Beyond the 64K Limit in TLS 1.3 Handshake

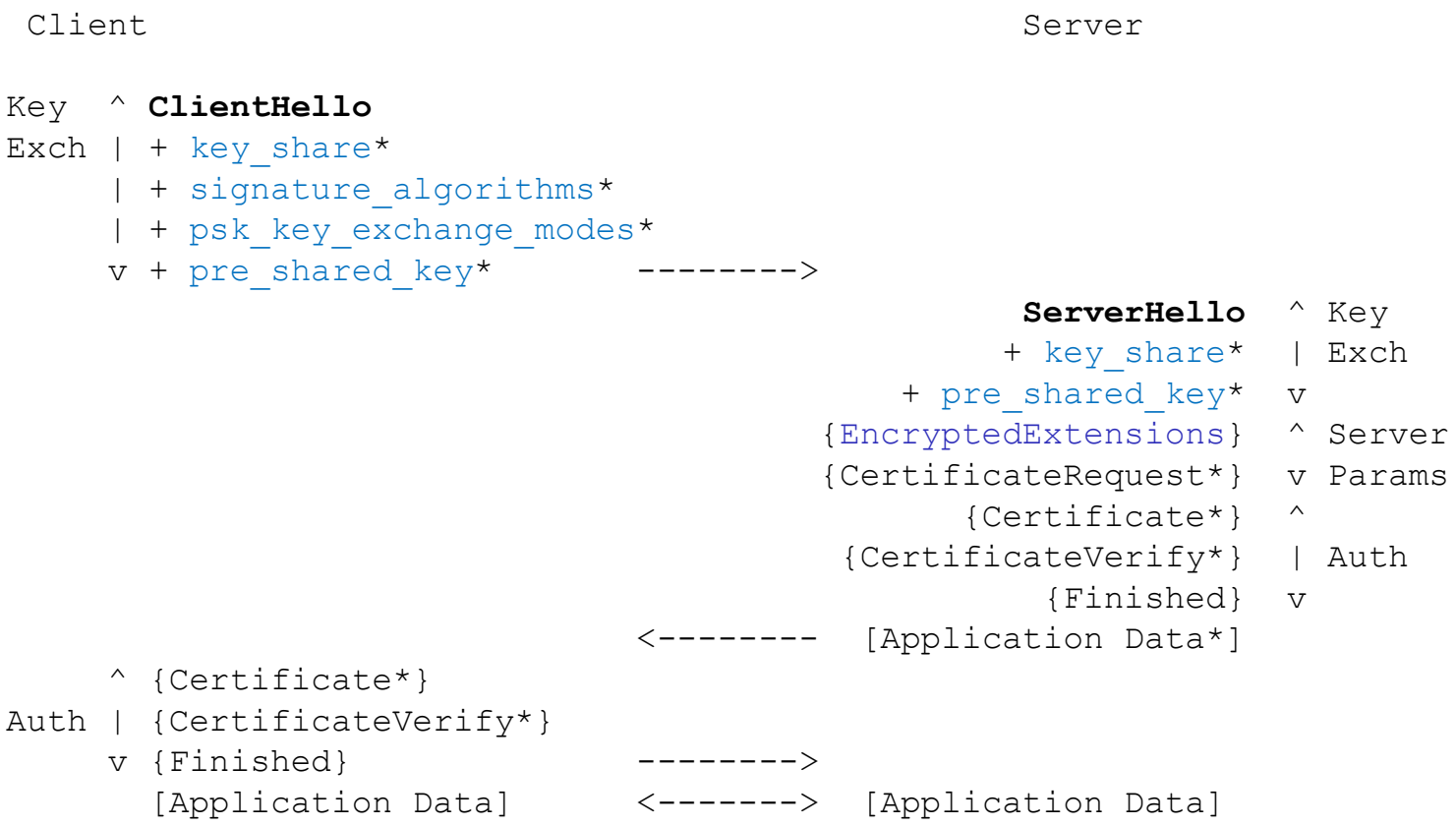
draft-wagner-tls-keysharepqc-08

Jonathan Wagner (jwagne31@charlotte.edu)
Yongge Wang (yongge.wang@charlotte.edu)
Valery Smyslov (svan@elvis.ru)
Yoav Nir (ynir.ietf@gmail.com)

IETF 125

TLS 1.3 Handshake

Basic full TLS handshake:



