PERC and WebRTC

`draft-roach-perc-webrtc-00`

IETF 98 – Chicago, Illinois, USA

PERC Working Group

Adam Roach

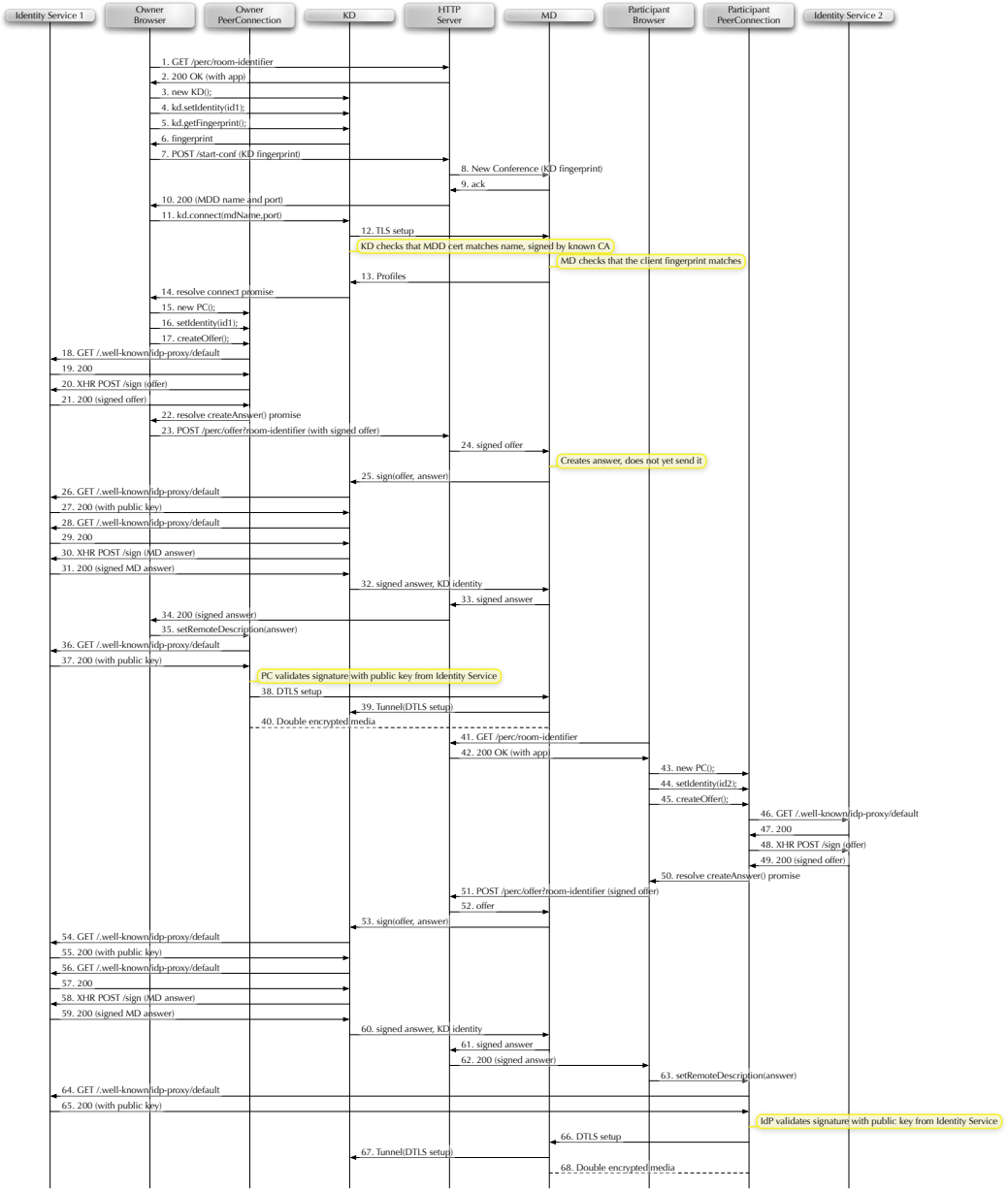
Purpose and Introduction

- Validate that the PERC model works with WebRTC
- Identify what additional protocol mechanisms (if any) are required
- Current version is call flows (ladder diagrams) with labeling for each steps
- Note: Key goal is to minimize the changes of an already-WebRTC enabled endpoint; in particular, this means using WebRTC identity

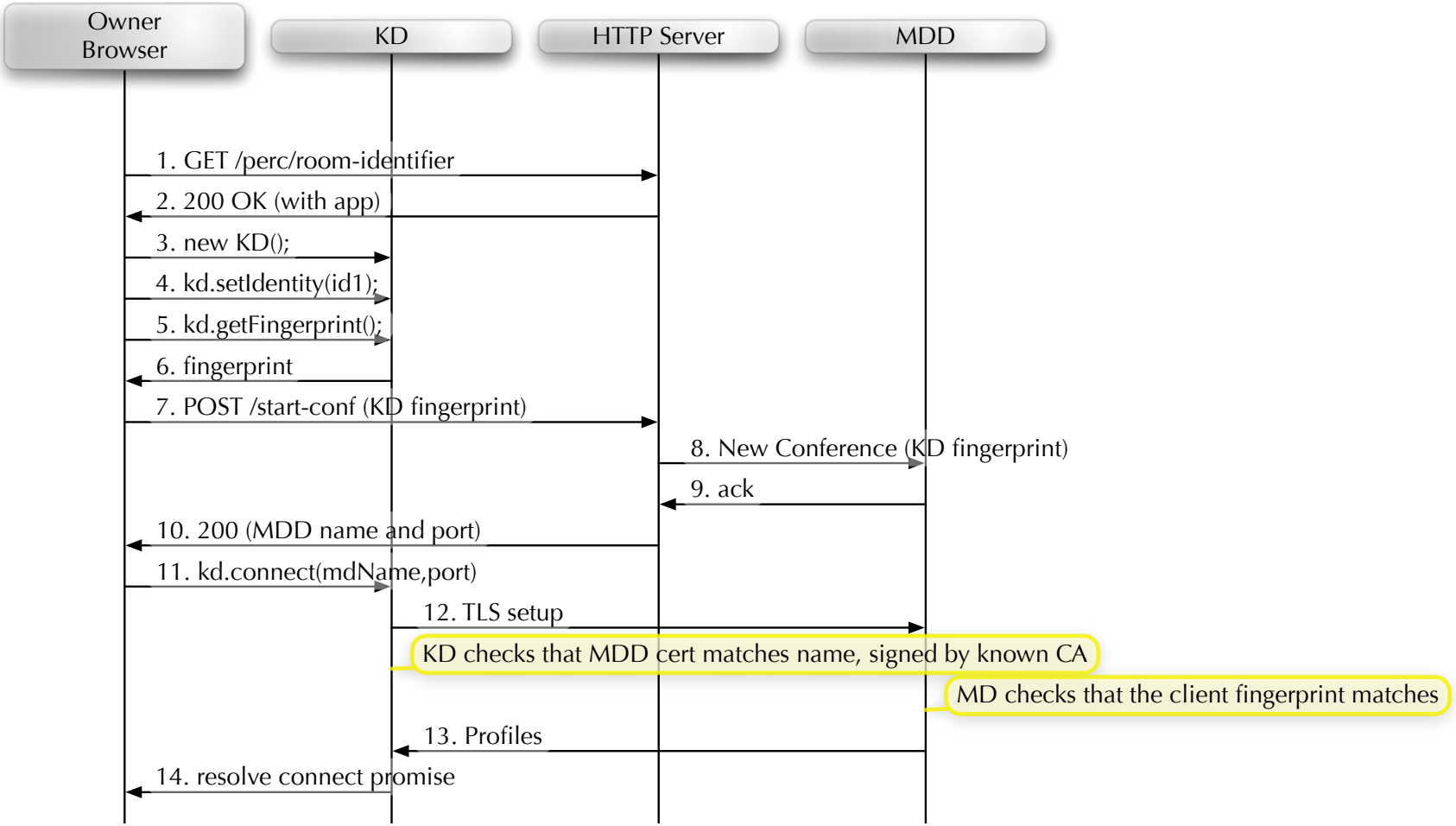
Important Notes

1. Key goal is to minimize the changes of an already-WebRTC enabled endpoint; in particular, this means using WebRTC identity
2. Diagrams show MD as a monolithic node. In practice, this will probably have a controller and a media handler, separated by a proprietary protocol. Although these flows may have bearing on this proprietary protocol, we do not call them out.

