EBERHARD KARLS UNIVERSITÄT TÜBINGEN

FACULTY OF SCIENCE
Communication Networks

Draft: Secure OIDC Authentication for WebRTC (SOAR)

by Jonas Primbs, Chair of Communication Networks, University of Tübingen, Germany

http://kn.inf.uni-tuebingen.de

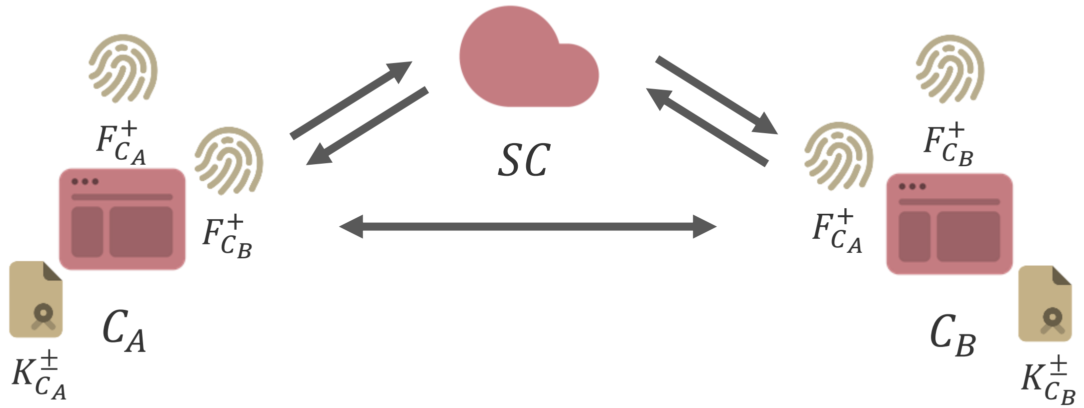
Outline

- Introduction to WebRTC
- Motivation
- Proposed solution
  - Overview
  - Authentication
  - Connection establishment
- Conclusion
- Discussion

Introduction to WebRTC

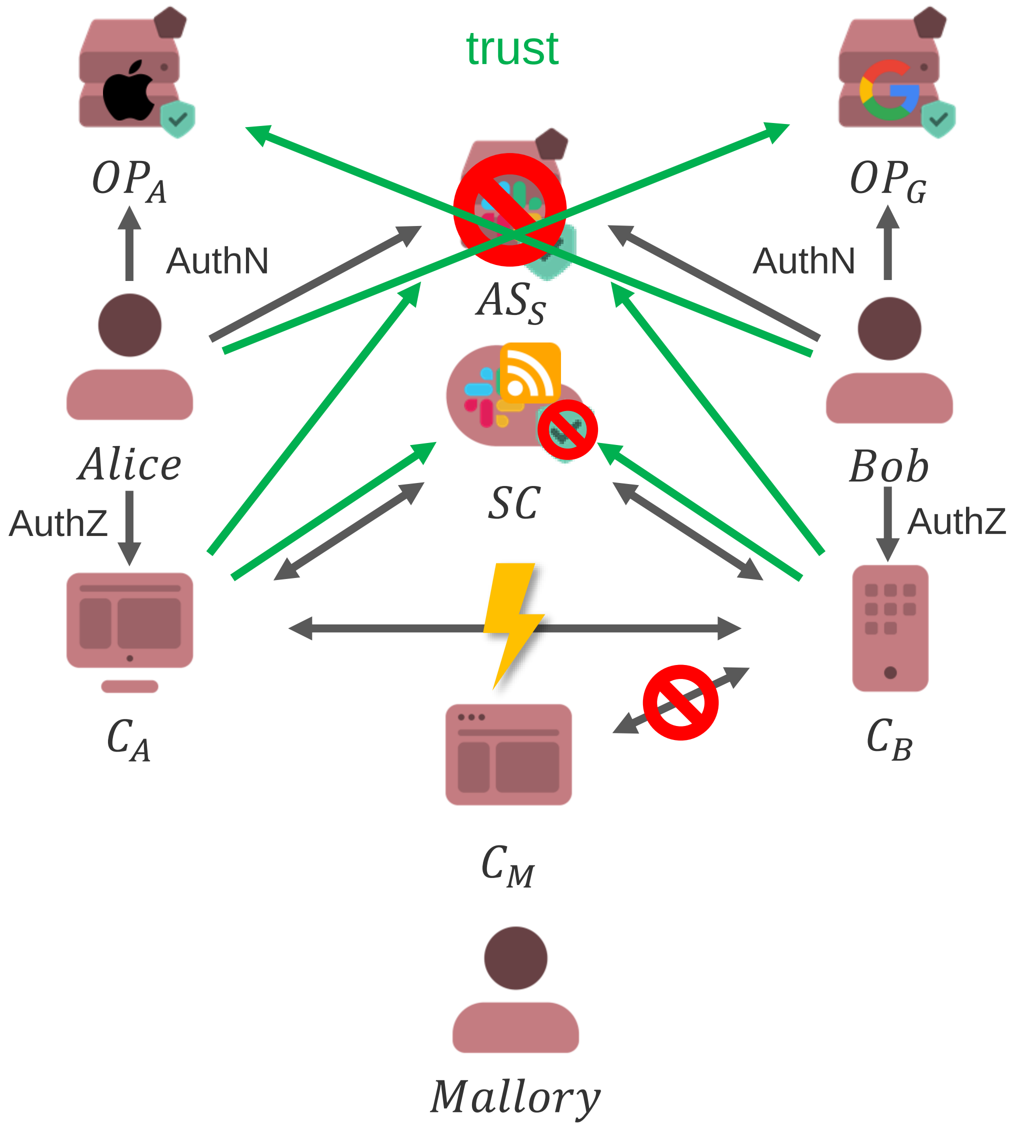
- Open P2P **Web** standard for **R**eal-**T**ime **C**ommunication
  - Standardized by W3C and IETF
  - Like VoIP/SIP, but simpler
  - Widely used for video telephony, collaboration, gaming, …
  - Supported by all major browsers
    - 95,77% of all users by 11/2021
    - Source: caniuse.com

