TLS 1.3

Eric Rescorla

Mozilla

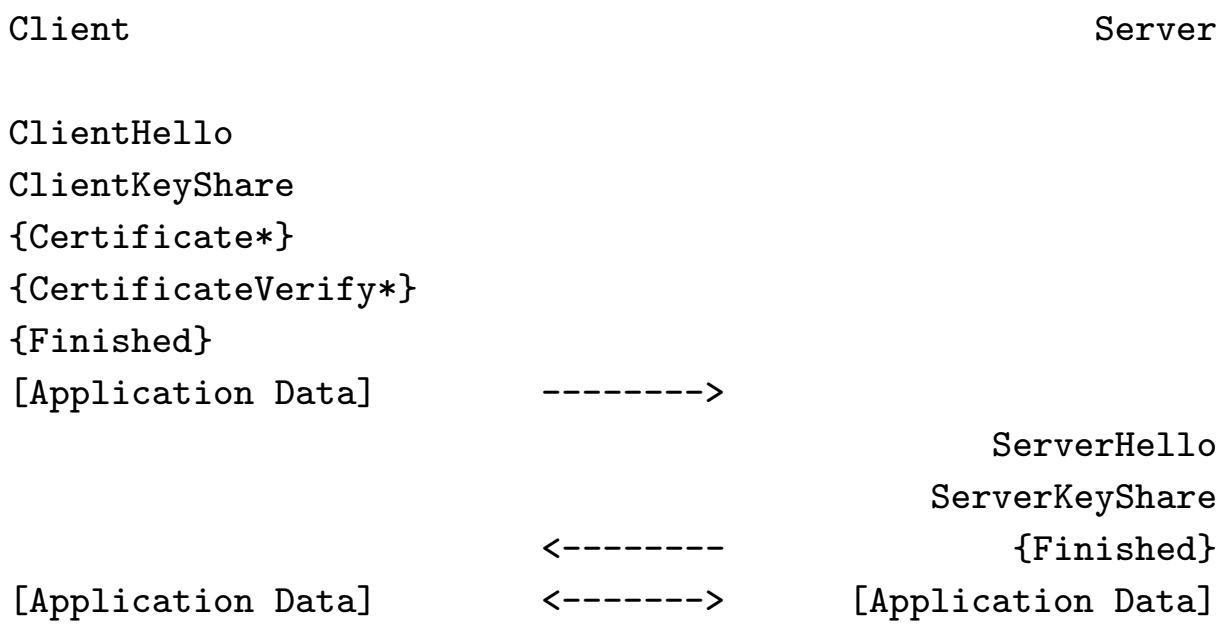
ekr@rtfm.com

Changes since -03

- draft-05
  - Prohibit SSL negotiation for backwards compatibility.
  - Fix which MS is used for exporters.
- draft-04
  - Modify key computations to include session hash.
  - Remove ChangeCipherSpec
  - Renumber the new handshake messages to be somewhat more consistent with existing convention and to remove a duplicate registration.
  - Remove renegotiation.
  - Update format of signatures with context.
  - Remove point format negotiation.

0-RTT

Overview of 0-RTT Flow



Key Agreement (well-understood)

- Client caches server's long-term (EC)DH share
- First flight data encrypted under client ephemeral/server static
- Server supplies ephemeral in first flight
- Rest of data encrypted under both ephemeral/static and ephemeral/ephemeral