Overview

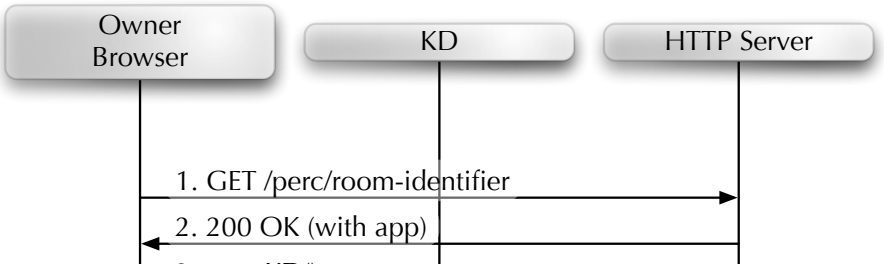

Identity Service 1
Owner Browser
Owner PeerConnection
KD
HTTP Server
MD
Participant Browser
Participant PeerConnection
Identity Service 2
1. GET /perc/room-identifier
2. 200 OK (with app)
3. new KD();
4. kd.setIdentity(id1);
5. kd.getFingerprint();
6. fingerprint
7. POST /start-conf (KD fingerprint)
8. New Conference (KD fingerprint)
9. ack
10. 200 (MDD name and port)
11. kd.connect(mdName,port)
12. TLS setup
KD checks that MDD cert matches name, signed by known CA
MD checks that the client fingerprint matches
13. Profiles
14. resolve connect promise
15. new PC();
16. setIdentity(id1);
17. createOffer();
18. GET /.well-known/idp-proxy/default
19. 200
20. XHR POST /sign (offer)
21. 200 (signed offer)
22. resolve createAnswer() promise
23. POST /perc/offer?room-identifier (with signed offer)
24. signed offer
Creates answer, does not yet send it
25. sign(offer, answer)
26. GET /.well-known/idp-proxy/default
27. 200 (with public key)
28. GET /.well-known/idp-proxy/default
29. 200
30. XHR POST /sign (MD answer)
31. 200 (signed MD answer)
32. signed answer, KD identity
33. signed answer
34. 200 (signed answer)
35. setRemoteDescription(answer)
36. GET /.well-known/idp-proxy/default
37. 200 (with public key)
PC validates signature with public key from Identity Service
38. DTLS setup
39. Tunnel(DTLS setup)
40. Double encrypted media
41. GET /perc/room-identifier
42. 200 OK (with app)
43. new PC();
44. setIdentity(id2);
45. createOffer();
46. GET /.well-known/idp-proxy/default
47. 200
48. XHR POST /sign (offer)
49. 200 (signed offer)
50. resolve createAnswer() promise
51. POST /perc/offer?room-identifier (signed offer)
52. offer
53. sign(offer, answer)
54. GET /.well-known/idp-proxy/default
55. 200 (with public key)
56. GET /.well-known/idp-proxy/default
57. 200
58. XHR POST /sign (MD answer)
59. 200 (signed MD answer)
60. signed answer, KD identity
61. signed answer
62. 200 (signed answer)
63. setRemoteDescription(answer)
64. GET /.well-known/idp-proxy/default
65. 200 (with public key)
IdP validates signature with public key from Identity Service
66. DTLS setup
67. Tunnel(DTLS setup)
68. Double encrypted media

4.1.1 Conference Establishment



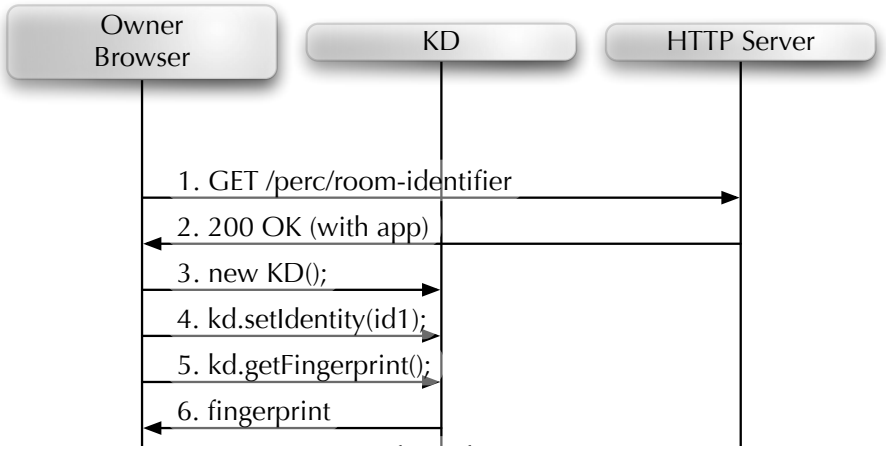
4.1.1 Conference Establishment

1. The conference owner loads the conference application. Although not required, information sufficient to identify the conference is frequently sent as part of the URL.
2. The HTTP server returns the conferencing web application.



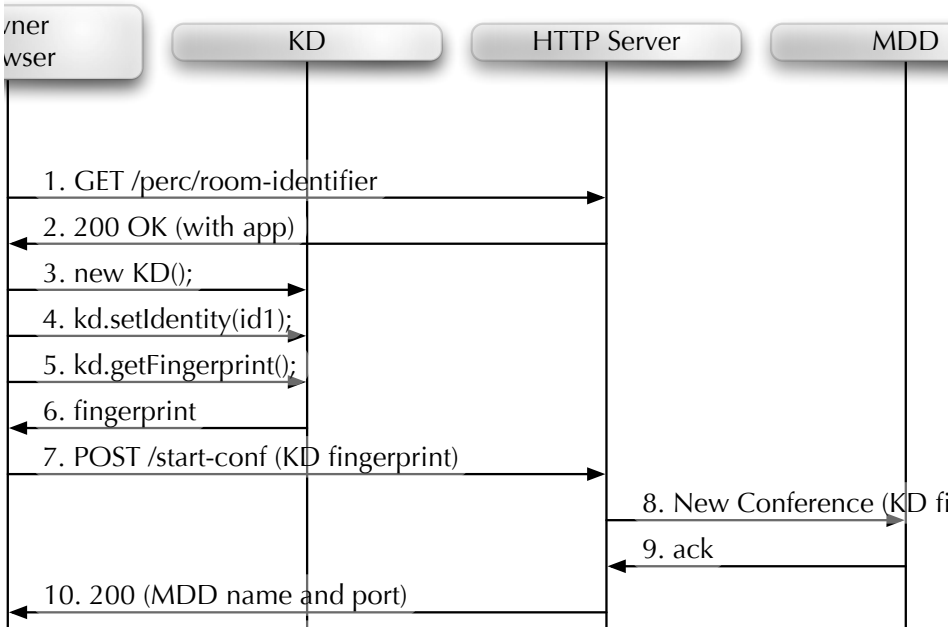
4.1.1 Conference Establishment

3. The conferencing web application instantiates a new Key Distributor (KD) object. Upon instantiation, this KD creates a new certificate.
4. The conferencing web app sets the owner's identity on the KD.
5. The application asks for the certificate's fingerprint…
6. …which the KD provides



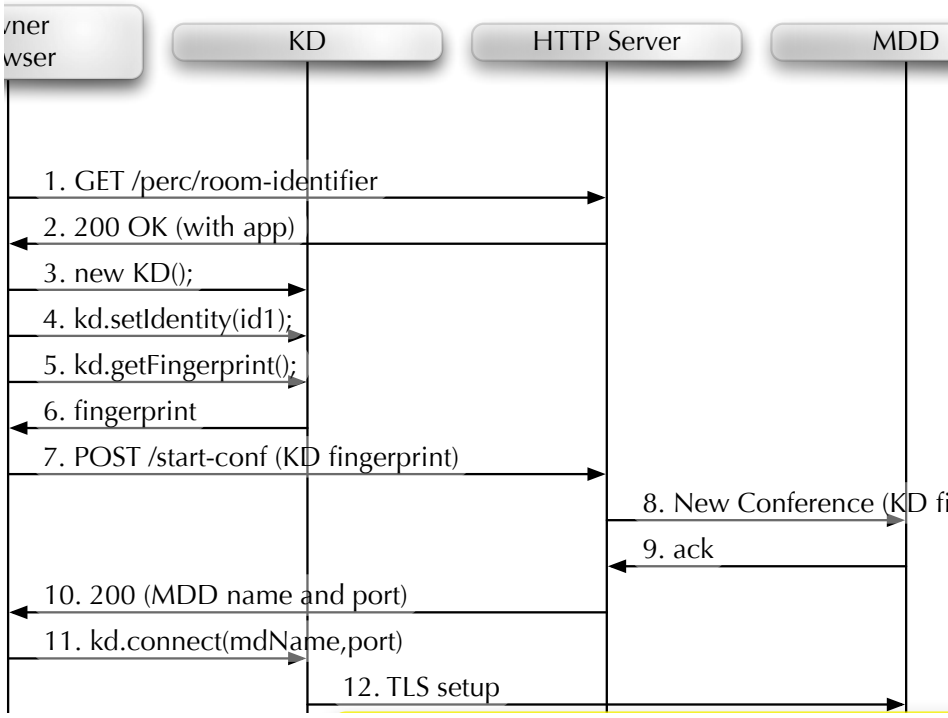
4.1.1 Conference Establishment

7. The application initiates the conference, including the KD cert fingerprint as part of the initiation message.
8. The HTTP server passes the KD fingerprint along to the Media Distributor (MD). This is used later by the MD to ensure that the correct KD is connecting to it.
9. The MD acknowledges creation of a new conference. This acknowledgement contains the hostname and port that the MD is listening on for the KD to connect to.
10. The web server responds to the conferencing web app with the hostname of the MD as well as the port it is listening on for the KD to connect.