- **Operation of WebRTC**
  - Each Peer generates an unsigned X.509 certificate
    - Only for identification of Peers
    - Peers cannot access $K^{\pm}$ but fingerprint F^{+}
  - Peers exchange connection information (IP addresses, fingerprints, …) via (un)trusted Signaling Channel SC
  - Peers establish secure DTLS-protected P2P channels



Motivation

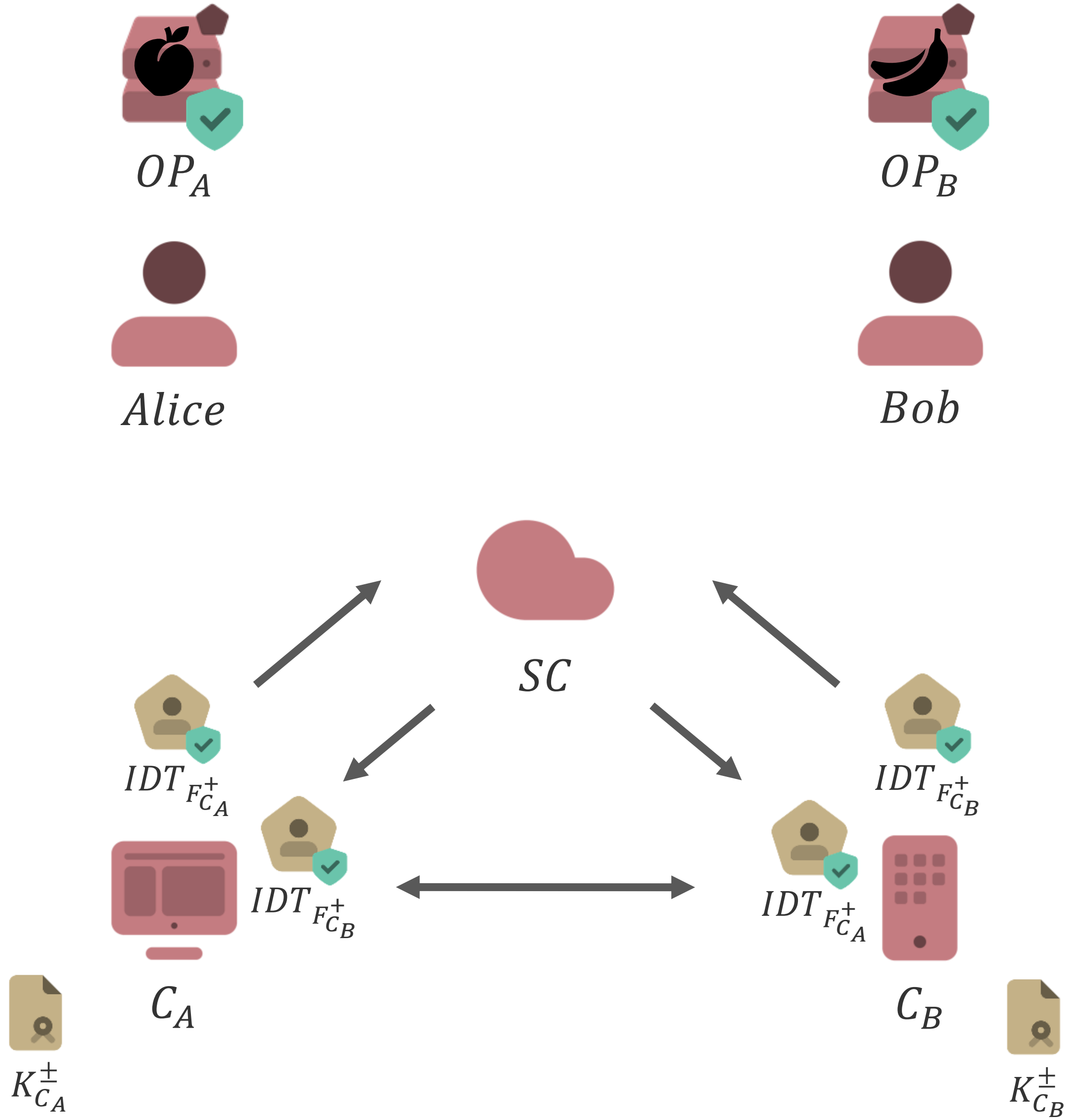
- **Example: Alice calls Bob**
- Alice uses her Client C_A to call Bob on Client C_B
  - In peer-to-peer (P2P) environment
- Q: How does Bob know that the call comes from Alice's Client C_A and not from Mallory's Client C_M?
- **A: Look at Slack!**
  - Alice and Bob authenticate with their OPs to AS_S
  - Alice and Bob authorize their Clients C_A and C_B to access Protected Resources on SC
    - AS_S issues Access Tokens
    - ~~AS_S must be trusted~~
  - SC validates Access Tokens and forwards Session Descriptions to authorized Clients
