rtpmap:99 rtx/90000
a=rtpmap:100 VP9/90000
a=rtpmap:101 rtx/90000
a=rtpmap:114 red/90000
a=rtpmap:115 rtx/90000
a=rtpmap:116 ulpfec/90000
a=fmtp:97 apt=96
a=fmtp:98 profile-id=0
a=fmtp:99 apt=98
a=fmtp:100 profile-id=2
a=fmtp:101 apt=100
a=fmtp:115 apt=114
a=rtcp:1 IN IP4 0.0.0.0
a=rtcp-fb:96 goog-remb
a=rtcp-fb:96 transport-cc
a=rtcp-fb:96 ccm fir
a=rtcp-fb:96 nack
a=rtcp-fb:96 nack pli
a=rtcp-fb:98 goog-remb
a=rtcp-fb:98 transport-cc
a=rtcp-fb:98 ccm fir
a=rtcp-fb:98 nack
a=rtcp-fb:98 nack pli
a=rtcp-fb:100 goog-remb
a=rtcp-fb:100 transport-cc
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=extmap:14 urn:ietf:params:rtp-hdrext:toffset
a=extmap:2 http://www.webrtc.org/experiments/rtp-hdrext/abs-send-time
a=extmap:13 urn:3gpp:video-orientation
a=extmap:3 http://www.ietf.org/id/draft-holmer-rmcat-transport-wide-cc-extensions-01
a=extmap:5 http://www.webrtc.org/experiments/rtp-hdrext/playout-delay
a=extmap:6 http://www.webrtc.org/experiments/rtp-hdrext/video-content-type
a=extmap:7 http://www.webrtc.org/experiments/rtp-hdrext/video-timing
