

Post-Handshake Authentication via PAKE for TLS 1.3

draft-guo-pake-pha-tls-01

<https://datatracker.ietf.org/doc/draft-guo-pake-pha-tls/01/>

Wei Guo, Liang Xia, Ji Li

Huawei

Introduction: Post-Handshake Authentication via PAKE for TLS 1.3

- ❑ In some cases, it is desirable to use **PAKE**-based post-handshake authentication over TLS channel to execute **password authentication** between a client and a server. The reasons are as follows:
 - The post-handshake authentication **does not need to change** the current TLS 1.3 protocol stack.
 - This can **defend against password leakages** caused by potential phishing attacks.
 - This strategy is often called defense-in-depth, the security of application-layer password authentication is still guaranteed even if the security of the underlying TLS-layer is broken.
 - The post-handshake authentication can optionally support **channel binding** to defend against potential man-in-the-middle (MITM) attacks on the underlying TLS channel.

- ❑ Note that the post-handshake authentication via **OPAQUE** has been discussed in [I-D.draft-sullivan-tls-opaque-01], which utilizes the mechanism of **Exported Authenticators** in TLS 1.3 [RFC9261]. However, this mechanism is **not applicable to** the following PAKE algorithms, where the client and server **do not have** long-term secret/public keys.
 - **SPAKE2** [RFC9382].
 - **CPace** [I-D.draft-irtf-cfrg-pace-13].
 - **SPAKE2+** [RFC9383].
 - ...

- ❑ Lastly, **PAKE algorithms negotiation** should also be considered.

PAKE messages: Post-Handshake Authentication via PAKE for TLS 1.3

To use PAKE as an application-layer password authentication over TLS 1.3, the following **PAKEHandshake** messages are defined.

```
struct {
    PAKEHandshakeType msg_type;           /* PAKE message type */
    uint24 length;                       /* bytes in message */
    select (PAKEHandshake.msg_type) {
        case pake_client_hello:          ① PAKEClientHello;
        case pake_server_hello:          ② PAKEServerHello;
        case pake_hello_retry_request:   ③ PAKEHelloRetryRequest;
        case pake_finished:              ④ PAKEFinished;
        case pake_status:                ⑤ PAKEStatus;
    };
} PAKEHandshake;
```

❑ The PAKEClientHello, PAKEServerHello and PAKEHelloRetryRequest messages are used to execute PAKE algorithms negotiation and the negotiated PAKE key exchange.

❑ The PAKEFinished message is used to send a message authentication code (MAC) for identity authentication, key confirmation, and handshake integrity.

❑ The PAKEStatus message is used to send the execution status of the PAKE protocol, indicating whether it has been successfully executed.

```
struct {
    PAKEAlgorithm supported_pake_algorithms<2..2^16-1>;
    opaque client_identity<0..2^16-1>;
    PAKEShareEntry client_shares<0..2^16-1>;
} PAKEClientHello; ①
```

```
struct {
    opaque server_identity<0..2^16-1>;
    PAKEShareEntry server_share;
} PAKEServerHello; ②
```

```
struct {
    PAKEAlgorithm selected_pake_algorithm;
} PAKEHelloRetryRequest; ③
```