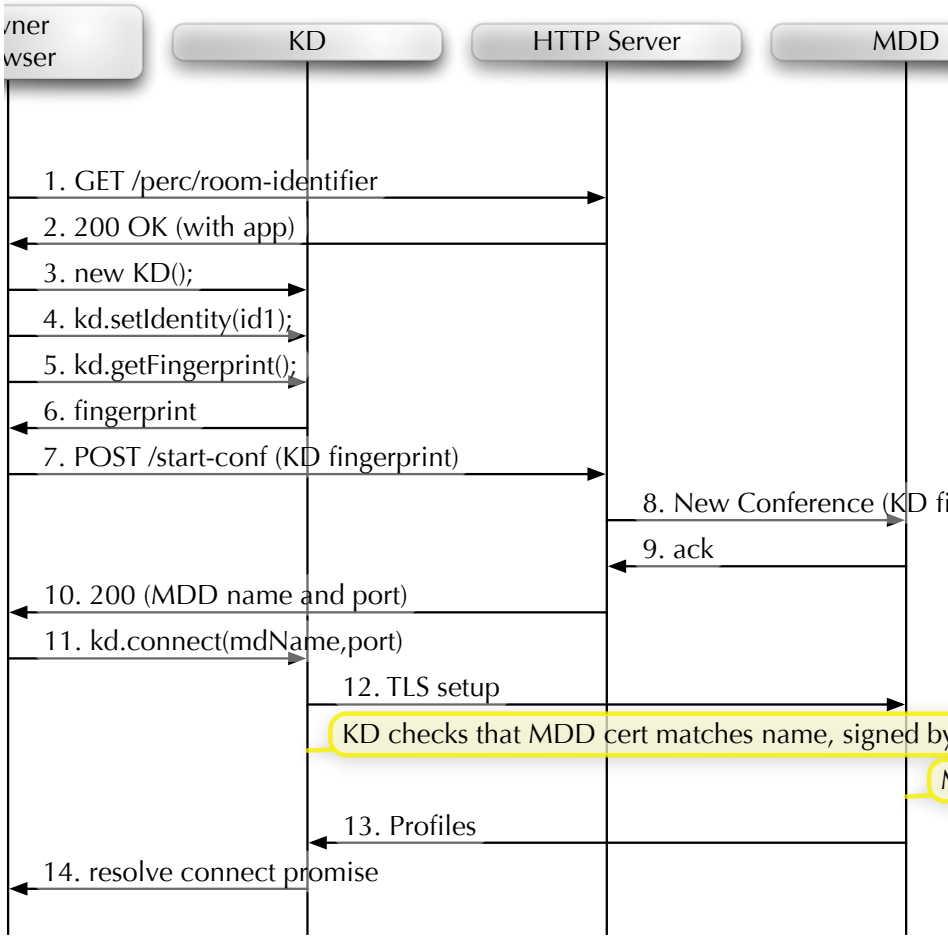
4.1.1 Conference Establishment

11. The application passes along the MD name and port to the KD object.
12. The KD initiates a TLS connection to the MD. This connection uses the protocol defined in [I-D.ietf-perc-dtls-tunnel]. The KD verifies that the certificate presented by the MD matches the name it used to connect to it, and that it chains up to a trusted certificate authority. The MD verifies that the client cert provided by the KD matches the fingerprint that it received earlier from the HTTP server.

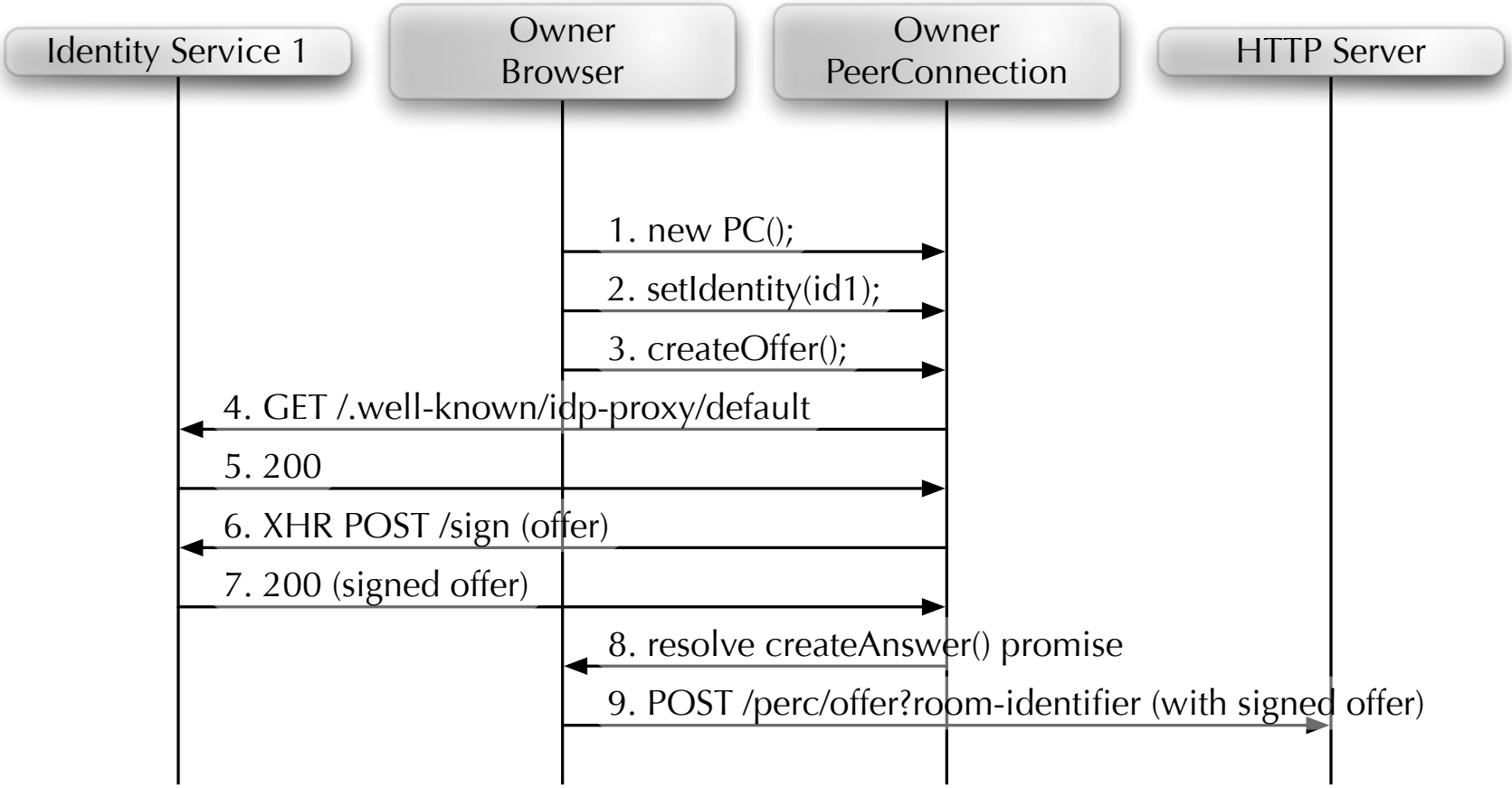


4.1.1 Conference Establishment

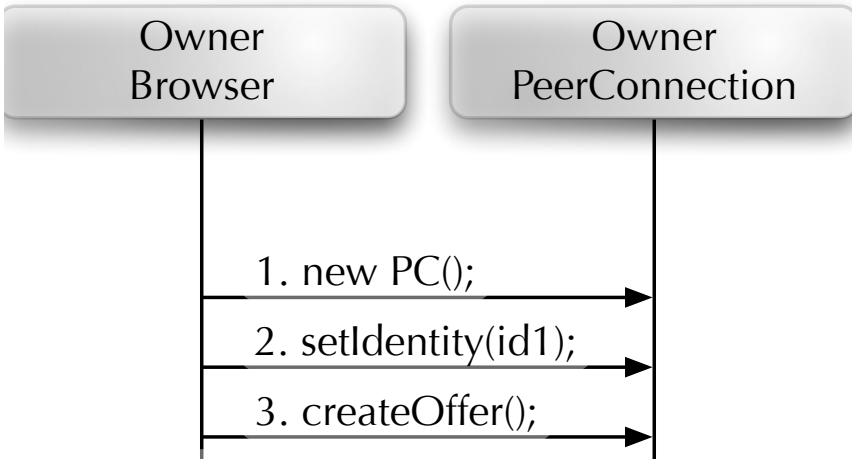
13. The MD returns its list of supported profiles, as described in [I-D.ietf-perc-dtls-tunnel].
14. The KD object indicates to the conferencing app that the KD-MD connection has been successfully established.