    - ~~SC must be trusted~~
- **New Solution**
  - No centralized AS required → only trusted OPs
  - No trusted SC and client authorization required



Solution Overview

- Concept primarily for WebRTC
  - Adaptable to other P2P use cases

1. Alice and Bob authenticate themselves to their OpenID Providers OPs
2. OP_A and OP_B issue ID Tokens $IDT_{F^+_{C_A}}$ and $IDT_{F^+_{C_B}}$ including Fingerprints of corresponding WebRTC Certificates $F^+_{C_A}$ and $F^+_{C_B}$
3. Clients exchange session descriptions and ID Tokens via SC
4. Clients verify user identities and establish connection



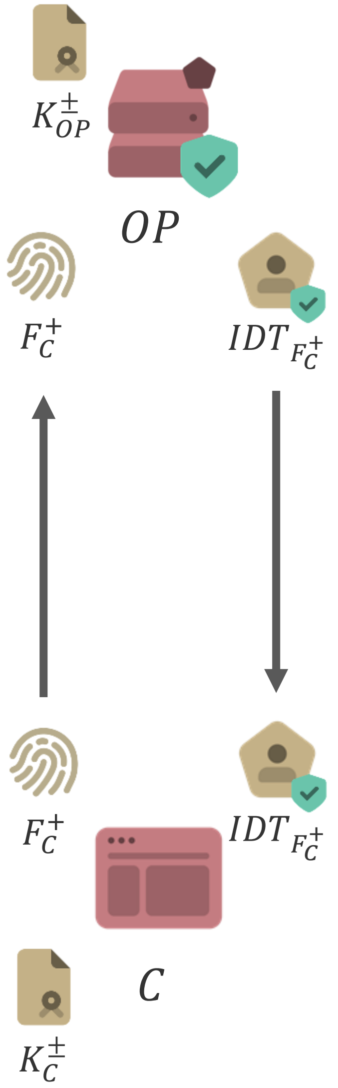
Authentication

1. **Certificate Generation**
   - Client C uses WebRTC API to generate unsigned X.509 certificate $K_C^{\pm}$
   - C extracts public key fingerprint F_C^+

2. **Token Request**
   - C requests Certificate-bound ID Token $IDT_{F_C^+}$
   - C provides F_C^+ NEW FEATURE!