a=extmap:8 http://www.webrtc.org/experiments/rtp-hdrext/color-space
a=setup:active
a=mid:video
a=sendrecv
a=ice-ufrag:XAiV
a=ice-pwd:RB7yTD33L6PFY83hjj0uSWjn
a=fingerprint:sha-256 A6:A2:B6:E3:E0:9D:6D:8B:6B:C2:EE:6F:47:10:34:F4:3C:0C:11:EF:DD:B6:95:7E:2A:A7:C3:87:92:5E:C8:9E
a=ssrc:4205491823 cname:PvTvS2sWMhUCiqO-2
a=ssrc:4205491823 msid:0700d112-9027-4538-a6c9-bf76656f764a-2 c7f3af91-73ca-4534-986e-c9139b0cafed-2
a=ssrc:4205491823 mslabel:0700d112-9027-4538-a6c9-bf76656f764a-2
a=ssrc:4205491823 label:c7f3af91-73ca-4534-986e-c9139b0cafed-2
a=ssrc:1810154588 cname:PvTvS2sWMhUCiqO-2
a=ssrc:1810154588 msid:0700d112-9027-4538-a6c9-bf76656f764a-2 c7f3af91-73ca-4534-986e-c9139b0cafed-2
a=ssrc:1810154588 mslabel:0700d112-9027-4538-a6c9-bf76656f764a-2
a=ssrc:1810154588 label:c7f3af91-73ca-4534-986e-c9139b0cafed-2
a=ssrc-group:FID 4205491823 1810154588
a=rtcp-mux
```

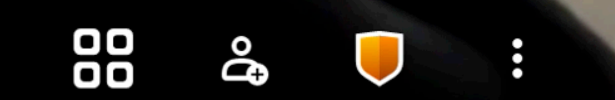
Saghul 1234567890 | Jitsi
Saghul 1234567890 | Jitsi
meet.jit.si

Saghul 1234567890
02:59

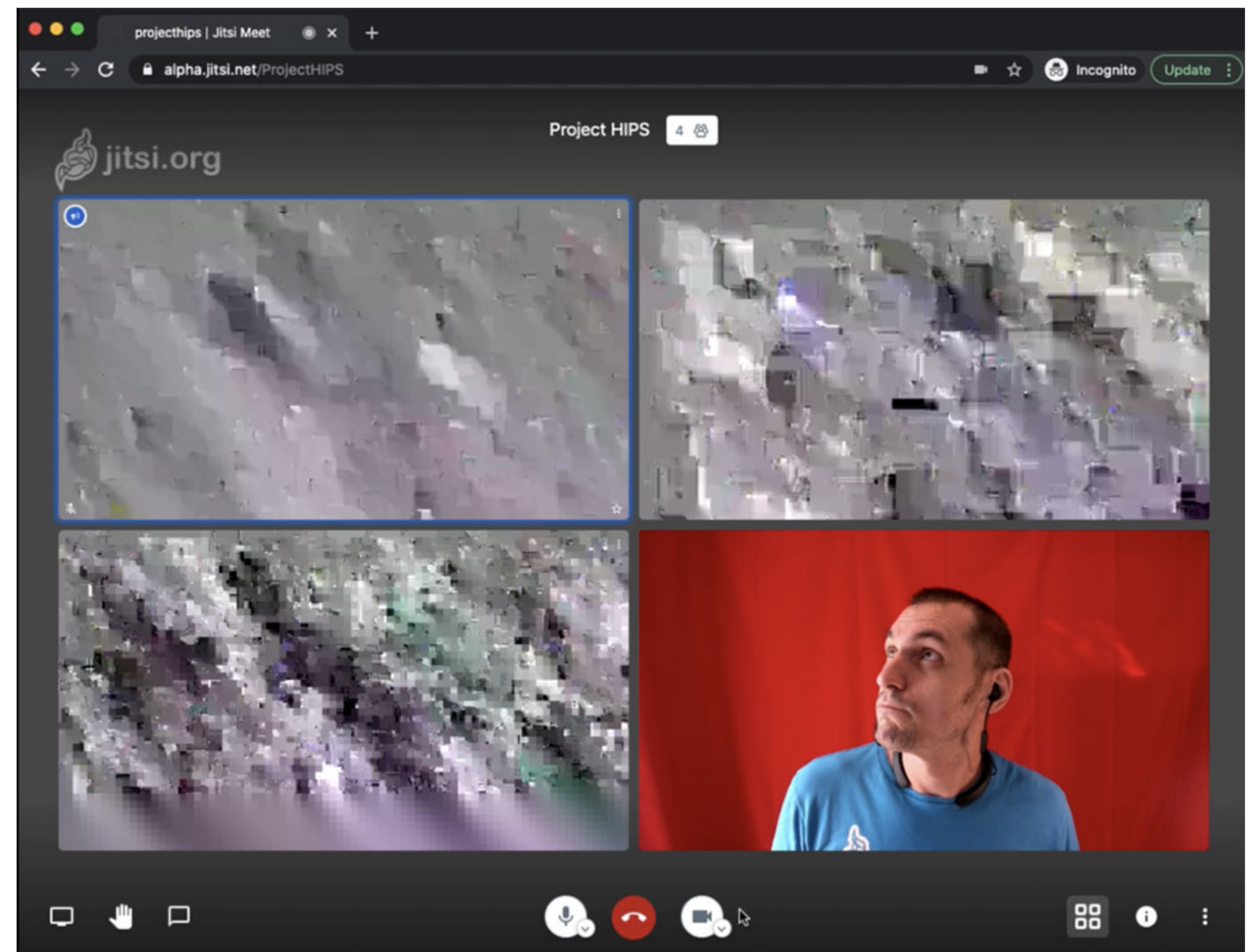








Building real E2EE for WebRTC



Why do we need E2EE?