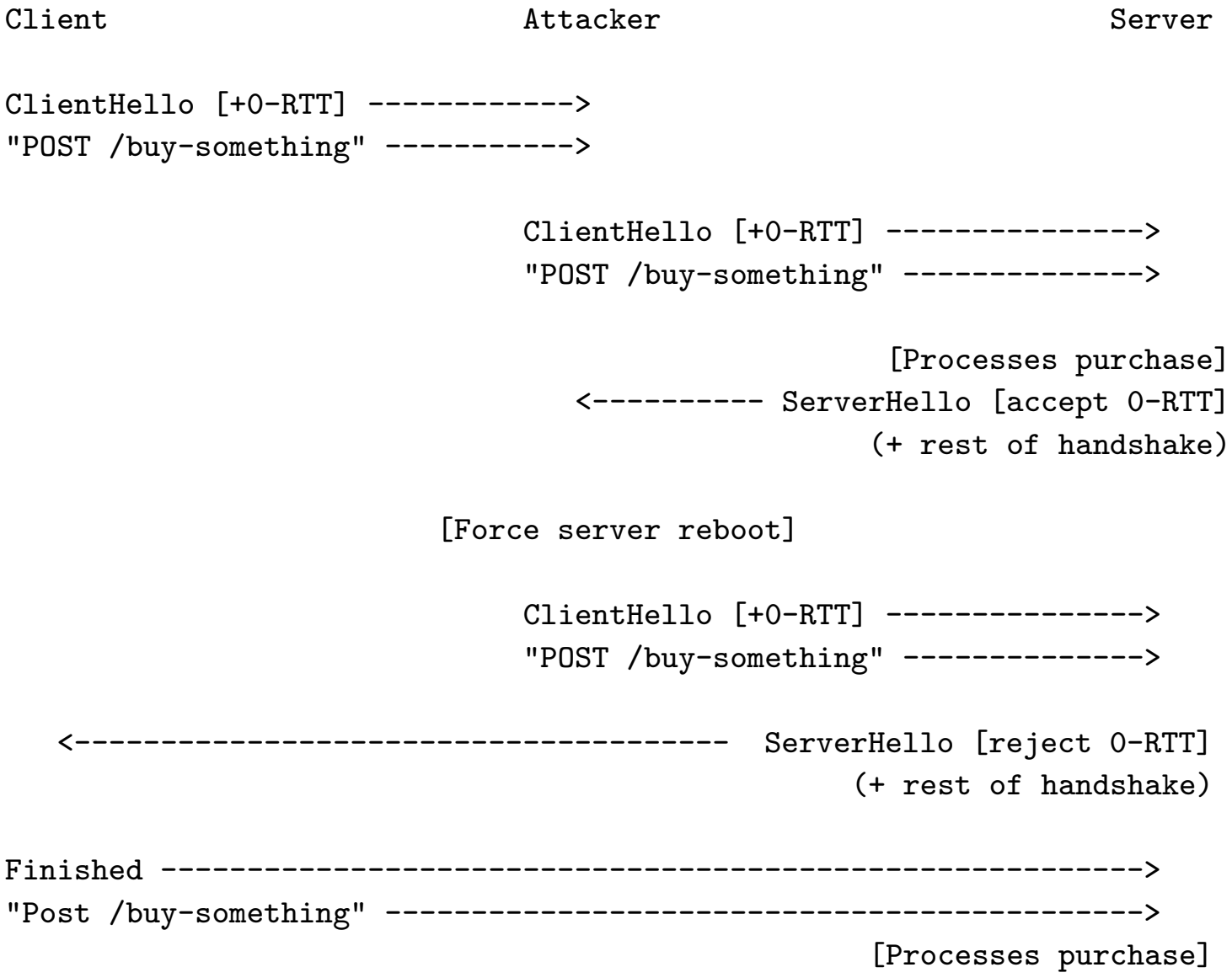
Anti-Replay

- TLS anti-replay is based on each side providing random value
  - Mixed into the keying material
- Not compatible with 0-RTT
  - Client has anti-replay (since they speak first)
  - Server's random isn't incorporated into client's first flight