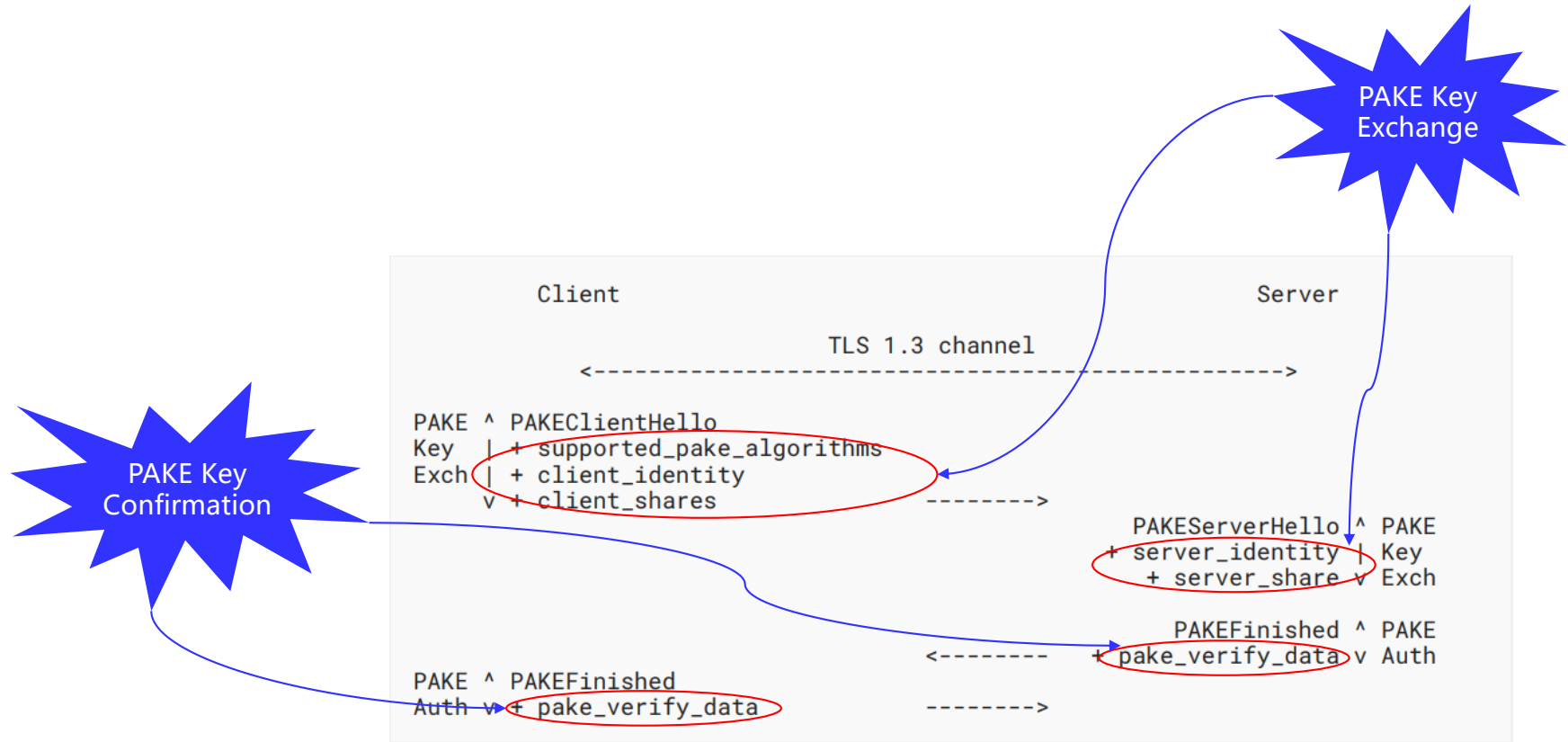
```
struct {
    opaque pake_verify_data[MAC.length];
} PAKEFinished; ④
```

```
enum {
    pake_success_notify (0),
    pake_unexpected_message (1),
    pake_handshake_failure (2),
    pake_illegal_parameter (3),
    pake_decode_error (4),
    pake_decrypt_error (5),
    pake_insufficient_security (6),
    pake_internal_error (7),
    (255)
} PAKEStatusDescription;
```

```
struct {
    PAKEStatusDescription description;
} PAKEStatus; ⑤
```

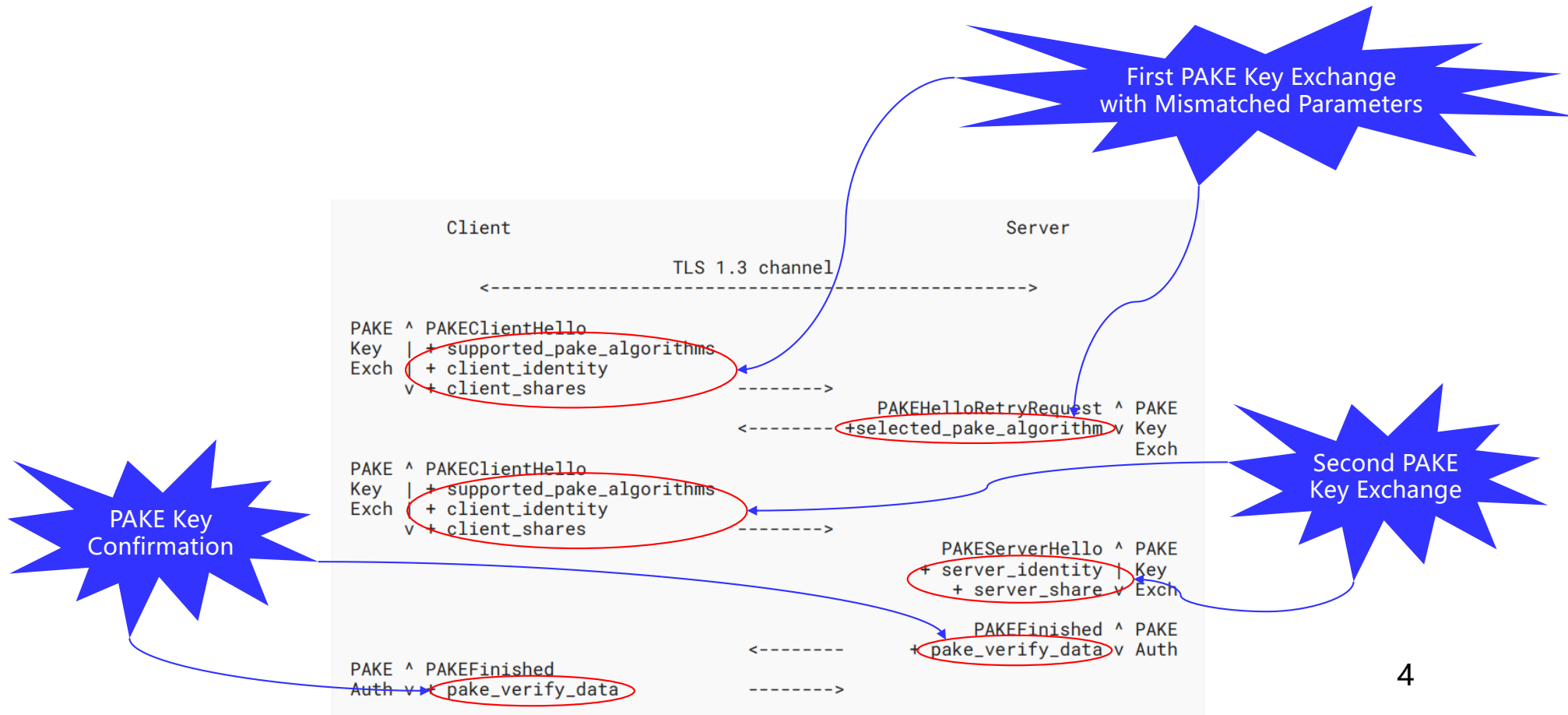
Basic Handshake: Post-Handshake Authentication via PAKE for TLS 1.3