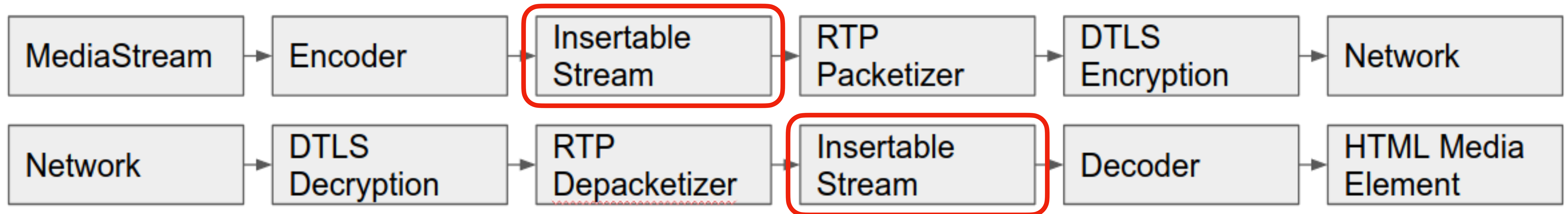
- Go check Emil Ivov’s “e2ee beyond buzzwords” talk
- Eliminate the need to trust SFUs

Insertable Streams

The API that unlocked it

- JavaScript API for manipulating **full frames**
- Data is mangled before **transport encryption**
- WebCrypto APIs can be used for encryption
- Worker friendly
- Chromium only as of today

Insertable Streams



Insertable Streams

Encrypting transform

```
// Create a PeerConnection
this.pc = new RTCPeerConnection({ encodedInsertableStreams: true });

// Add the stream and encrypt it
stream.getTracks().forEach((track) => {
   const sender = this.pc.addTrack(track, stream);
   const insertableStreams = sender.createEncodedStreams();
   const transformer = new TransformStream({ transform: encrypt });

   insertableStreams.readableStream
     .pipeThrough(transformer)
     .pipeTo(insertableStreams.writableStream);
});
```

```
function encrypt(chunk, controller) {
    // AES encrypt with WebCrypto APIs ...
    controller.enqueue(chunk);
}
```

Insertable Streams

Decrypting transform

```
// Handle remote tracks and decrypt them
peerConnection.ontrack = e => {
     const transformer = new TransformStream({ transform: decrypt });
     const insertableStreams = e.receiver.createEncodedStreams();

     insertableStreams.readableStream
         .pipeThrough(transformer)
         .pipeTo(insertableStreams.writableStream);
 };
```

```
function decrypt(chunk, controller) {
    // AES decrypt with WebCrypto APIs...
    controller.enqueue(chunk);
}
```

Don’t roll your own crypto

Hello SFrame!

```