3. **Token Response**
   - OP generates $IDT_{F_C^+}$ which includes F_C^+
   - OP signs it with its private key K_{OP}^-
   - OP issues $IDT_{F_C^+}$ to C

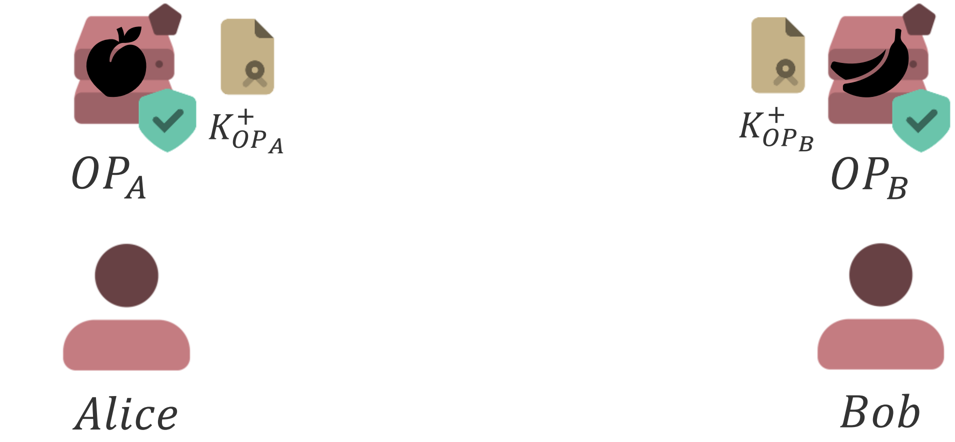
- ▶ Alice does this with C_A, $K_{C_A}^{\pm}$, $F_{C_A}^+$, and $IDT_{F_{C_A}^+}$ at OP_A
- ▶ Bob does this with C_B, $K_{C_B}^{\pm}$, $F_{C_B}^+$, and $IDT_{F_{C_B}^+}$ at OP_B



Connection Establishment (1)

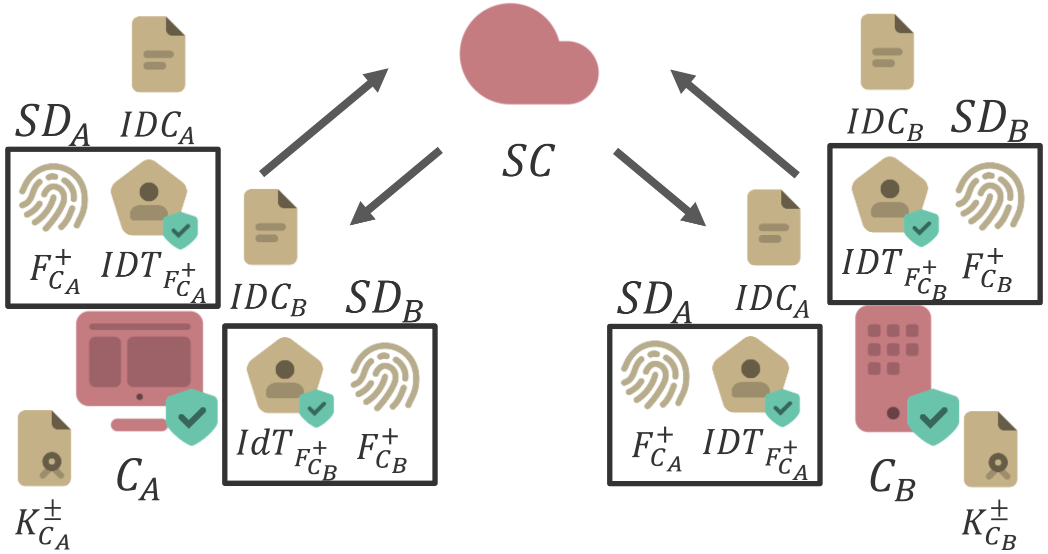
1. ID Challenge Exchange

- C_A generates ID Challenge $IDC_A = hash(IDT_{F^+_{C_A}})$
- C_A sends IDC_A via SC to C_B
- C_B generates ID Challenge $IDC_B = hash(IDT_{F^+_{C_B}})$
- C_B sends IDC_B via SC to C_A