4.1.2 Owner Sends Offer to Conference

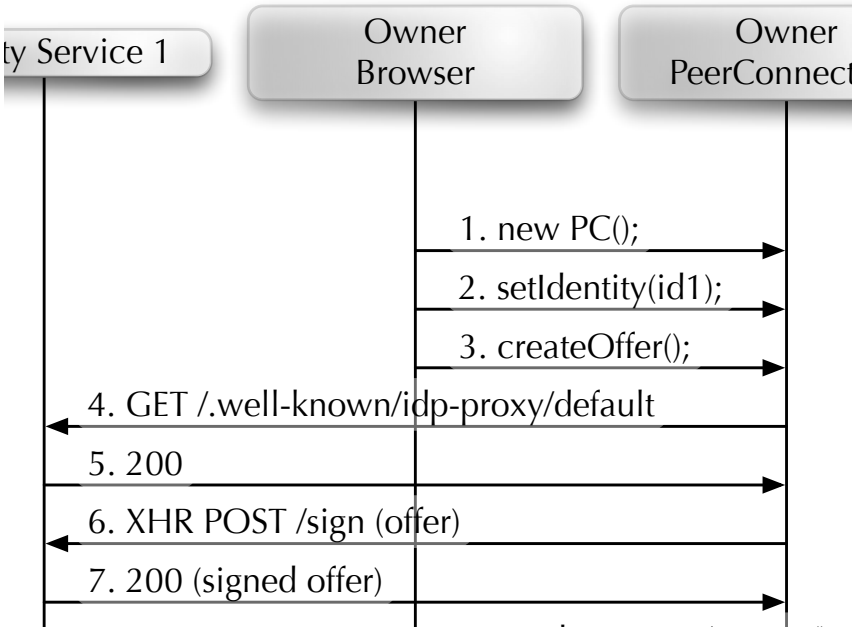


4.1.2 Owner Sends Offer to Conference



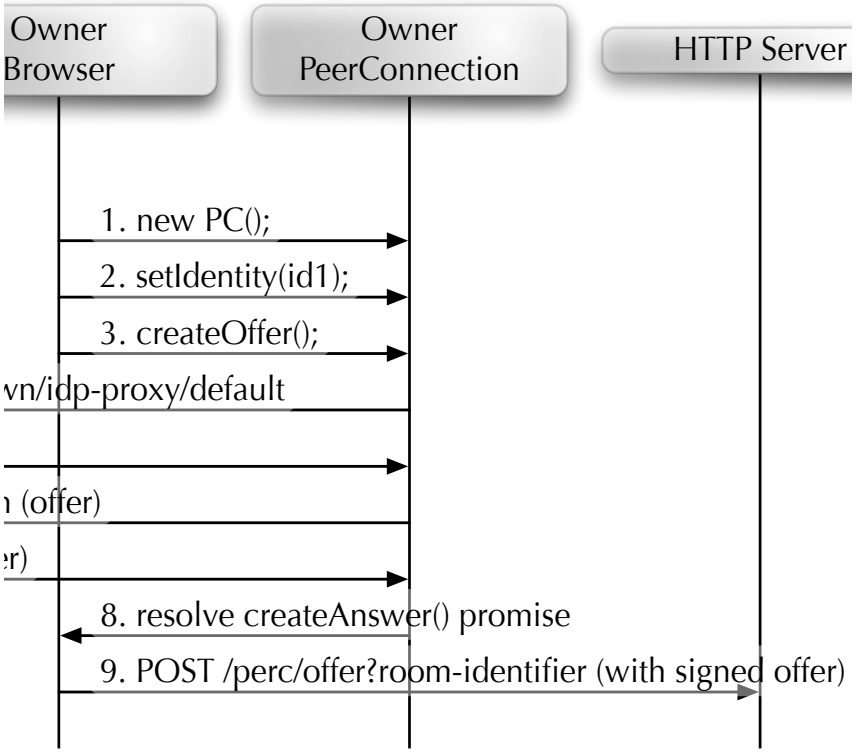
1. The conferencing web application creates a new WebRTC RTCPeerConnection (PC) object to allow sending and receiving media.
2. The conferencing web app sets the owner's identity on the PC…
3. …and requests an offer to be created.

4.1.2 Owner Sends Offer to Conference



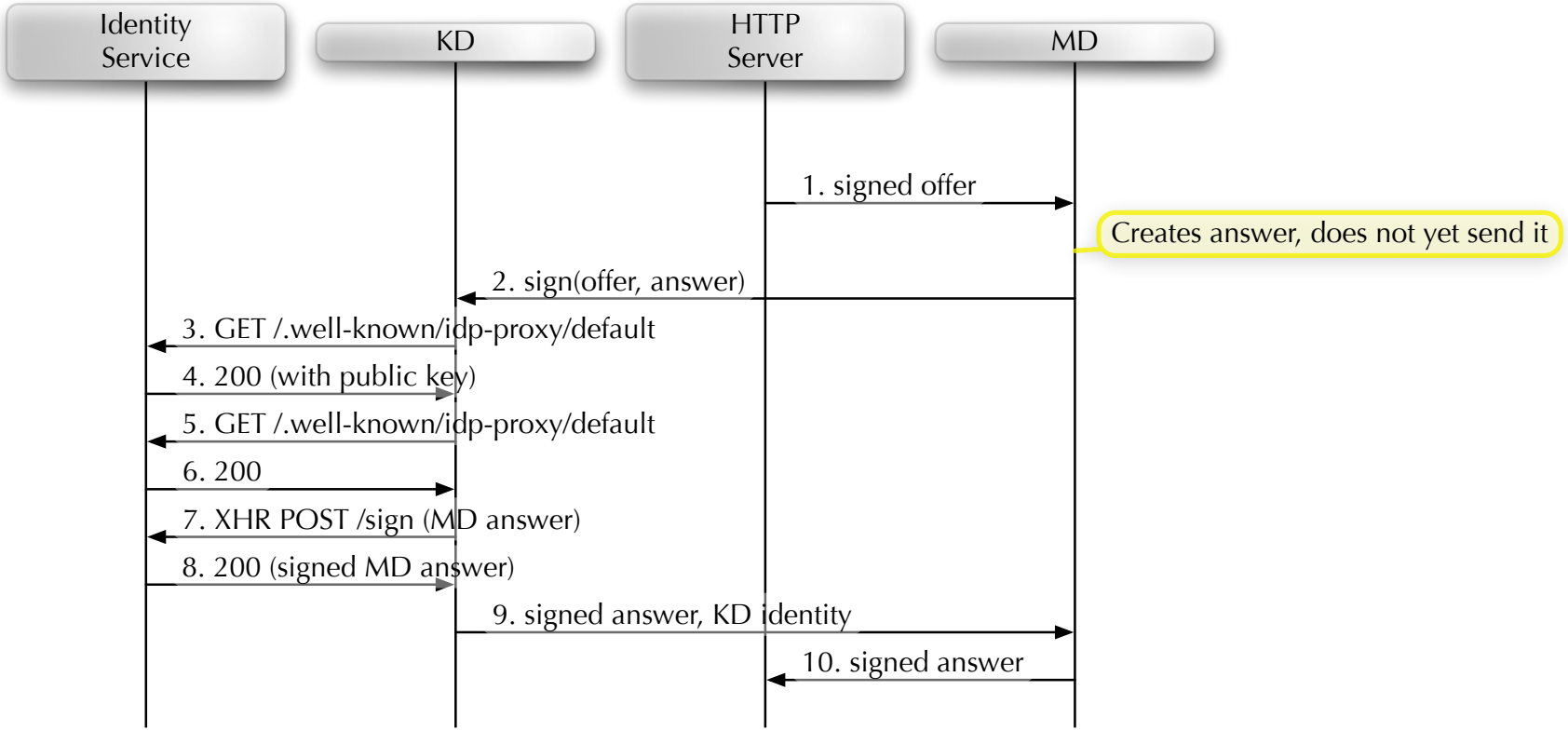
4. As described in [I-D.ietf-rtcweb-security-arch], the PC requests the IDP proxy code from the Identity Service…
5. …which it returns.
6. Upon being executed, the idp-proxy code sends the offer to the Identity service…
7. …which verifies user's identity, the signs the offer, and returns the signed offer to the PC.

