Learning TLS1.3 with Go

2024.3.21

Talk Overview

- TLS1.3 overview
  - Three Essential Security Properties
  - Cryptographic Building Blocks
  - Brief History of SSL/TLS
  - Key Features and Improvements in TLS1.3
  - TLS1.3 Handshake Flow
- Implementation with Go
  - TLS Record Protocol
  - ClientHello, ServerHello
  - Key Schedule
  - Certificate, CertificateVerify
  - Finished (Server), Finished (Client)
  - After TLS Handshake
- Summary

Demo code: https://github.com/shu-yusa/go-tls/

Why I chose TLS1.3 as a theme

- Personal interests
  - Broad interest in network protocols
  - Inspired by “Bulletproof TLS and PKI” (Recently translated into JP)
- Importance of TLS1.3
  - Improved security
    - Removes unsafe elements and fixes vulnerabilities
  - Improved performance
    - Shorter handshake RTT, reduces latency
- TLS in Go
  - Go has own implementation (not OpenSSL-based)
  - Rich cryptographic packages in standard library
  - Demo code: https://github.com/shu-yusa/go-tls/

TLS 1.3 Overview

Three Essential Security Properties

- **Confidentiality**
  - Ensuring that the communication is not readable by unauthorized parties
- **Integrity**
  - Verifying that the message has not been altered during transmission
- **Authenticity**
  - Confirming the identity of the communicating parties

Cryptographic Building Blocks

- **Symmetric-key Cryptography**
  - Provides confidentiality, e.g. AES, ChaCha20
  - Provides integrity and authenticity when combined with AEAD, e.g. AES-GCM
- **Public-key Cryptography**
  - Facilitates encryption, digital signatures, and key exchange, e.g. RSA, ECC
- **One-way Hash Functions**
  - Used in digital signatures, MAC, key derivation, RPNG, etc, e.g. SHA-256
- **Message Authentication Code (MAC)**
  - Provides integrity and authenticity, e.g. HMAC
- **Digital Signatures**
  - Ensures integrity, authenticity, and non-repudiation, e.g. RSA, ECDSA
- **Pseudorandom Number Generators (PRNGs)**
  - Generate random keys and nonces for various cryptographic operations

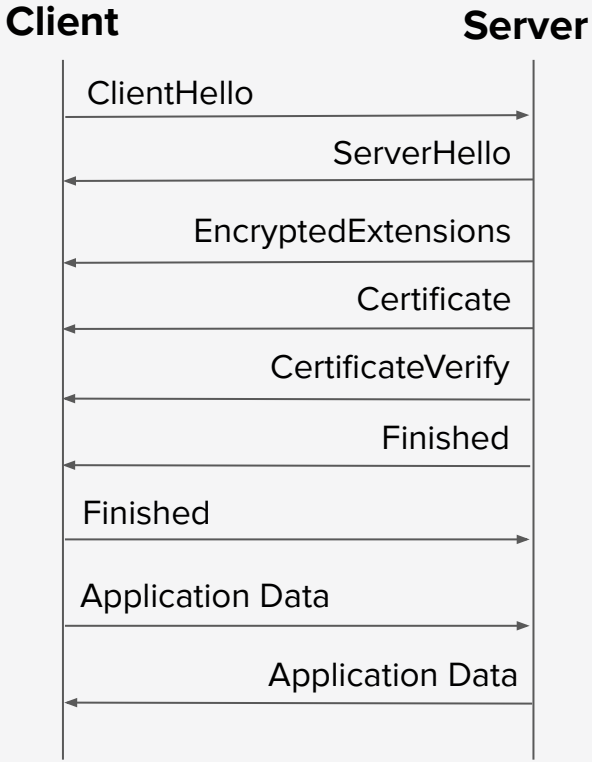
Brief History of SSL/TLS

- SSL (Secure Sockets Layer)
  - Developed by Netscape in 1994
  - SSL 2.0 - first released version in 1994, prohibited in 2011 (RFC 6176)
  - SSL 3.0 - released in 1995, deprecated in 2015 (RFC 7568)
- TLS (Transport Layer Security)
  - Standardized version of SSL 3.0 by IETF
  - TLS 1.0 - released in 1999, deprecated in 2021 (RFC 8996)
  - TLS 1.1 - released in 2006, deprecated in 2021 (RFC 8996)
  - TLS 1.2 - released in 2008, SHA-256, Authenticated encryption, etc
  - TLS 1.3 - released in 2018

Key Features and Improvements in TLS1.3

- Major overhaul with significant improvements in security and performance
- Faster handshake
  - 1-RTT (Round Trip Time) for full handshake (2-RTT in TLS1.2)
  - 0-RTT mode available, but with some security tradeoffs
- Modernized cipher suites
  - More secure algorithms, legacy ones removed
  - AEAD mandatory for record protection, providing confidentiality, integrity, and authenticity
- Improved key exchange algorithms
  - Removal of RSA-based key exchange, ensuring forward secrecy
  - Only Diffie-Hellman-based key exchange allowed (e.g., ECDHE)