2. Session Description Exchange

- C_A generates session description offer SD_A
  - Contains $F^+_{C_A}$ and $IDT_{F^+_{C_A}}$
- C_A sends SD_A via SC to C_B
- C_B applies SD_A if validation (next slide) successful
- C_B generates session description answer SD_B
  - Contains $F^+_{C_B}$ and $IDT_{F^+_{C_B}}$
- C_B sends SD_B via SC to C_A
- C_A applies SD_B if validation (next slide) successful

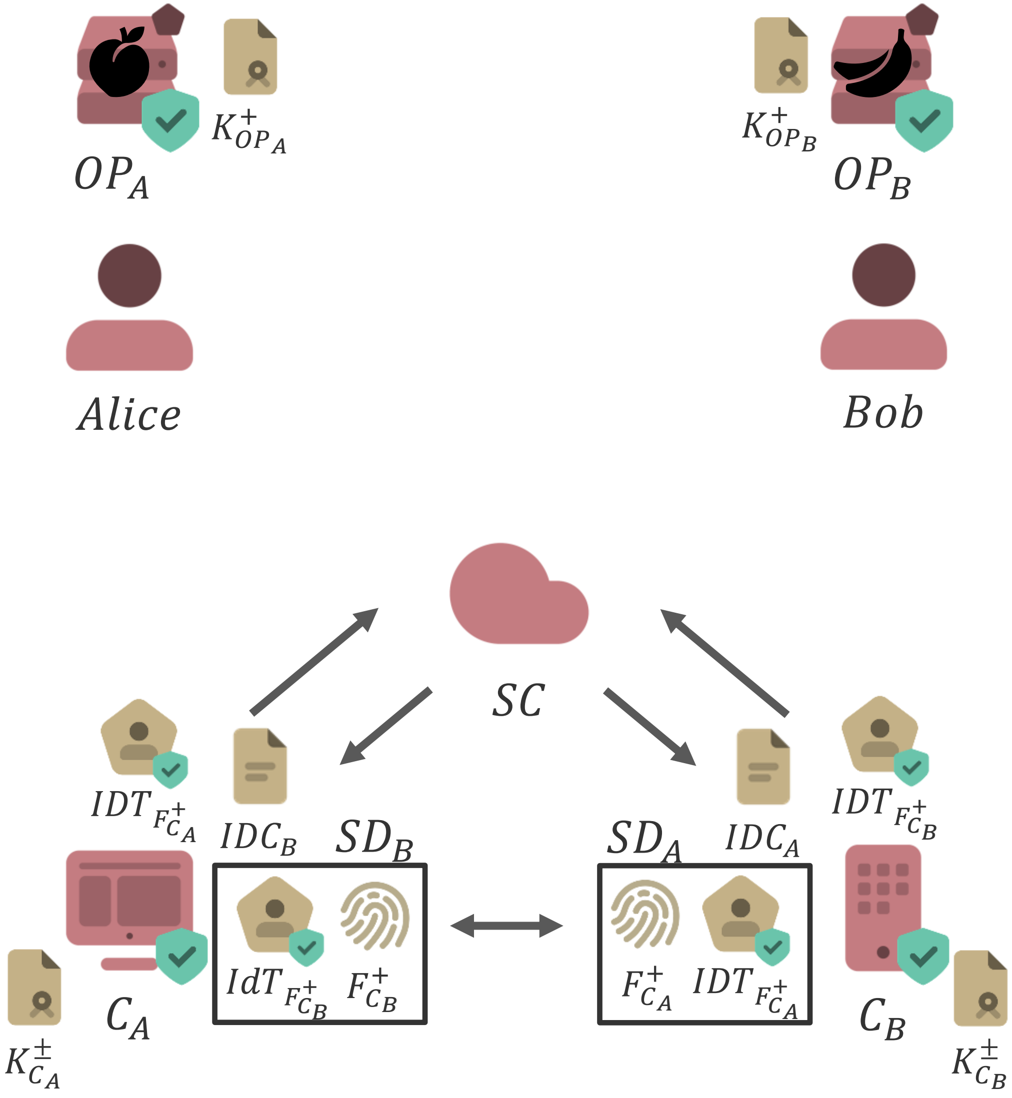


Connection Establishment (2)

- **Four validation steps**
  - Individual for each Client
  1. **ID Challenge Verification**
     - Received ID Challenge must correspond to received ID Token: $IDC = Hash(IDT_{F_C^+})$
  2. **ID Token Validation**
     - ID Token must be valid (see OIDC standard)
     - ID Token issuer (OP) must be trusted
  3. **Fingerprint Verification**
     - Fingerprint in ID Token $IDT_{F_C^+}$ must be equal to Fingerprint F_C^+ of Session Description SD