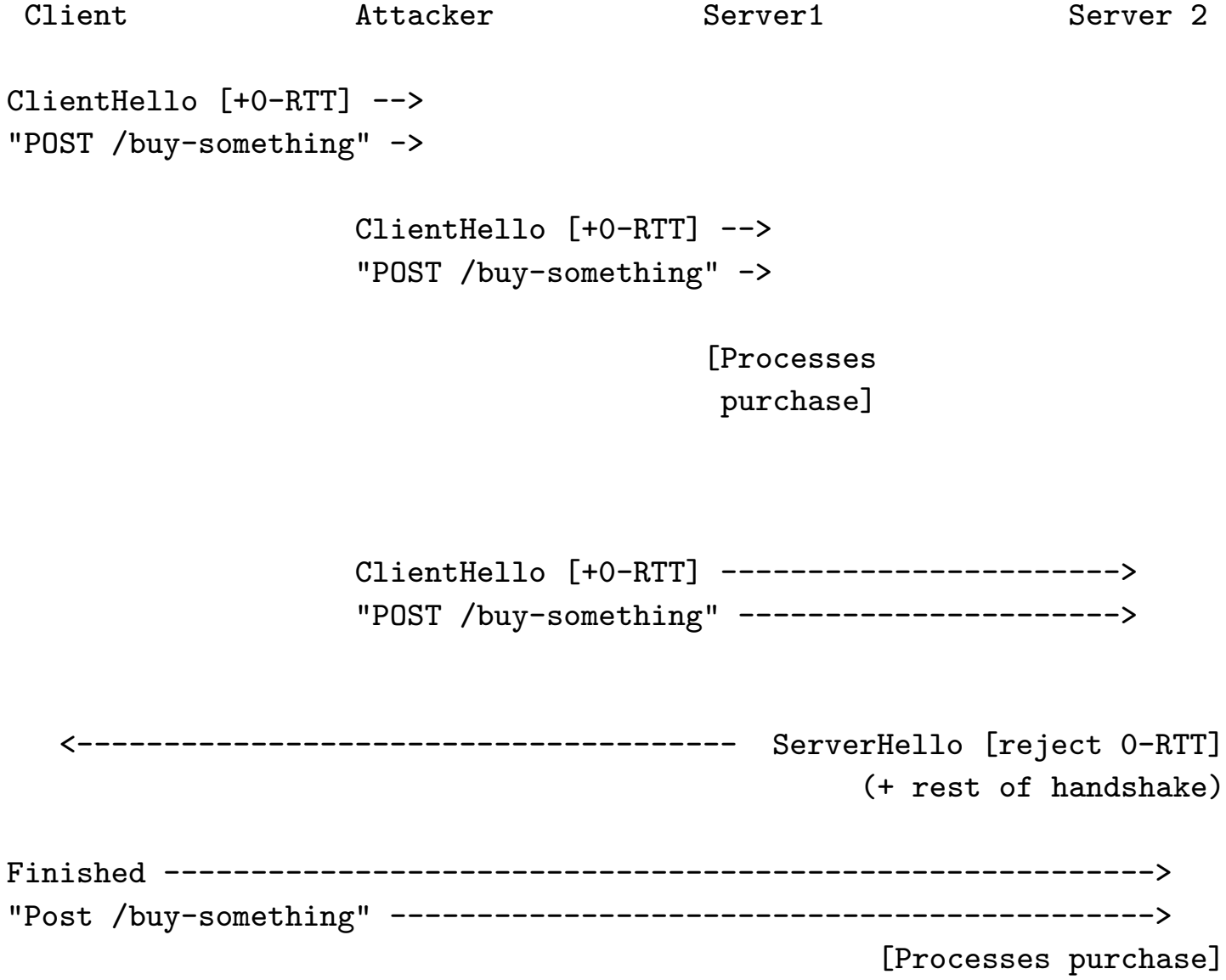
Anti-Replay (borrowed from Snap Start)

- Server needs to keep a list of client nonces
- Indexed by a server-provided context token
- Client provides a timestamp so server can maintain an anti-replay window

This doesn't work



Less Contrived



Options

- Abandon 0-RTT (boo hiss)
- Keep server state globally and temporally consistent
- Remove TLS anti-replay guarantee for the first flight
- Remove TLS reliable delivery guarantee for the first flight

- This is an inherent problem (as far as we know)

Example API (option 3)

```
c = new TLSConnection(...)
c.setReplayable0RTTData("GET /....")
c.connect();
```