Message flow for full PAKE handshake over TLS 1.3 channel.



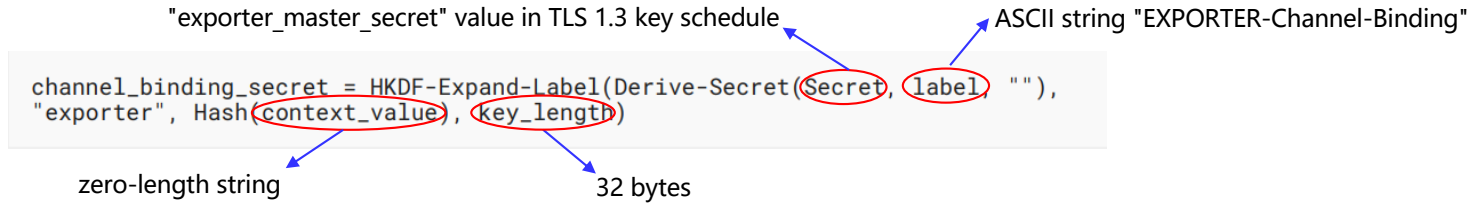
Handshake with Retry: Post-Handshake Authentication via PAKE for TLS 1.3

Message flow for full PAKE handshake with mismatched parameters over TLS 1.3 channel.

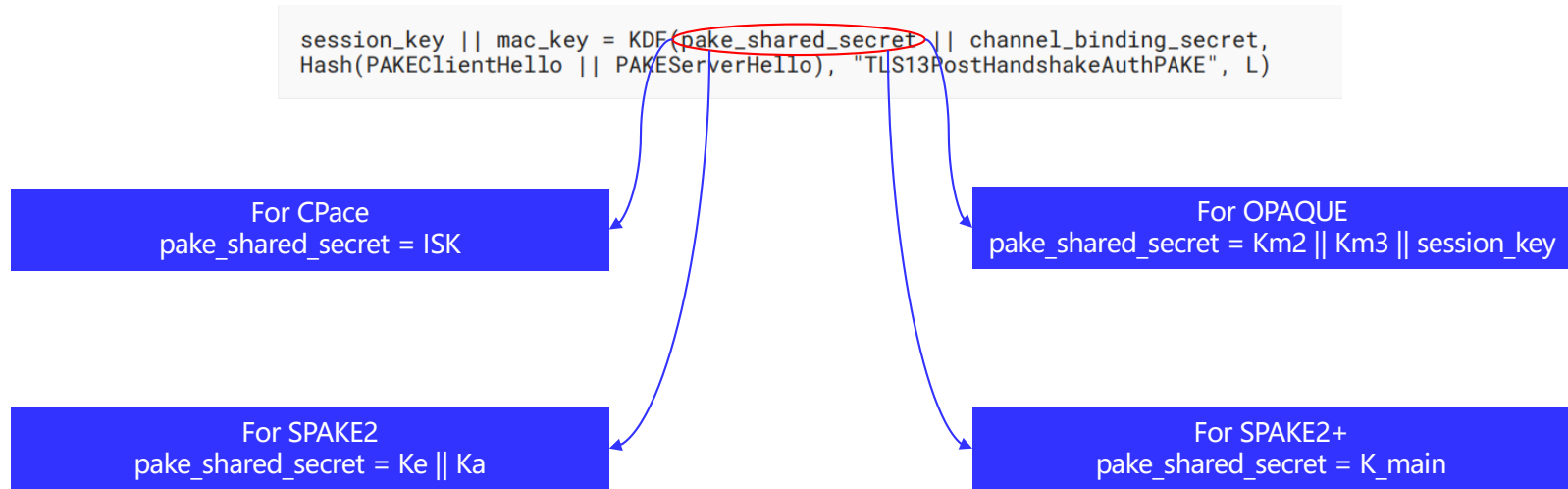


Channel Binding: Post-Handshake Authentication via PAKE for TLS 1.3

- According to the channel binding type of "tls-exporter" defined in [RFC9266], the **channel binding secret** is computed as follows.



- When channel binding is used, the **session key** and **MAC key** are computed as follows.



Thanks