+------------+------------------------------------------+^+
  |S|LEN|X|KID |         Frame Counter                   | |
+^+------------+------------------------------------------+ |
| |                                                       | |
| |                                                       | |
| |                                                       | |
| |                                                       | |
| |                   Encrypted Frame                     | |
| |                                                       | |
| |                                                       | |
| |                                                       | |
| |                                                       | |
+^+-------------------------------------------------------+^+
| |                 Authentication Tag                    | |
| +-------------------------------------------------------+ |
|                                                           |
|                                                           |
+----+Encrypted Portion            Authenticated Portion+---+
```

SFrame

End-to-end encryption and authentication for media frames

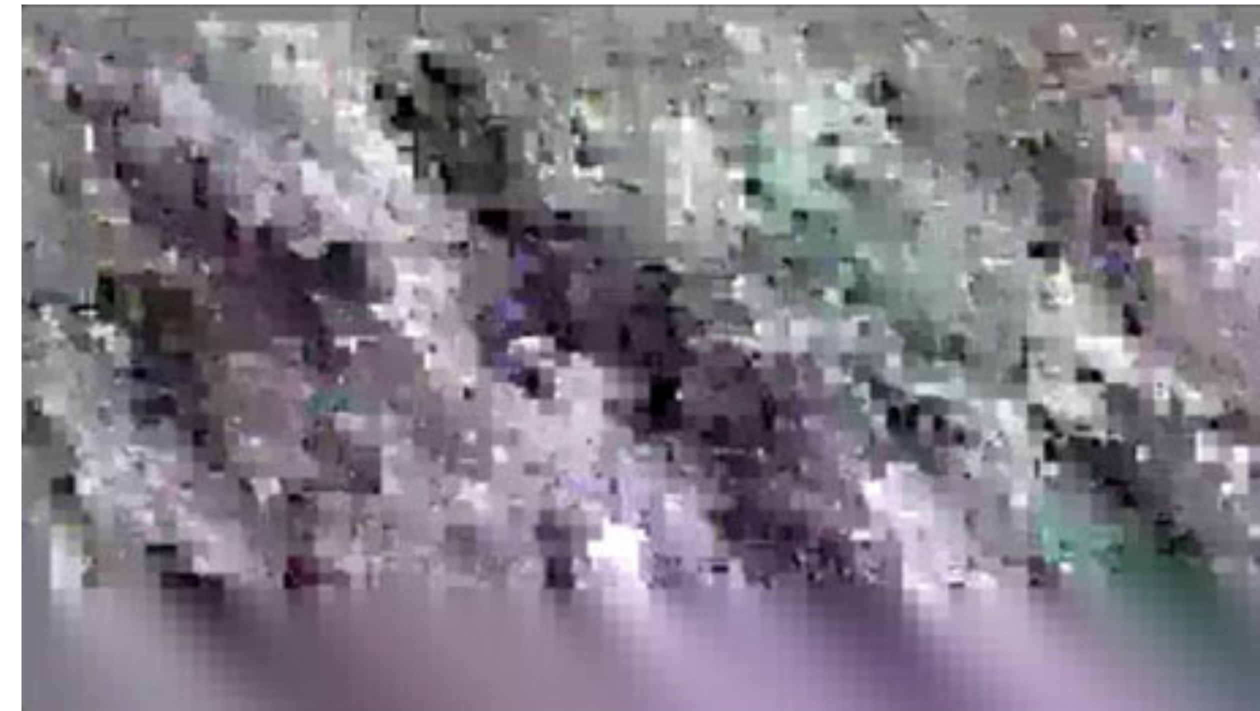
- draft-omara-sframe (early stages)
- IETF WG formed
- Apple experimenting with a native implementation
- Bring your own key management (MLS, Signal, olm, other)

Insertable Streams in Jitsi Meet

- Encryption keys
  - AES-CTR 256bit + HMAC SHA-256 (truncated)
- Signing keys
  - ECDSA P-521
- “JFrame”, a slight variation of SFrame
- All encryption happens in a Worker

Insertable Streams

The Result