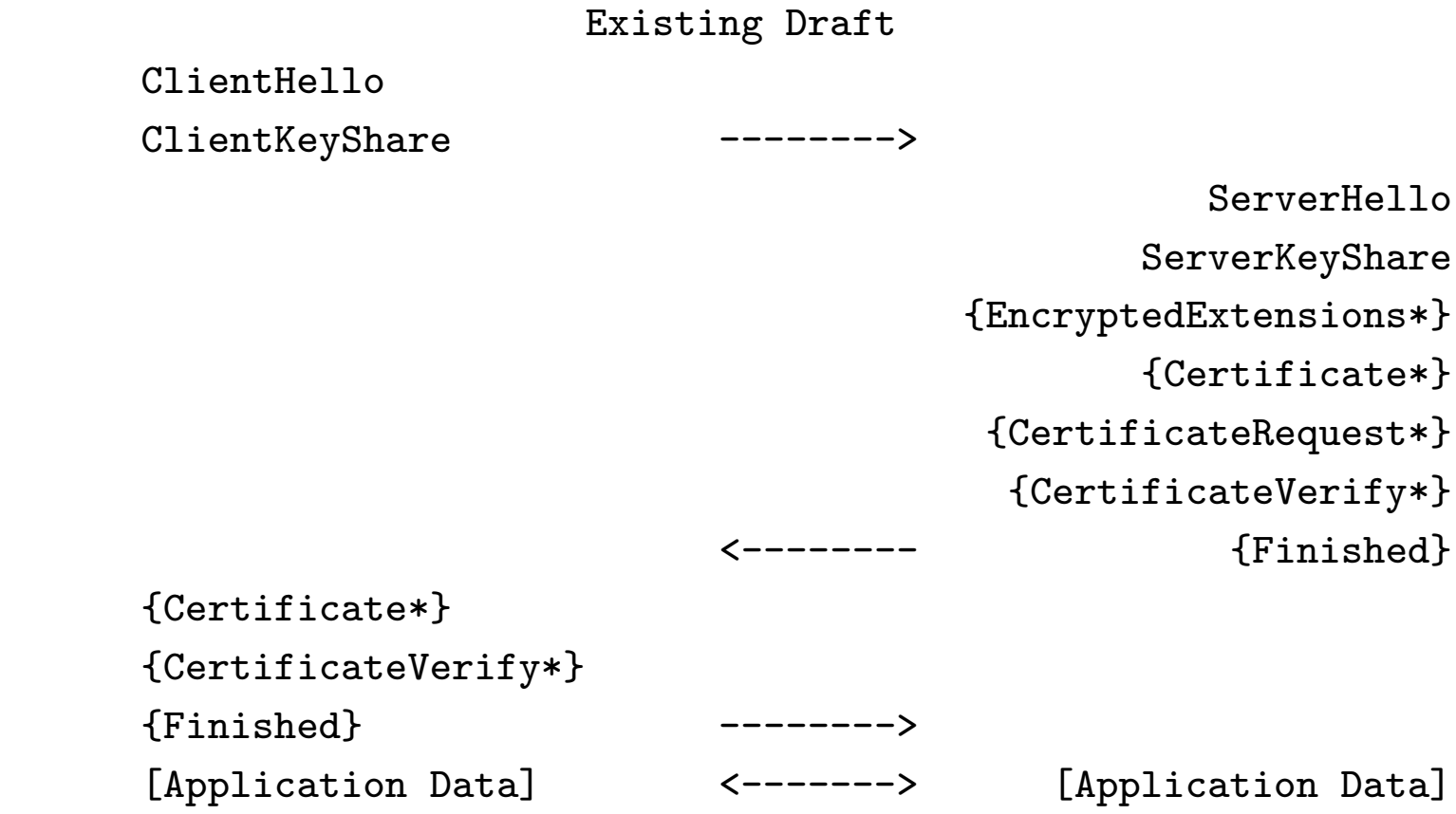
Example API (option 4)

```
c = new TLSConnection(...)
c.setUnreliable0RTTData("GET /....")
c.connect()
if (c.delivered0RTTData()) {
   // Things are cool
} else {
   // Try to figure out whether to replay or not
}
```