3. **DTLS Handshake = 4. Certificate Verification**
   - After C_A and C_B have applied Session Descriptions, WebRTC performs DTLS handshake
     - Verifies whether F_C^+ from SD matches K_C^+

- Successful connection proves possession of $K_{C_A}^-$ / $K_{C_B}^-$



Conclusion

- **Opportunities**
  - Decentralized and federated authentication
  - With OpenID Connect
  - In a P2P environment

- **Advantages**
  - No joint Authorization Server required for Alice and Bob
  - Alice and Bob decide which remote Client's OpenID Provider they trust
  - No trusted Signaling Channel or dedicated Signaling Server required

Discussion: What's Next?

- **Certificate-bound ID Tokens**
  - Inclusion of certificate fingerprint in ID Token required
  - Standardization by OIDF?
  - Request procedure at OP's Token Endpoint
    - Proposed solution:
      Additional POST body parameters in Token Request
      `x5t_val=[base64url encoded fingerprint]&`
      `x5t_alg=S256`
  - Encoding specification in ID Token required
    - Proposed solution (like in RFC 8705):

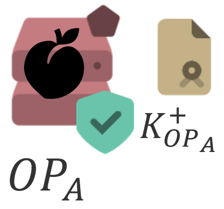
```
"cnf":{
  "x5t#S256": "[base64url encoded fingerprint]"
}
```

- **Transfer of ID Token via Session Description Protocol (SDP)**
  - Additional SDP attribute for ID Token required
  - Standardization by IETF?
    - Proposed solution (like in RFC 8827):
      `a=identity:[ID Token]`

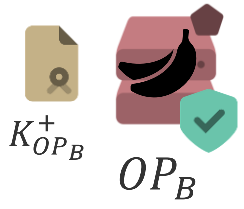
- Q: Is this also compatible to GNAP? If not: How can we achieve this?

Suggestions?

Let me know!

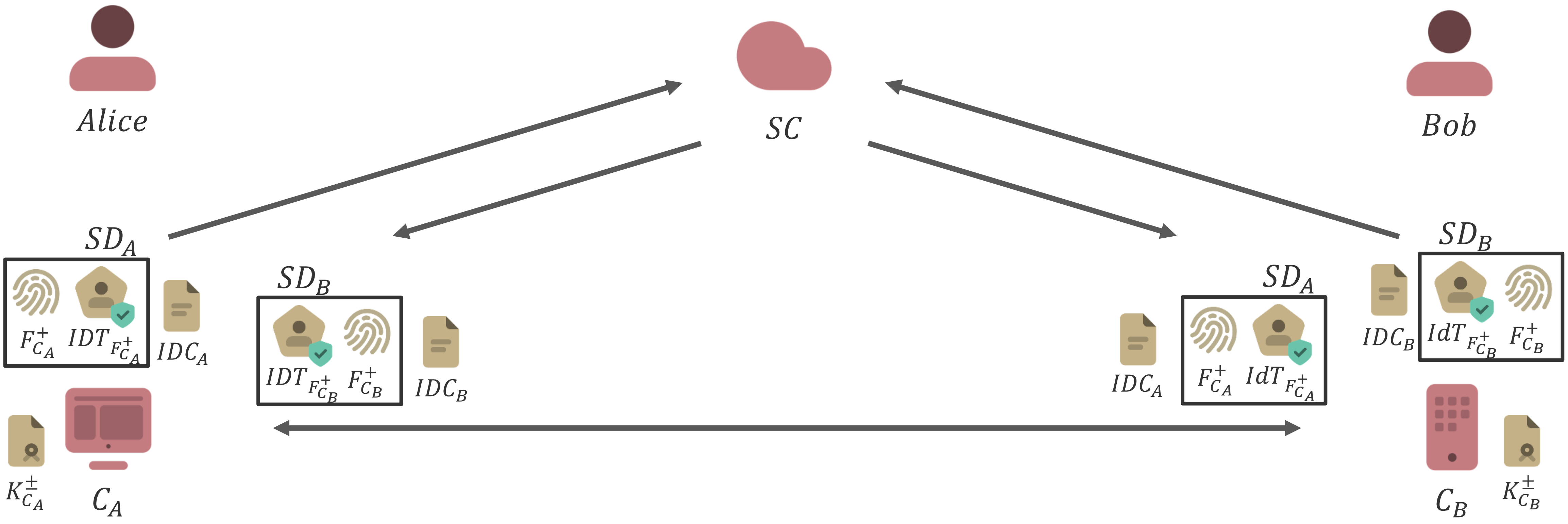


Time for questions!