4.1.2 Owner Sends Offer to Conference

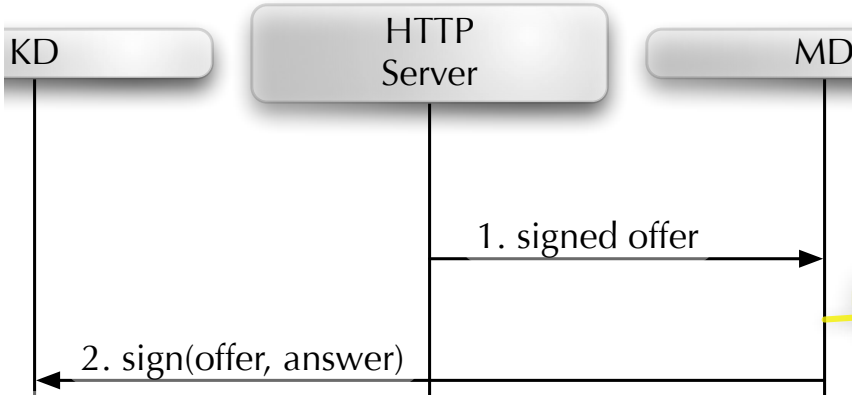


8. The PC then returns the signed offer to the conferencing web app.
9. The conferencing web app sends the signed offer to the HTTP server to begin establishing the media connection.

4.1.3 Conference Processes Owner Offer

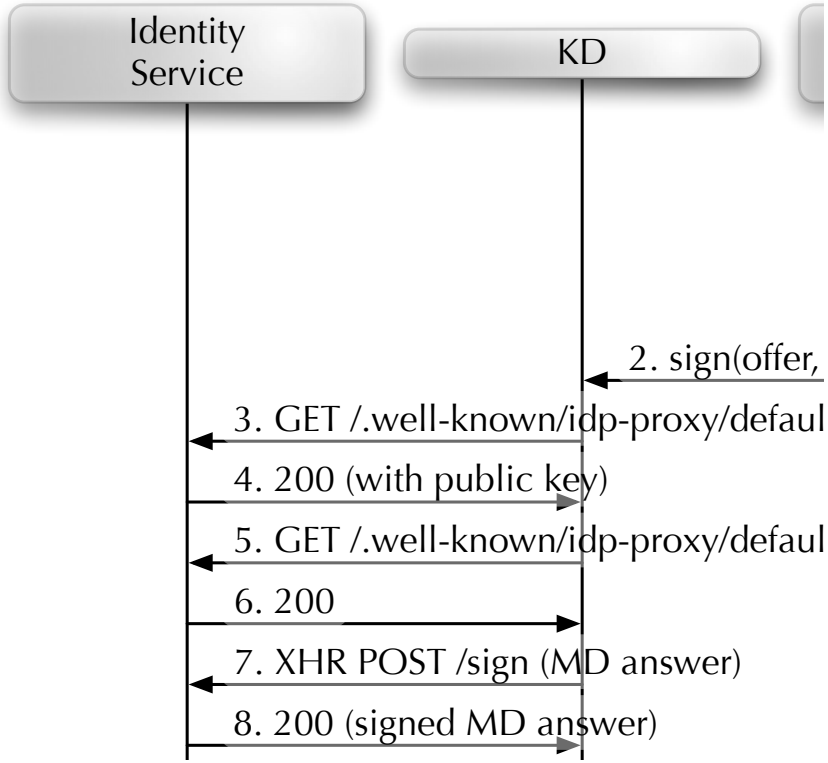


4.1.3 Conference Processes Owner Offer



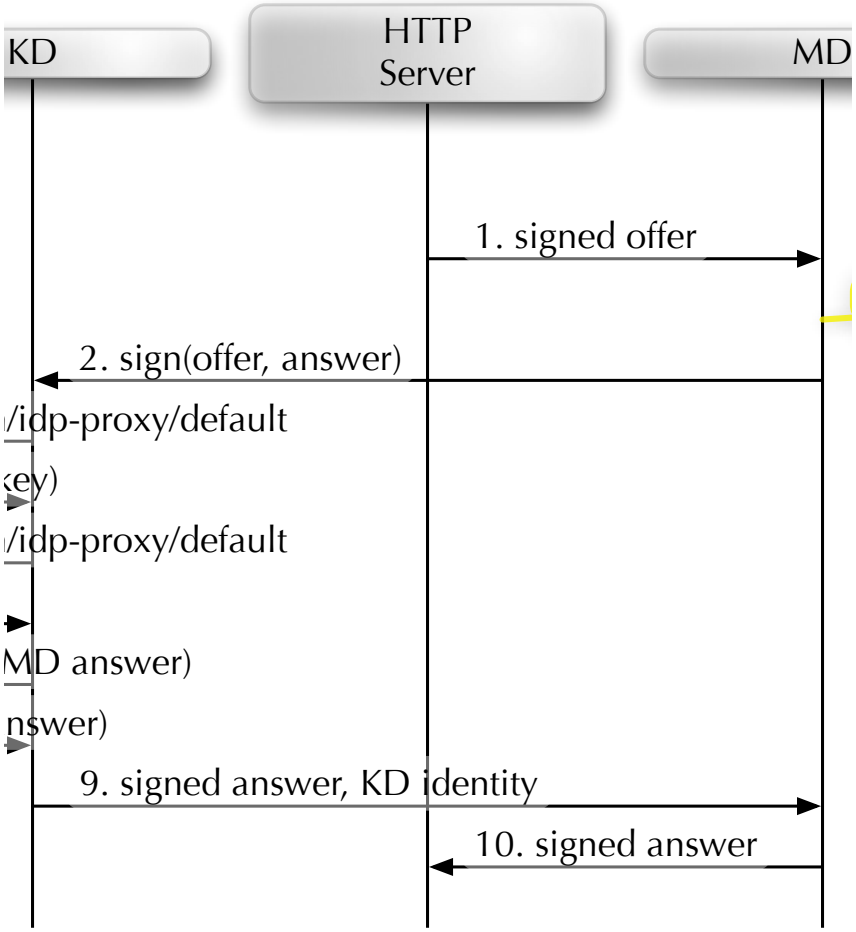
1. The HTTP server sends the signed offer to the MD to allow it to start setting up the media connection.
2. The MD creates an answer to the received offer, and sends both offer and answer over the MD-KD tunnel connection.

4.1.3 Conference Processes Owner Offer



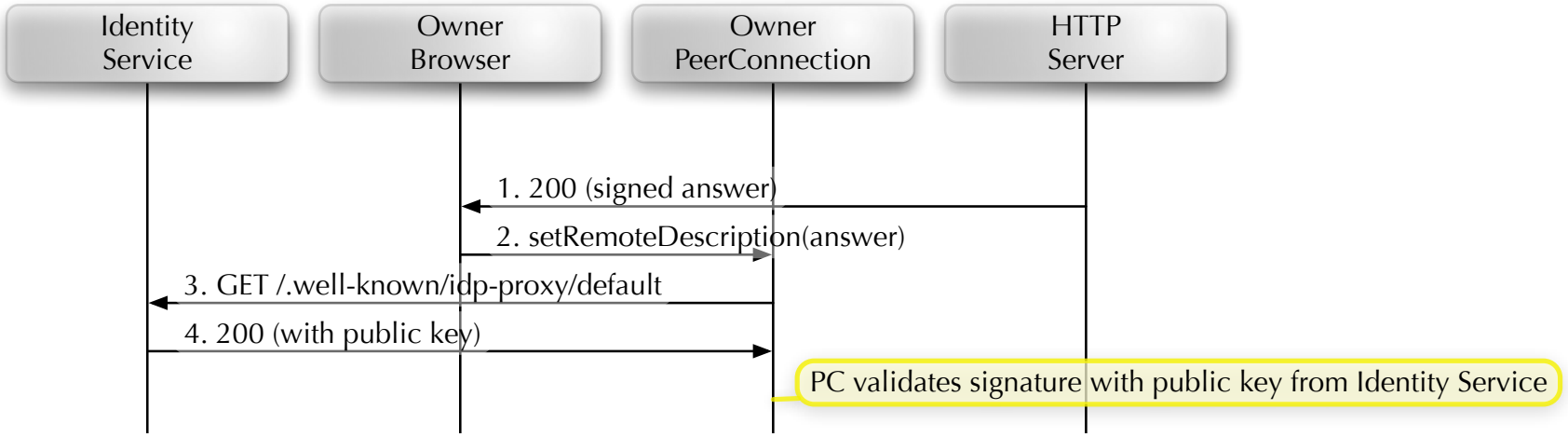
3. Similar to the PC validation procedure described in [I-D.ietf-rtcweb-security-arch], the KD requests the IDP proxy code from the Identity Service based on the identity in the offer…
4. …which it returns. The KD uses the IDP proxy code to validate the identity of the party that generated the offer.
5. Similar to the PC signing procedure described in [I-D.ietf-rtcweb-security-arch], the KD requests the IDP proxy code from the Identity Service…
6. …which it returns.
7. Upon being executed, the idp-proxy code sends the MD answer to the Identity Service…
8. …which verifies KD's identity, the signs the MD answer, and returns the signed MD answer to the KD.

