Designing a ticket

Resumption ticket layout and encryption is left to the server to decide. The server essentially encrypts the session status and passes it to the client for safe keeping. When the client wants to resume it simply sends the ticket back to the server, who decrypts it. From a client point-of-view its seen as a ticket that allows rapid re-admission to a session.

The most commonly used mechanism is described in RFC5077. But this document is somewhat outdated, and assumes an earlier version of TLS. So we can improve on it. It is just an advisory – it is not mandated for TLS1.3. If we can make it smaller by eliminating some redundant stuff – why not.

Encrypting the key

The first thing to decide upon is the method of encryption, and the provenance/maintenance of the key and IV that the server uses to encrypt/decrypt the ticket.

One thing working to our advantage is that we really don't care too much if the resumption mechanism fails. Its not the end of the world. We can always do a new full connection. It is just a nice performance bonus when resumption does work. So for example if the server crashes (a rare event) its not a problem that all outstanding tickets are immediately invalidated (as the key to decrypt them has probably been lost). But on the other hand a connection based on an external PSK would be expected to work every time.

I recommend using AES-GCM-128 for ticket encryption (which improves on the old school AES-CBC and HMAC approach which required two keys, as described in RFC5077). The server simply generates a random AES-128 TEK (Ticket Encryption Key) at start up, and uses it to encrypt every ticket issued to every client. For an IV use a 96-bit ticket number generated randomly at start up which increments after each ticket, and include this ticket number as unencrypted but authenticated data at the start of the GCM ciphertext.

```
struct {
      opaque iv[12];
      opaque encrypted_state<0..2^24-1>;
      opaque mac[16];
} ticket;
```

What's in the ticket?

We would like to keep the ticket size small. However if the client authenticated itself using a certificate chain, the top level certificate (which specifies all of the stuff that the client has successfully laid claim to and proved about itself in the original handshake – name, public key, etc) must be included. Also to be included is the PSK derived from the original connection (or provided externally) the cipher suite in use during the original handshake, the group used previously for the key exchange, and finally a 32-bit timestamp indicating the time of creation of the ticket.

When the server decrypts the ticket, it is in a position to resume the connection with the same state of knowledge about the client that it held during the original connection. If the ticket is considered out-of-date by the server, the resumption attempt fails. This is again a server decision, although the lifetime of the ticket was indicated to the client at the time of its creation.

```
struct {
    uint32 timestamp;
    uint16 cipher_suite;
    uint16 group;
    opaque psk[48];
    ClientIdentity client_identity;
} StatePlaintext;

enum {
   anonymous(0),
   certificate_based(1),
   psk(2)
} ClientAuthenticationType;

struct {
    ClientAuthenticationType client_authentication_type;
    select (ClientAuthenticationType) {
        case anonymous: struct {};
        case certificate_based:
            X509_Certificate<0..2^24-1>;
        case psk:
            opaque psk_identity<0..2^16-1>;
    };
} ClientIdentity;
```

By default the client does not authenticate – in which case the authentication type is anonymous. In which case the encrypted ticket size could be as small as 88 bytes.

One minor issue: I am assuming that in the case of an external PSK that the group for key exchange is also agreed. It makes little sense to agree on a PSK and not agree on a key exchange group, in which case an expensive Handshake Retry Request (HRR) would be needed on each connection.

In the case of a resumption I am reasonably assuming that the same key exchange group will be used that succeeded in the original full handshake, possibly as modified by an HRR.