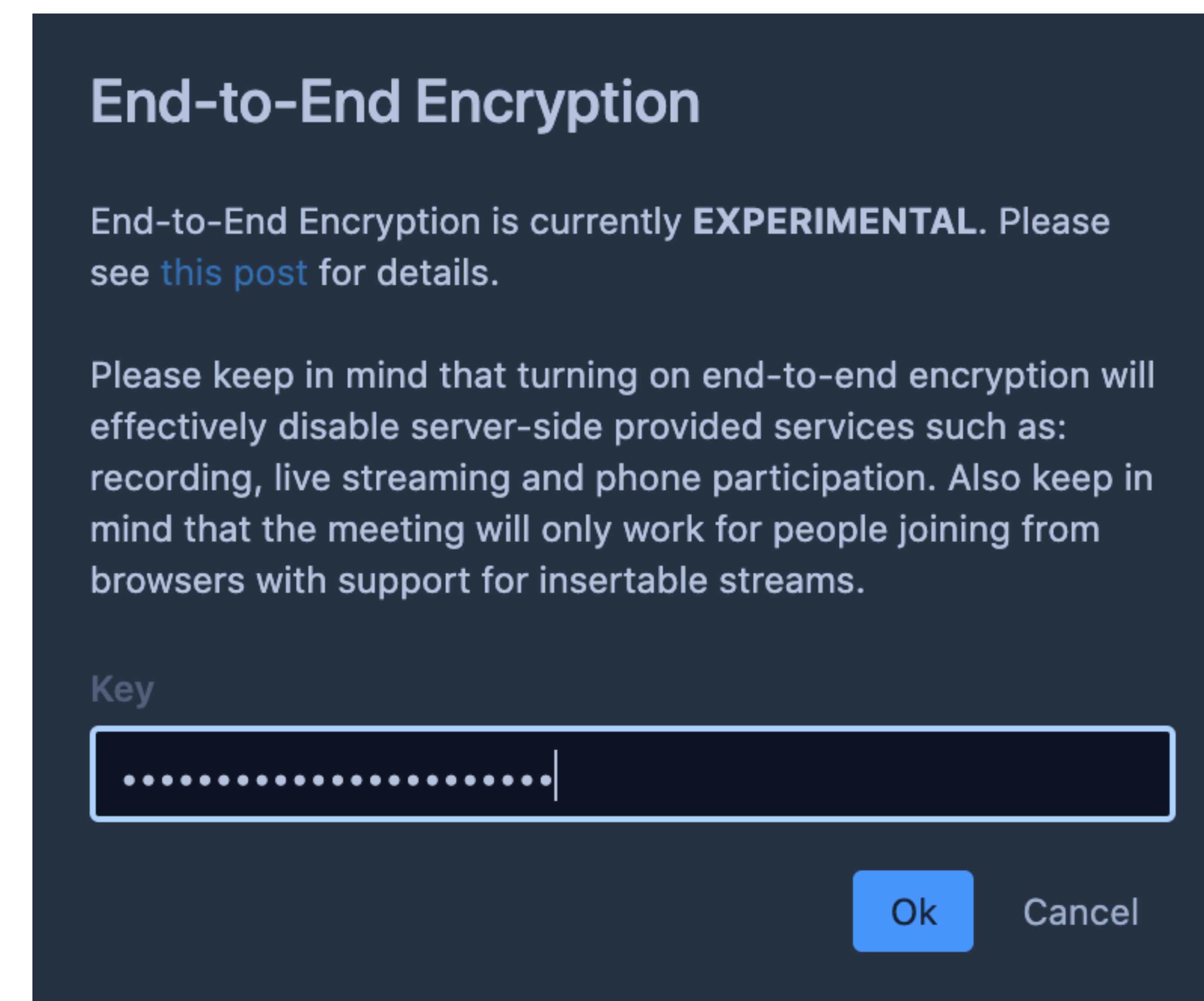
Key management

The missing piece

Unmanaged

Shared passphrase

- Users type a shared passphrase obtained out-of-band
- Encryption key is derived from the passphrase using PBKDF2
- The key never leaves the user machine



Managed

Hello olm!

- E2EE channel using Matrix's libolm
- Randomly generated per-participant keys
- Automatic key rotation and ratcheting
- Keys are exchanged using the olm channel
- User verification using SAS

[matrix]

End-to-End Encryption is currently EXPERIMENTAL. Please keep in mind that turning on end-to-end encryption will effectively disable server-side provided services such as: recording, live streaming and phone participation. Also keep in mind that the meeting will only work for people joining from browsers with support for insertable streams.

WARNING: Not all participants in this meeting seem to have support for End-to-End encryption. If you enable it they won't be able to see nor hear you.

Enable End-to-End Encryption

Implementation

Show me the code!

- Self-contained in lib-jitsi-meet
- ~1000 lines of code
- https://github.com/jitsi/lib-jitsi-meet/tree/master/modules/e2ee

Future

What’s next?

- Finish SAS validation
- Bring back unmanaged mode and make it configurable
- Collaborate with the IETF SFrame WG
- UI/UX polish

Thanks

We didn’t do this alone

- Philipp Hancke, for working with us to make E2EE possible in Jitsi Meet
- Matrix, for libolm and the help understanding how to use it properly
- Google, for championing the insertable streams effort
- Our community, for all the love and support

@jitsinews | @saghul