Hardcoded Limits on Data Size

TLS 1.3 has hardcoded limits on the size of its data structures

- the size of Handshake messages is limited to **2^24** bytes
- the size of extensions in CH & SH (& EE) is limited to **2^16-1** bytes
  - the size of each extension content is also limited to **2^16-1** bytes

```
struct {
   HandshakeType msg_type;    /* handshake type */
   uint24 length;             /* remaining bytes in message */
   select (Handshake.msg_type) {
       case client_hello:          ClientHello;
       case server_hello:          ServerHello;
       ...
   };
} Handshake;
```

```
struct {
   ProtocolVersion legacy_version = 0x0303;
   Random random;
   opaque legacy_session_id<0..32>;
   CipherSuite cipher_suites<2..2^16-2>;
   opaque legacy_compression_methods<1..2^8-1>;
   Extension extensions<8..2^16-1>;
} ClientHello;
```

```
struct {
  ProtocolVersion legacy_version = 0x0303;
  Random random;
  opaque legacy_session_id_echo<0..32>;
  CipherSuite cipher_suite;
  uint8 legacy_compression_method = 0;
  Extension extensions<6..2^16-1>;
} ServerHello;
```

```
struct {
  ExtensionType extension_type;
  opaque extension_data<0..2^16-1>;
} Extension;
```

What if we need more space?

- An immediate goal: key_share cannot fit public key of some PQ KEMs (or their composition)
  - **IMPORTANT**: we **do not discuss** these KEMs for TLS and do not discuss **whether** they may be needed, we focus only on **how** this can be done
- A broader goal: be prepared for larger TLS extensions
  - ECH already roughly doubles the size of outer CH, future extensions may require more
  - **IMPORTANT**: we **do not discuss** whether this is likely to happen and is **suitable** for most uses of TLS (e.g. for web browsing), we focus on **how** this can be done

Proposal 1 – Change CH & SH Format

- Just increase the size of extensions and define a new (bigger) extension for (some) PQ key shares

```
struct {
  NamedGroup group;
  select (KeyShareEntry.group) {
  case <large PQ alg>: Empty;
  ...
  default:    opaque key_exchange<1..2^16-1>;
  }
} KeyShareEntry;
```

```
struct {
  NamedGroup group;
  select (KeyShareEntryPQC.group) {
  case <large PQ alg>: opaque key_exchange<1..2^24-1>;
  ...
  default:             Empty;
  }
} KeyShareEntryPQC;
```

```
struct {
  ExtensionType extension_type;
  opaque extension_data<0..2^24-1>;
} Extension;
```

```
struct {
  KeyShareEntry client_shares<0..2^16-1>;
  KeyShareEntryPQC client_shares<0..2^24-1>;
} KeyShareClientHello;
```

```
struct {
  KeyShareEntry server_share;
  KeyShareEntryPQC server_sharePQC;
} KeyShareServerHello;
```