DH-Based Handshake

Background Reading

- Git branch:
  `https://github.com/ekr/tls13-spec/tree/WIP_dh_based`
- Text file: `https://github.com/ekr/tls13-spec/blob/ietf92_materials/draft-ietf-tls-tls13-dh-based.txt`
- HTML diff:
  `https://github.com/ekr/tls13-spec/blob/ietf92_materials/draft-ietf-tls-tls13-dh-based-diff.html`



{} Indicates messages protected using keys derived from the handshake master secret.

[] Indicates messages protected using keys derived from the master secret.

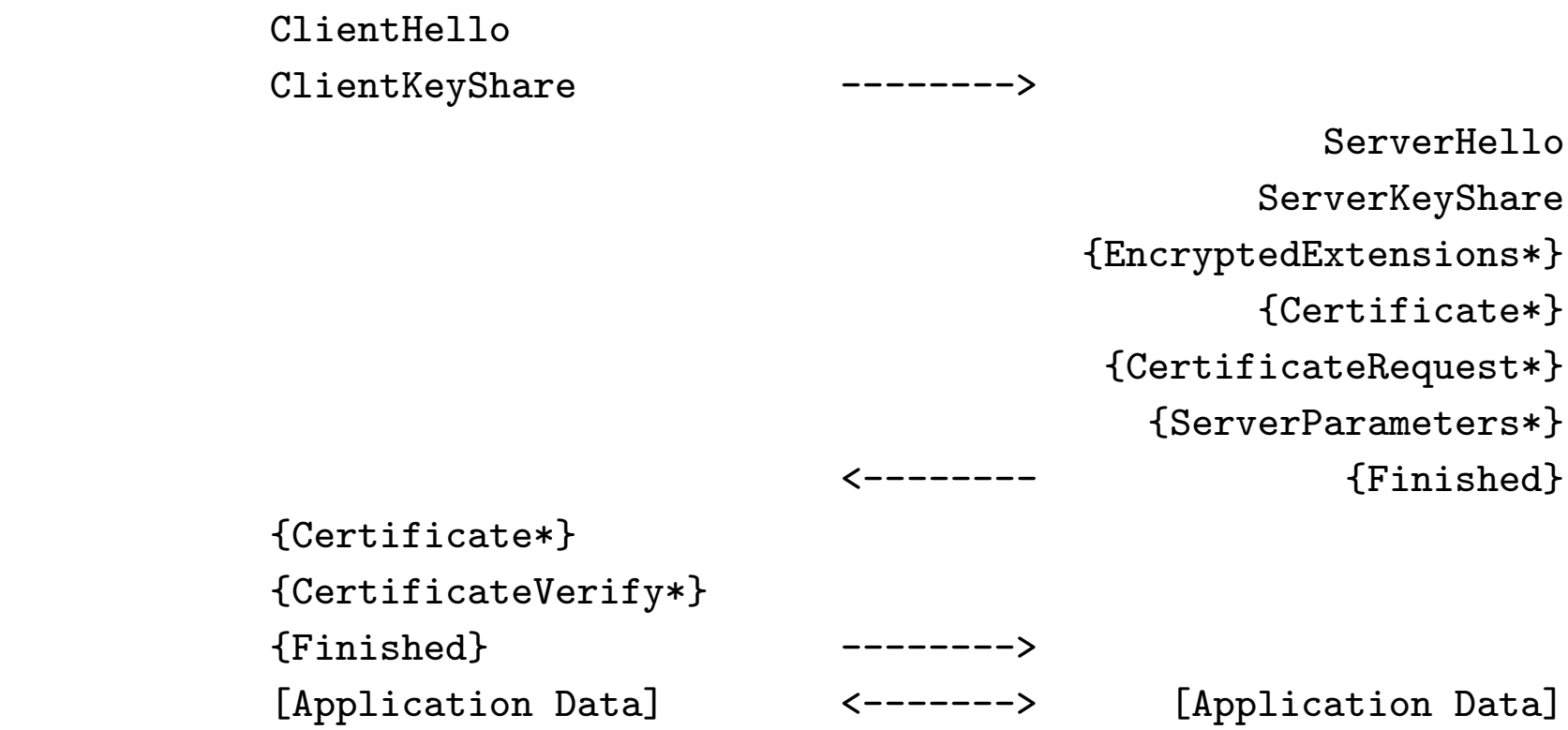
Right now we have two modes

- Signature-based (naive 1-RTT)
- Static DH-based (cached server 1-RTT, 0-RTT)
- Krawczyk/Wee propose just doing static DH

Basic idea

- Server has a semi-static DH key (just like 1-RTT)
- Probably really has long-term signing key
  - Used to sign the semi-static key