4.1.3 Conference Processes Owner Offer



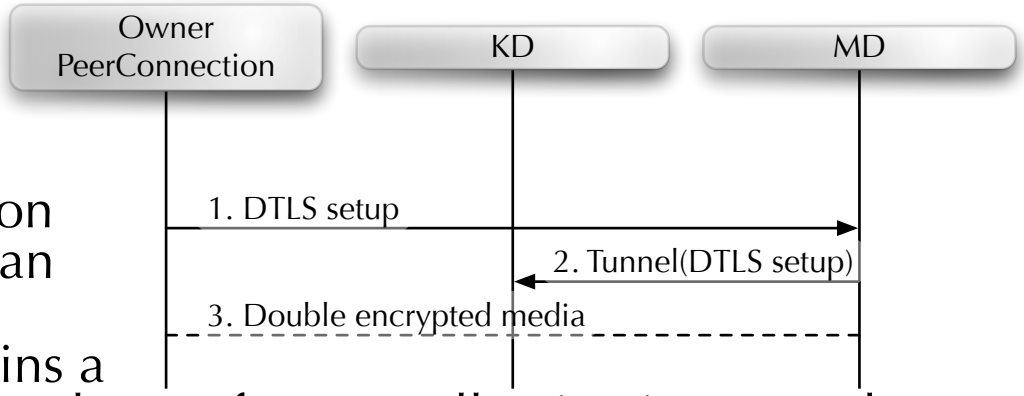
9. The KD sends the signed MD answer back to the MD.
10. The MD sends the signed answer to the HTTP server.

4.1.4 Owner Processes Answer



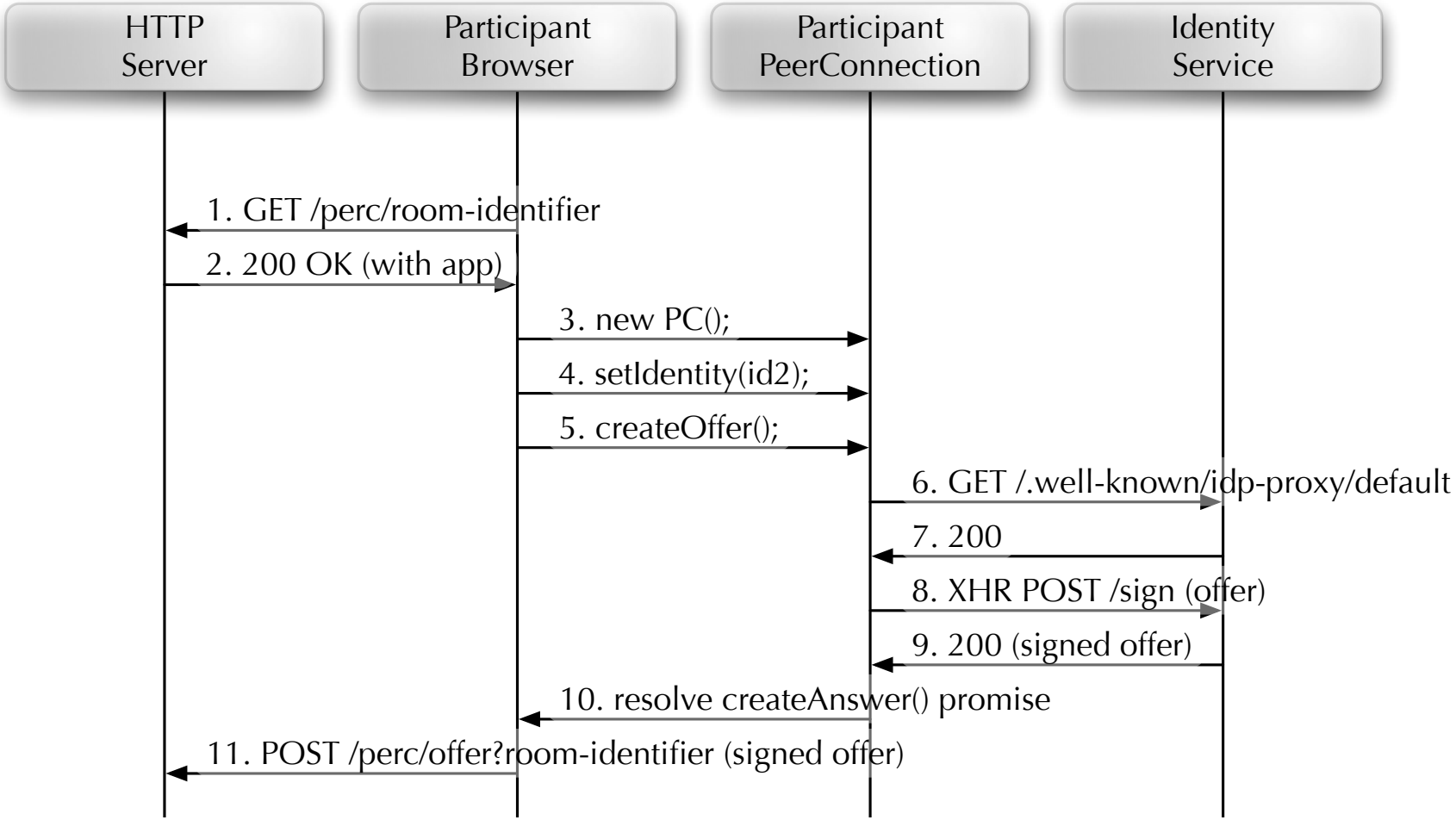
1. To complete the offer/answer exchange, the HTTP server returns the signed answer (asserting the KD's identity) to the owner's conference web app.
2. The conference web app sets the remote description on the PC to the received answer
3. Using the identity validation procedure described in [I-D.ietf-rtcweb-security-arch], the PC requests the IDP proxy code from the Identity Service based on the identity in the answer…
4. …which it returns. The PC uses the IDP proxy code to validate the identity of the party that generated the answer.

4.1.5 Owner sets up media connection

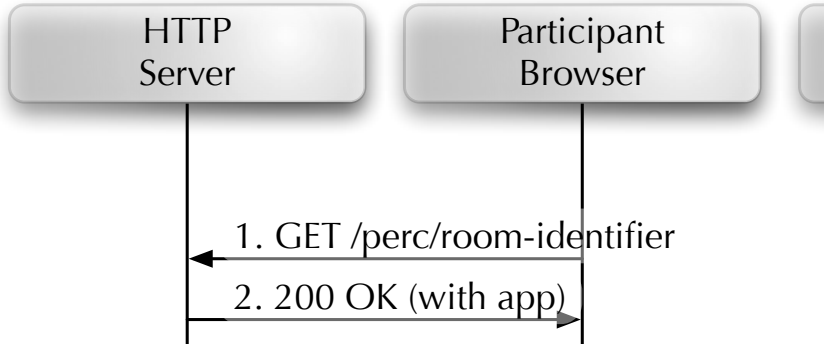


1. The PC, using the addressing information present in the answer, negotiates a DTLS association towards the MD. We make an assumption that the SDP generated by the MD contains a port number that is unique to the conference, allowing it to correlate the incoming DTLS messages to the correct KD.
2. The MD uses the tunneling protocol defined in [I-D.ietf-perc-dtls-tunnel] to forward the DTLS setup messages between the PC and the KD. These DTLS setup messages make use of the mechanism described in [I-D.ietf-perc-srtp-ekt-diet] to establish end-to-end keys for the media.
3. Using the mechanism described in [I-D.ietf-perc-double], the PC now begins to send and received SRTP-encrypted media to and from the MD.