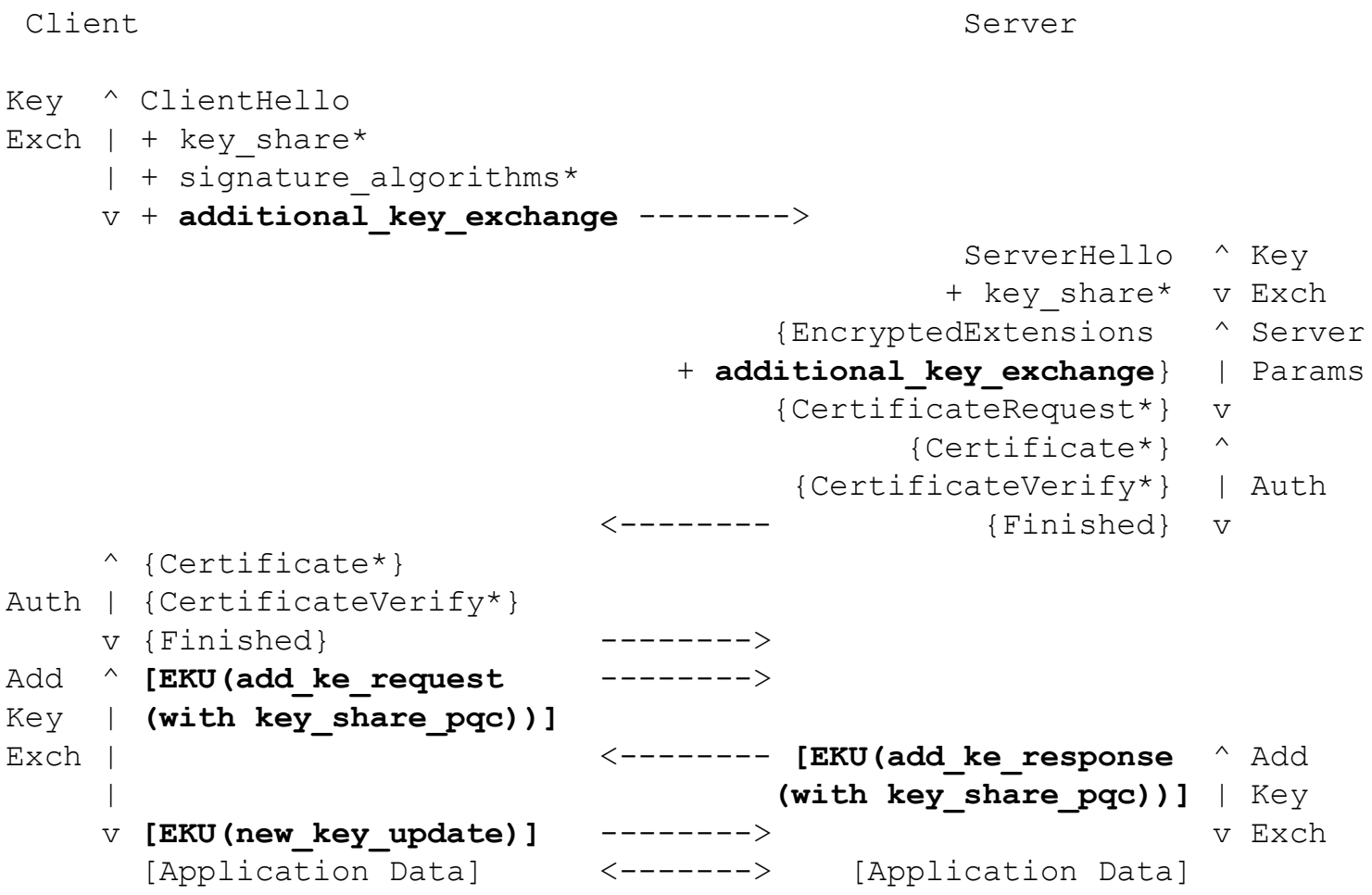
Proposal 1 - Pro & Cons

- **Simplicity (just works)**
- TLS state machine is not affected (the sequence of handshake messages remains the same)
- No additional round trips
- **No backward compatibility (works only if both client and server support this new format)**
- Applicable only for key shares (not a general solution)
- It is not clear how this will interact with ECH
- It is not clear how middleboxes will handle modified CH and SH

Proposal 2 – Utilize Extended Key Update

- draft-ietf-tls-extended-key-update defines a mechanism for post-handshake update of session keys using new key exchange
  - currently this is limited only to group negotiated during handshake, but can be extended to negotiate a different key exchange
- Use of Extended Key Update for additional key exchange would make the key exchange in TLS always hybrid and non-composite (a la in IKEv2)
  - application data must be delayed until additional key exchange finishes

Proposal 2 – Message Sequence



Proposal 2 - Pro & Cons

- **No modifications to TLS handshake messages (no problems with ECH)**
- No appeared problems with middleboxes
- Backward compatible solution
- **TLS state machine is affected (application data must be delayed until additional key exchange completes)**
- Complexity
- Applicable only for key shares (not a general solution)
- It is unclear how this would interact with regular Extended Key Update

Proposal 3 – New Handshake Message

- Define new handshake message AuxHandshakeData that would follow CH or SH, put chunks of large data there and reference these chunks from CH/SH
- Negotiate its presence via HRR

```
struct {
  HandshakeType msg_type;
  uint24 length;             /* bytes in message */
  select (Handshake.msg_type) {
  ...
  case aux_handshake_data:    AuxHandshakeData;
  };
} Handshake;
```

```
enum {
  ...
  aux_data(TBD),
  (65535)
} ExtensionType;

struct {
} AuxData;
```