  - .. offline versus online signing
- Key exchange computations the same for 0-RTT and 1-RTT (and similar with PSK)

DH-Based Draft



{} Indicates messages protected using keys derived from the handshake master secret.

[] Indicates messages protected using keys derived from the master secret.

Server Parameters (prototype)

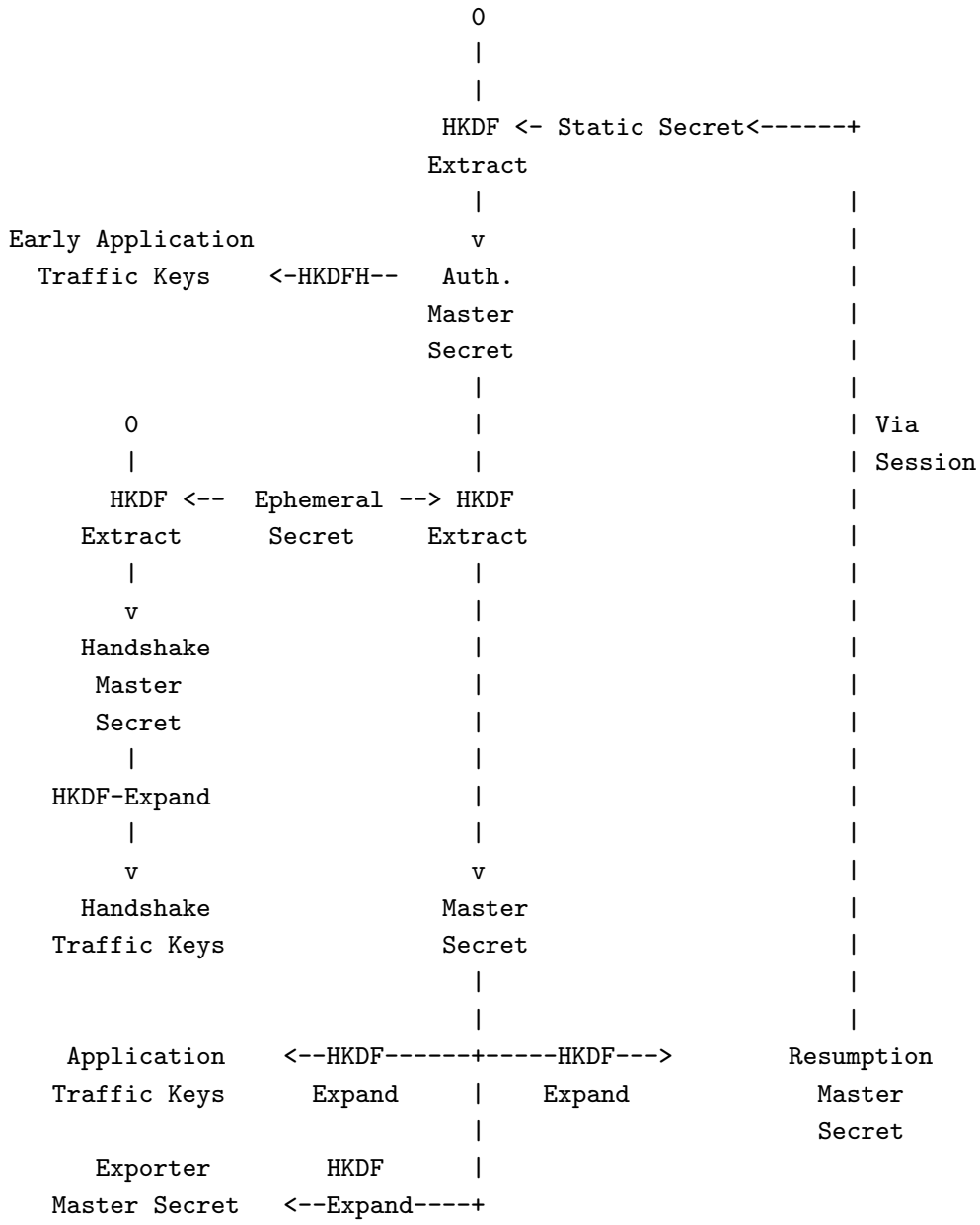
```
struct {
  uint64 not_before;
  uint64 not_after;
  opaque key_identifier<0..2^8-1>;
  NamedGroup group;
  opaque server_key<1..2^16-1>;
} UnsignedParameters;

enum { online(0), (255)} ParametersType;

struct {
    UnsignedParameters parameters;
    ParametersType params_type;
    digitally-signed struct {
      UnsignedParameters parameters;
      opaque session_hash[Hash.length];
    };
    opaque zero_rt_id<0..2^16-1>;
} ServerParameters;
```