4.1.6 Participant Joins Conference

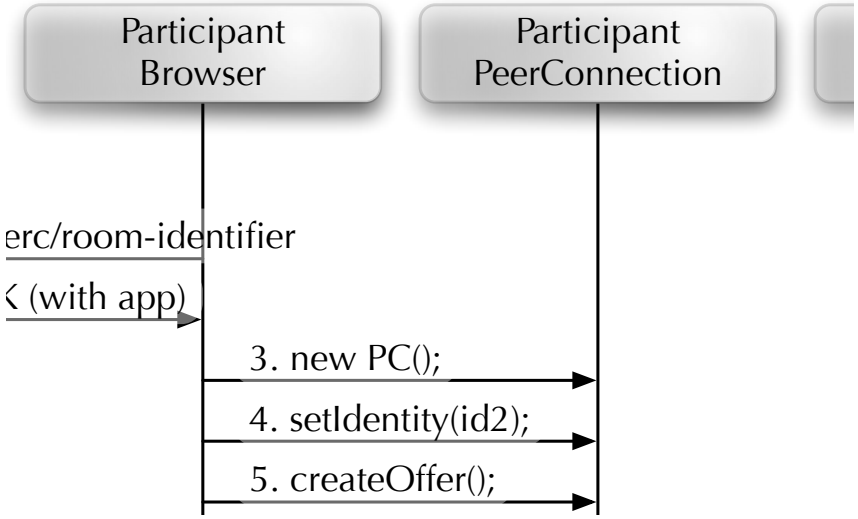


4.1.6 Participant Joins Conference



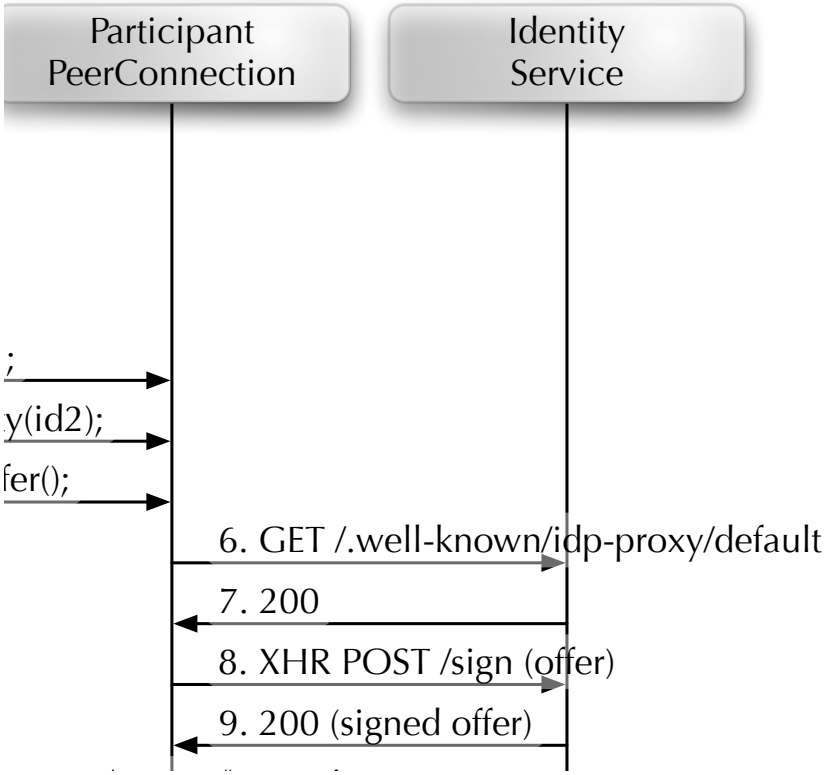
1. The conference owner loads the conference application. Although not required, information sufficient to identify the conference is frequently sent as part of the URL.
2. The HTTP server returns the conferencing web application.

4.1.6 Participant Joins Conference



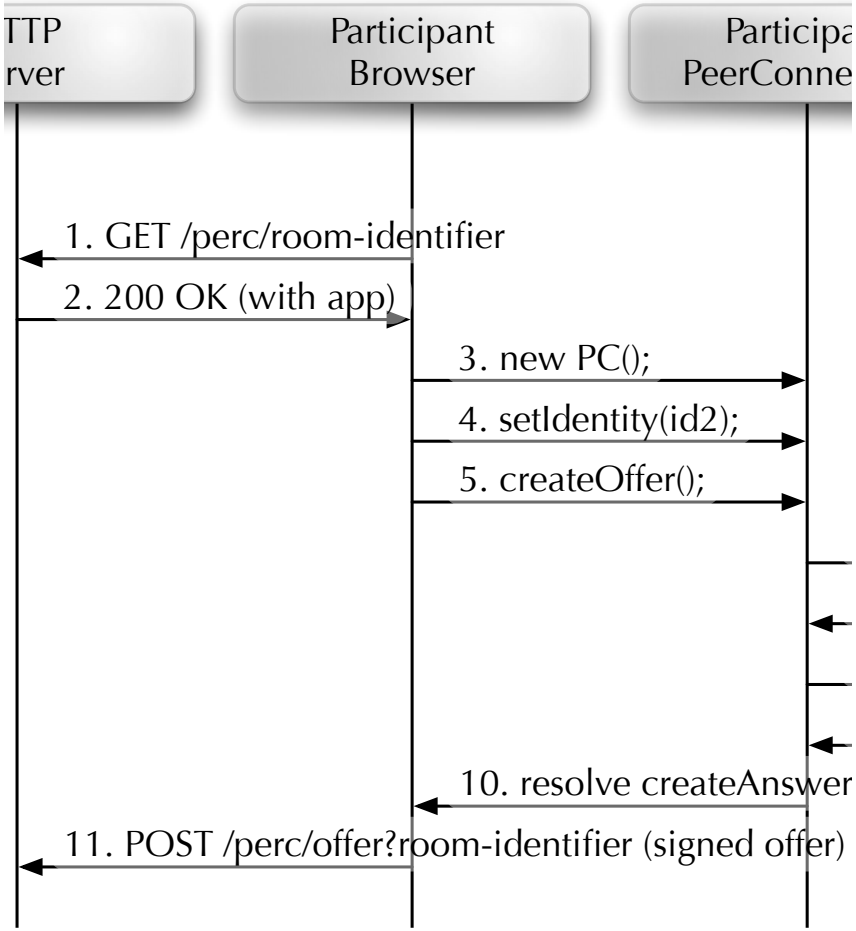
3. The conferencing web application creates a new WebRTC RTCPeerConnection (PC) object to allow sending and receiving media.
4. The conferencing web app sets the owner's identity on the PC…
5. …and requests an offer to be created.

4.1.6 Participant Joins Conference



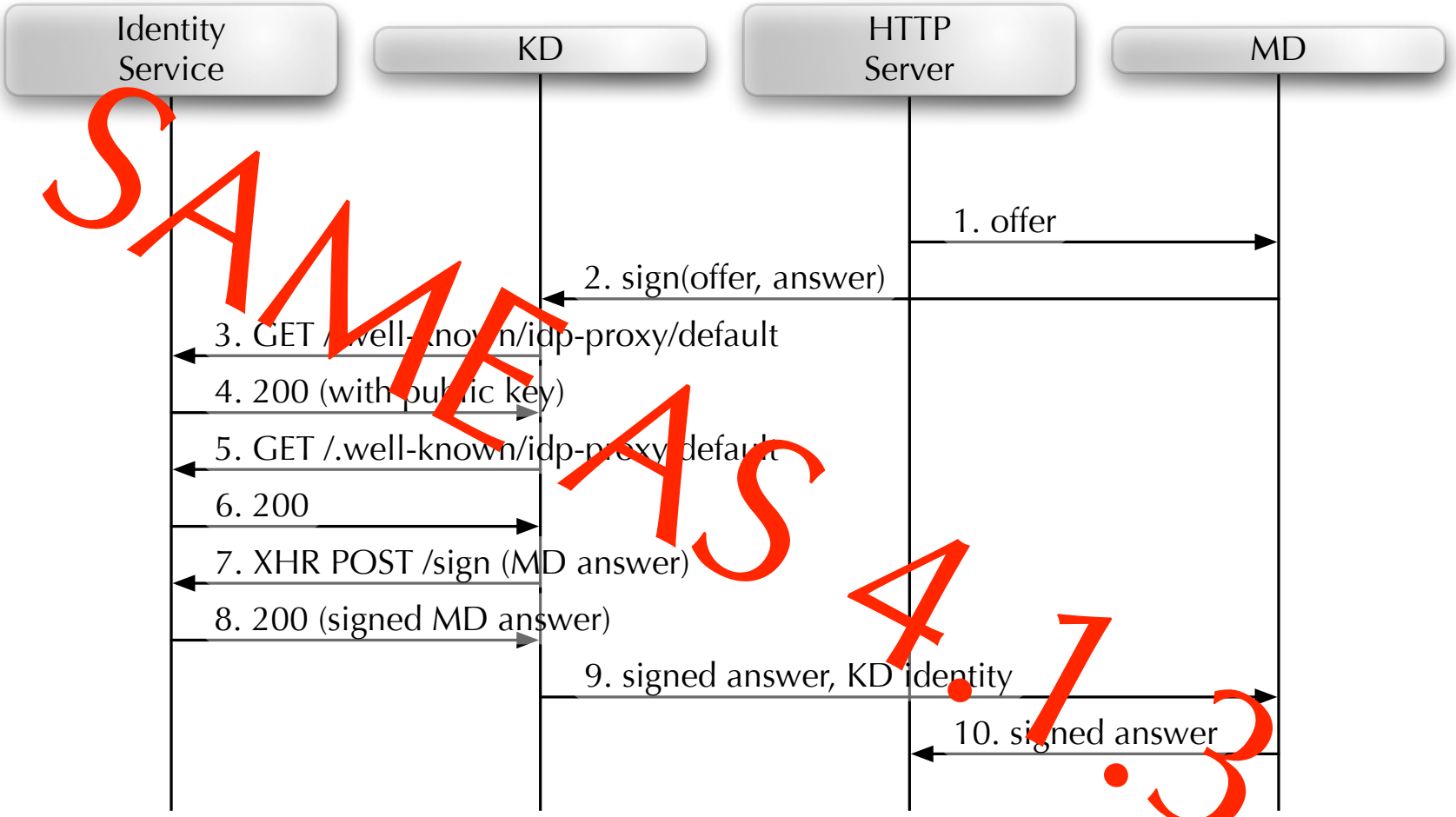
6. As described in [I-D.ietf-rtcweb-security-arch], the PC requests the IDP proxy code from the Identity Service…
7. …which it returns.
8. Upon being executed, the idp-proxy code sends the offer to the Identity service…
9. …which verifies user's identity, the signs the offer, and returns the signed offer to the PC.