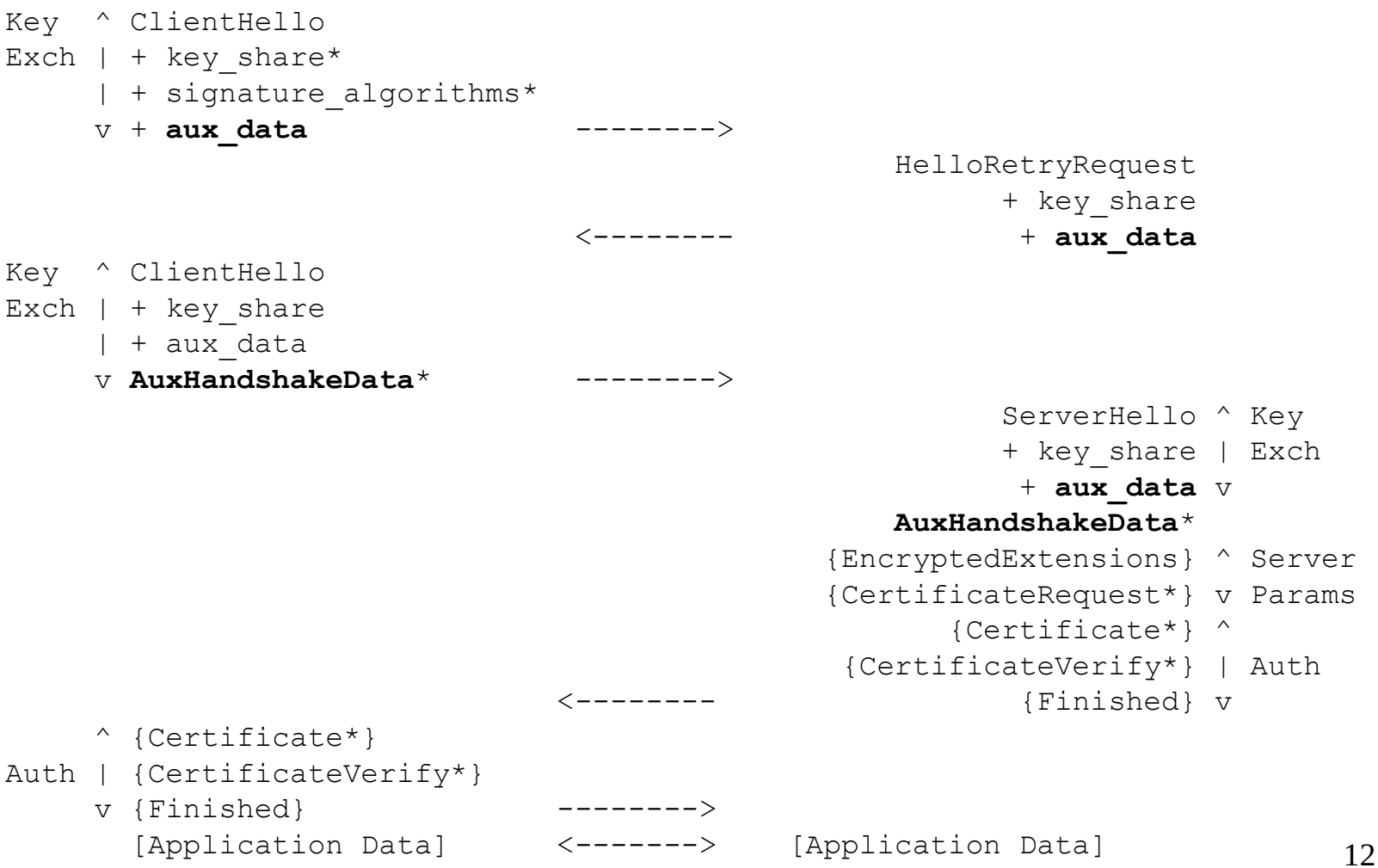
```
struct {
  opaque  data<0..2^24-1>;
} AuxHandshakeDataEntry;

struct {
  AuxHandshakeDataEntry   aux_data<0..2^24-1>;
} AuxHandshakeData;
```

Proposal 3 – New Handshake Message for Large Key Shares

- In particular case of key shares, key_exchange for those KEMs may be treated as LargeKeyShareRepresentation (alternatively, its size may indicate whether is key share data itself or reference to AuxHandshakeData where actual key share data is)

```
struct {
  uint8 form;
  select (LargeKeyShareRepresentation.form) {
  case 0:     uint16      aux_data_entry_index;
  default:    opaque      key_exchange<0..2^16-1>;
  };
} LargeKeyShareRepresentation;
```

Proposal 3 – Message Sequence



Proposal 3 - Pro & Cons

- **Generic solution (not only for key shares)**
- No modifications to CH & SH messages (hopefully this simplifies interaction with ECH)
- Backward compatible solution
- **HRR is always needed unless client knows that server supports this extension**
- It is unclear how this would interact with middleboxes (but hopefully they only look closely at CH and SH)

Thanks!

Comments?

Any other ideas?