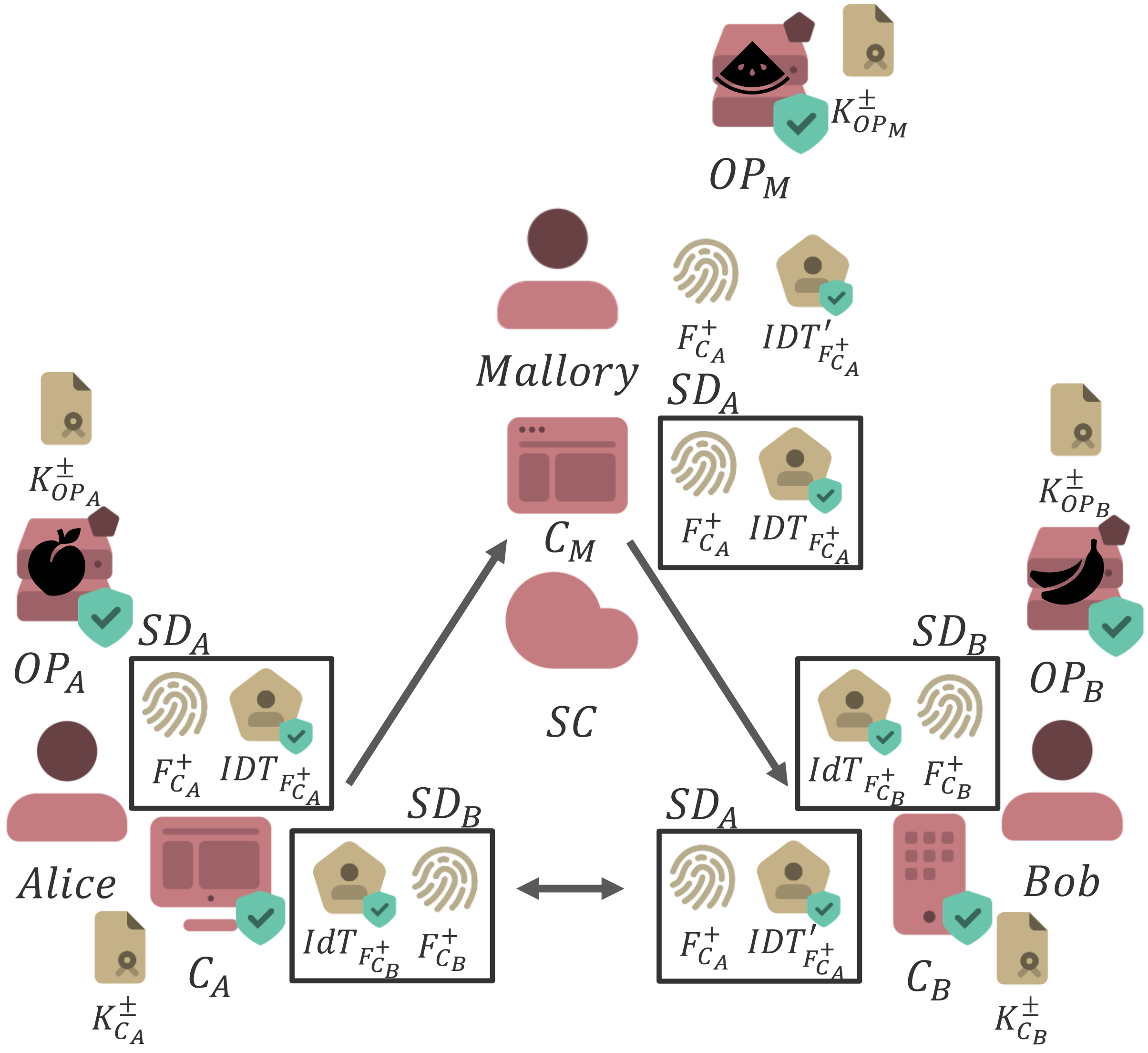
Feedback and contributions welcome!

Contact me: jonas.primbs@uni-tuebingen.de



Why use ID Challenge?