What's being signed by the server?

- Online signature
  - Rough consensus not to do offline now
- Signature over `session_hash + UnsignedParameters`


0
HKDF <- Static Secret<------+
Extract
Early Application
Traffic Keys <-HKDFH-- Auth.
Master
Secret
0
HKDF <-- Ephemeral --> HKDF
Extract Secret Extract
Handshake
Master
Secret
HKDF-Expand
Handshake
Traffic Keys
Master
Secret
Via
Session
Application <--HKDF------+-----HKDF---> Resumption
Traffic Keys Expand | Expand Master
Secret
Exporter HKDF
Master Secret <--Expand----+

Performance Comparison (ignoring amortized computations and fixed base)

Mode	Client	Server
Current TLS 1.3 (1-RTT)	Verify + 1 VB	Sign + 1 VB
OPTLS (online signing)	Verify + 2 VB	Sign + 2 VB
OPTLS (offline signing)	Verify + 2 VB	2 VB
OPTLS (online signing, $g^s == g^y$)	Verify + 1 VB	Sign + 1 VB
OPTLS (HMQV)	Verify + 1.x VB	Sign + 1.x VB
0-RTT	2 VB	2 VB

How do we provide keys for 0-RTT

- Reuse/cache the keys from 1-RTT
  - This means you can't use $g^s == g^y$
  - Annoying tradeoff for the server
- Have the server supply a separate key
  - Two keys in ServerParameters?
  - Some other encoding
- Neither of these is that awesome

HKDF

- Some sense that we should convert to HKDF
  - Clearer cryptographic properties
  - Is there anything wrong with the TLS PRF?
- This is an orthogonal question
  - But I'd like to make the change at the same time

AEAD IV (I)

- PR ???
- TLS 1.2 (well, GCM) uses a partially explicit IV
  - 32 bits generated from MS
  - 64 bits explicit
  - 32 bits block counter
- This chews up bandwidth
  - ChaCha draft reuses sequence number
  - OK (?) if the module checks

AEAD IV (II)

- Interim consensus: use record sequence number and pad with 0s
- Brian Smith: should we have a random offset?
- Options
  - Do nothing
  - Per-session prefix
  - Per-session offset
  - Per-session XOR mask
- Let's decide

Other issues?