TLS 1.3 Client Authentication

Andrei Popov, Microsoft Corp.

Issue 1: TLS Client Authentication After the Handshake

- The user navigates to the site's landing page, no client authentication required.
- Eventually, the user requests a protected resource, the site triggers renegotiation with client authentication.
- The site validates client credential and authorizes the request.

TLS 1.3 does not allow renegotiation, therefore sites designed as shown above can't use TLS1.3.

Proposal: Pull Client Authentication out of the Handshake

1. CertificateRequest, client Certificate, CertificateVerify and NewSessionTicket messages use a new content type distinct from "handshake".

2. The client can send Certificate followed by CertificateVerify at any time application data is permitted, regardless of whether the server had previously sent CertificateRequest.

3. The server can send CertificateRequest and NewSessionTicket at any time application data is permitted. Alternatively, the server can encapsulate CertificateRequest in an application protocol message.

4. The server can send NewSessionTicket in response to client CertificateVerify or Finished.

Open Issues:

- We need to decide what CertificateVerify would be signing (e.g. the handshake hash or some form of RFC5705 Exported Keying Material + the client certificate itself).
- Should we try to identify messages belonging to the same transaction, i.e. CertificateRequest or Certificate include a random ID and any messages sent in response (e.g. Certificate, CertificateVerify, NewSessionTicket) echo this ID?
- Do we need a signal in the ServerHello that tells the client to not send any application data until the server's CertificateRequest? What would that mean for 0-RTT?

Issue 2: Client Certificate Selection

- Currently, CertificateRequest message allows the selection of client certificate by:
  - Signature algorithm,
  - Hash algorithm, and
  - Distinguished Names of acceptable (intermediate and/or root) certificate authorities.
- For some sites and EAP-TLS deployments, these selection criteria are insufficient, and result in poor UX where a confusing certificate picker dialog has to be displayed.
- Customers are requesting flexible certificate selection criteria using KU, EKU, Issuance Policy, other OIDs, logical expressions, etc...
- Over time, new extension OIDs are being added to certificates.

Proposal: Add CertificateExtensions field to CertificateRequest for TLS 1.3

certificate_extensions

- A list of certificate extension OIDs [RFC5280] with their allowed values, represented in DER-encoded format.
- If the server has included a non-empty certificate_extensions list, the client end-entity certificate MUST contain all of the specified extension OIDs that the client recognizes.
- For each extension OID recognized by the client, all of the specified values MUST be present in the client certificate (but the certificate MAY have other values as well).
- The client MUST ignore and skip any unrecognized certificate extension OIDs.
- If the client supplies a certificate that does not satisfy the request, the server MAY respond with a fatal unsupported_certificate alert.
- TLS implementations should rely on their PKI libraries to perform certificate selection using certificate extension OIDs.

Proposed CertificateRequest for TLS 1.3

```
struct {
    opaque certificate_extension_oid<1..2^8-1>;
    opaque certificate_extension_values<0..2^16-1>;
} CertificateExtension

struct {
    ClientCertificateType certificate_types<1..2^8-1>;
    SignatureAndHashAlgorithm
       supported_signature_algorithms<2..2^16-2>;
    DistinguishedName certificate_authorities<0..2^16-1>;
    CertificateExtension certificate_extensions<0..2^16-1>;
} CertificateRequest;
```