4.1.6 Participant Joins Conference

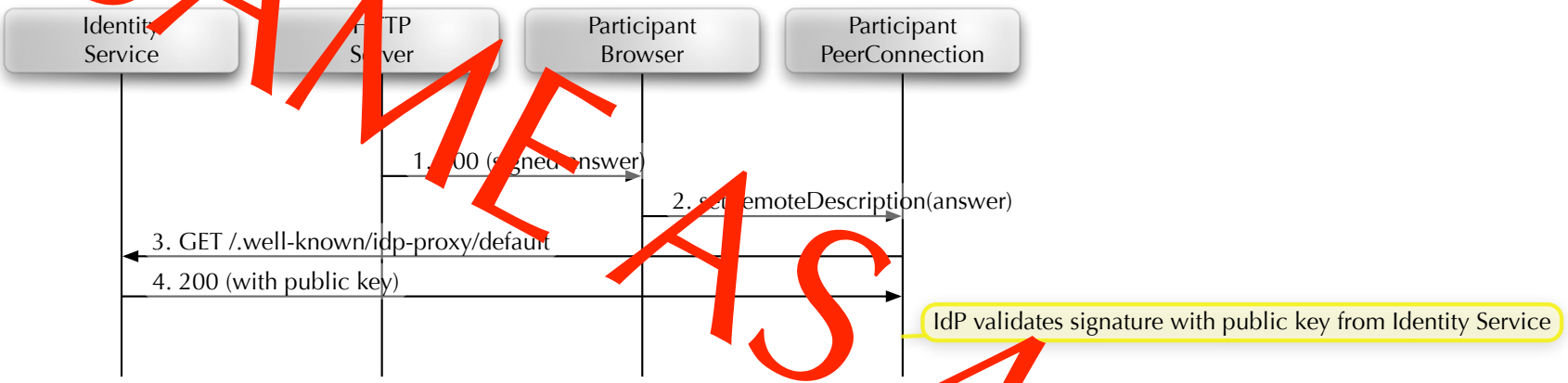


10. The PC then returns the signed offer to the conferencing web app.
11. The conferencing web app sends the signed offer to the HTTP server to begin establishing the media connection.

4.1.7 Conference Processes Participant Offer

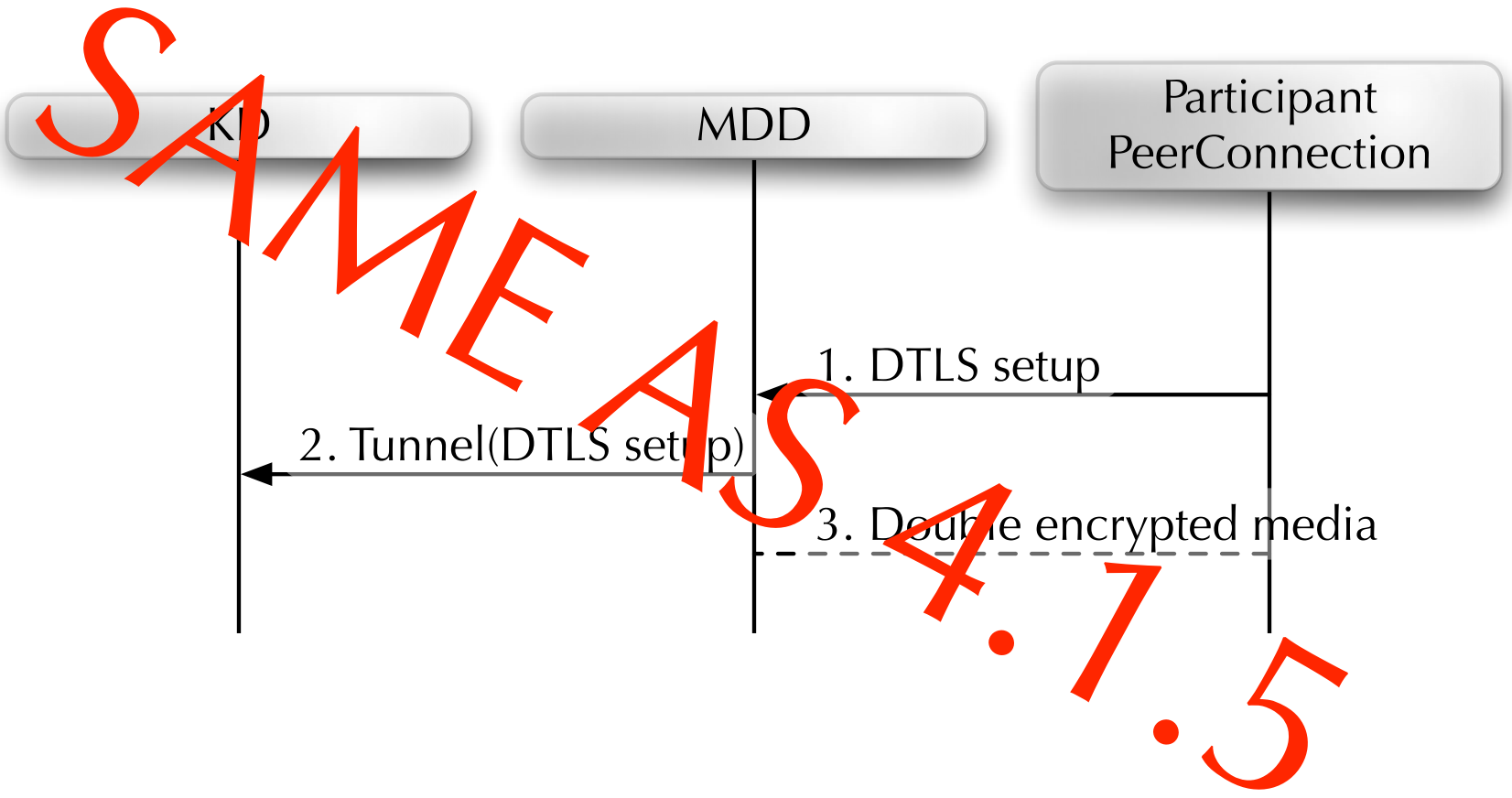


4.1.8 Participant Processes Answer



SAME AS 4.1.4

4.1.9 Participant sets up media connection



NEW STUFF WE NEED

New DOM API (not for this WG)

```
interface RTCKeyDistributor : EventTarget {
    void setIdentityProvider(DOMString provider,
                             optional RTCIdentityProviderOptions options);

    Promise<DOMString> getIdentityAssertion();
    readonly attribute Promise<RTCIdentityAssertion> peerIdentity;
    readonly attribute DOMString?                    idpLoginUrl;
    readonly attribute DOMString?                    idpErrorInfo;

    Promise<RTCCertificate> getCertificate();

    Promise<void> connect(DOMString mdHost, unsigned short mdPort);

    attribute EventHandler              onfingerprintfailure;
}
```

New Tunnel Messages


```
struct {
  opaque offer<1..2^24-1>;
  opaque answer<1..2^24-1>;
} SignAnswer;
```

```
struct {
  opaque answer<1..2^24-1>;
} SignAnswerAck;
```

Next Steps

- Sanity checking approach, information flow
- Is this all we need?
  - I don't think we need anything special in the SDP – support for PERC can be inferred from double crypto suites.
- Probably need a co-author to help, if we plan to progress this work