- Without ID Challenge, Mallory can exchange ID Token
  - C_A sends SD_A via SC to C_B
  - Mallory intercepts SD_A and extracts $F^+_{C_A}$
  - Mallory requests $IDT'_{F^+_{C_A}}$ from OP_M
    - $IDT'_{F^+_{C_A}}$ authenticates Mallory with C_A's certificate
  - Mallory exchanges $IDT_{F^+_{C_A}}$ by $IDT'_{F^+_{C_A}}$ and forwards everything else
  - Back-channel untouched
- Bob thinks that he has a connection to Mallory instead of Alice

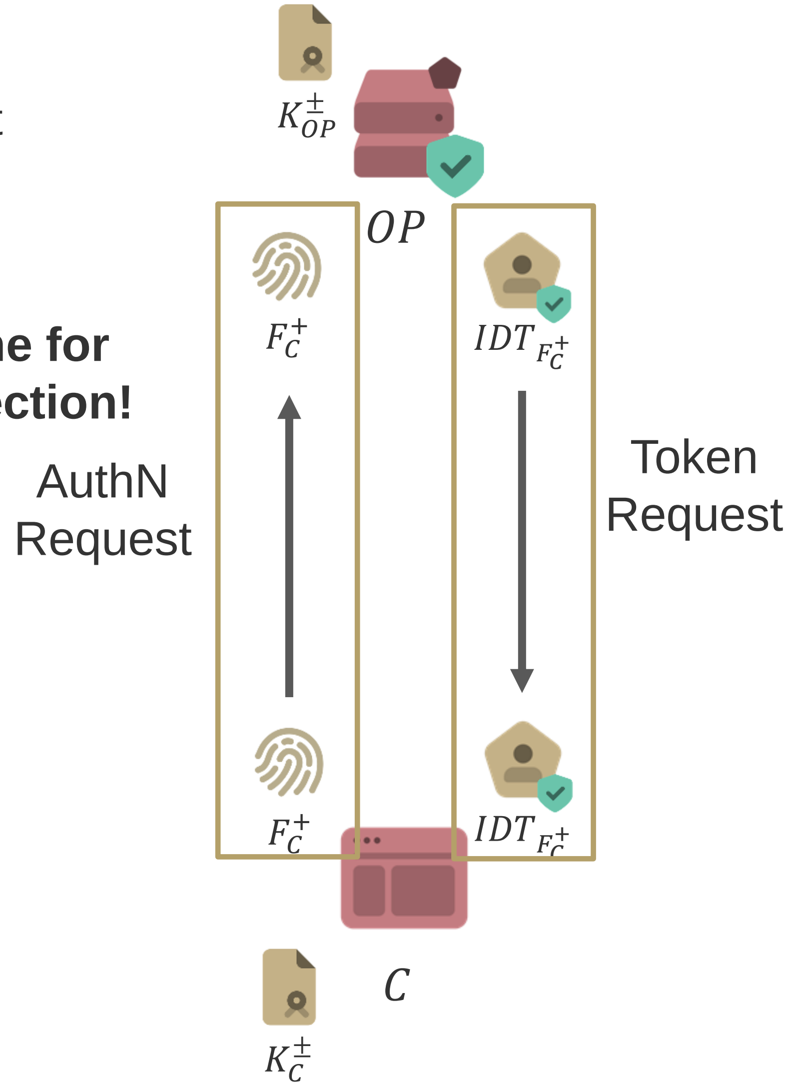


The "Hacky Solution"

- **Implementation with current OIDC specs possible!**
  - Fingerprint can be provided as Nonce value in Authentication Request

- **Disadvantages**
  - Unconventional use of Nonce
    - But standard-conform
    - May collide with faulty legacy implementations
  - Two requests required for each ID Token
    - Authentication Request (GET)
      - May require user interaction
      - Full GET request must not exceed 2048 characters
    - Token Request (POST)

This must be done for every peer connection!



Former Solutions

1. **Trusted Signaling Server SS → See Slack**
   - Centralized SS validates Access and Identity Tokens of Clients
   - Clients trust SS to forward session descriptions only to authenticated clients
   - May be a specific 3rd party application
   - Requires AS that SS trusts to

2. **Security Assertions**
   - Client stores its own fingerprint as Security Assertion on centralized AS
   - Remote Client accesses fingerprint for validation with Authorization Code from Client and validates Client's identity
   - Centralized AS required, trusted by every Client