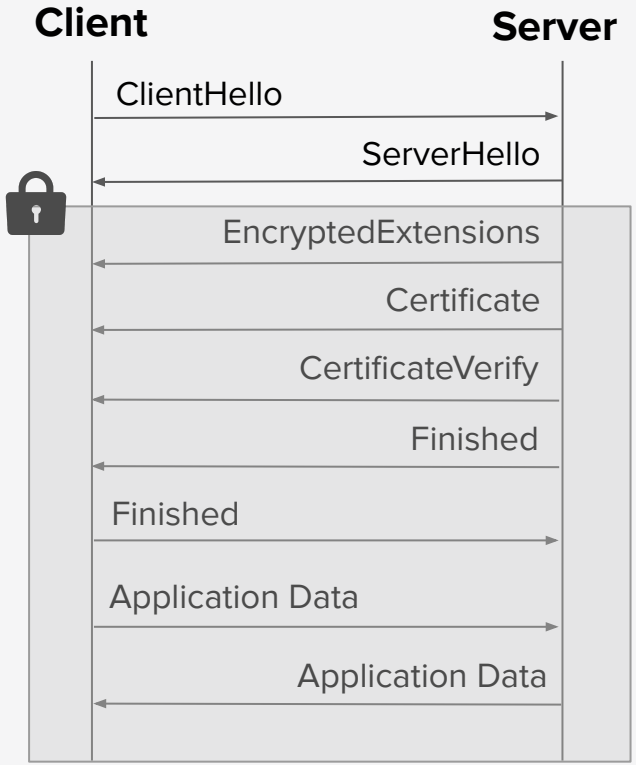
TLS1.3 Handshake Flow



TLS establishes a secure communication channel:

- Key Exchange
  - Agree on the cryptographic algorithms
  - Exchange public keys to derive shared secret keys
- Server Authentication
  - Client verifies server's identity based on its certificate
- Encryption of Application Data
  - Use shared keys to encrypt application data

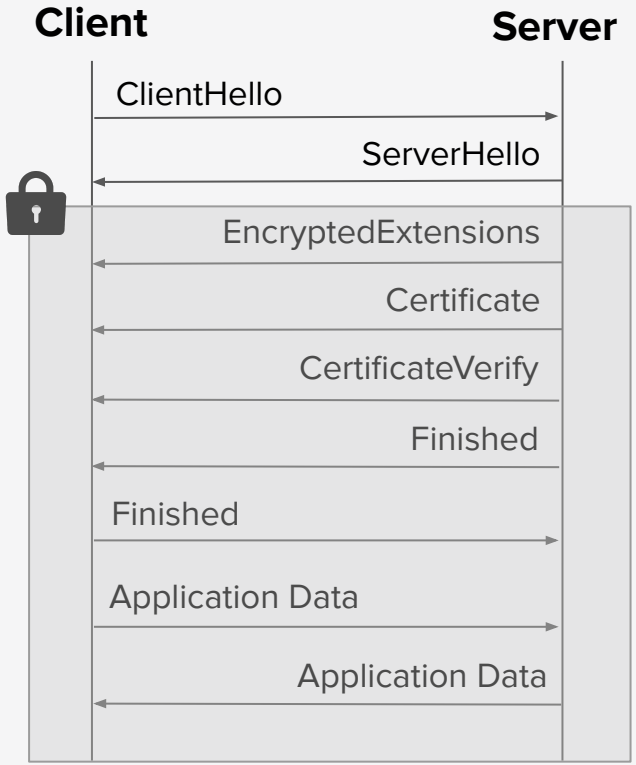
TLS1.3 Handshake Flow



TLS establishes a secure communication channel:

- Key Exchange
  - Agree on the cryptographic algorithms
  - Exchange public keys to derive shared secret keys
- Server Authentication
  - Client verifies server's identity based on its certificate
- Encryption of Application Data
  - Use shared keys to encrypt application data

TLS1.3 Handshake Flow



TLS establishes a secure communication channel:

- Key Exchange
  - Agree on the cryptographic algorithms
  - Exchange public keys to derive shared secret keys
- Server Authentication
  - Client verifies server's identity based on its certificate
- Encryption of Application Data
  - Use shared keys to encrypt application data

Exploring each step with Go code examples

TLS1.3 Handshake with Go

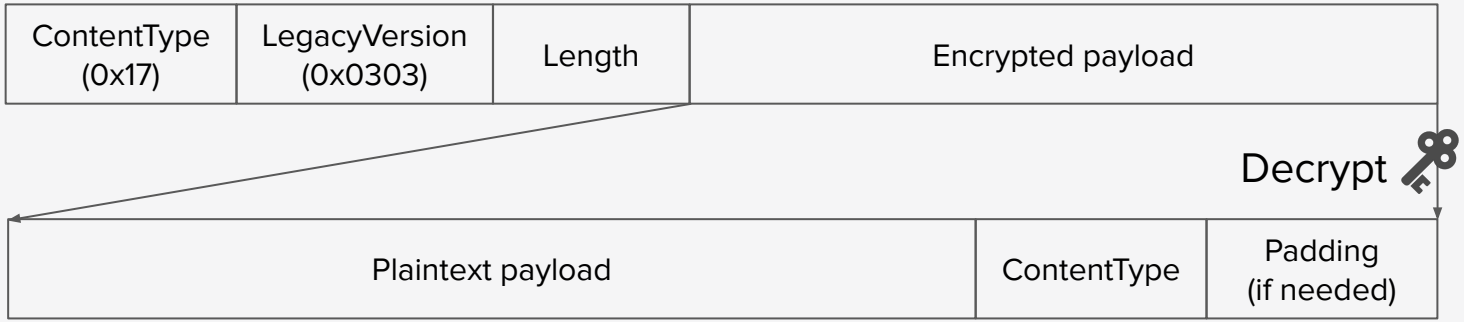
TLS Record Protocol

ContentType	LegacyVersion (0x0303)	Length	Payload

- **ContentType**: Subprotocol type, 1 byte
  - handshake (0x16), application_data (0x17), alert (0x15), etc
  - Always application_data (0x17) for encrypted data
- **LegacyVersion**: Set to 0x0303 (TLS 1.2) for backward compatibility, 2 bytes
- **Length**: Payload length, 2 bytes
- **Payload**: Data payload
  - Plaintext for certain handshake messages (e.g., ClientHello, ServerHello)
  - Encrypted data for later handshake messages and application data

TLS Record Protocol

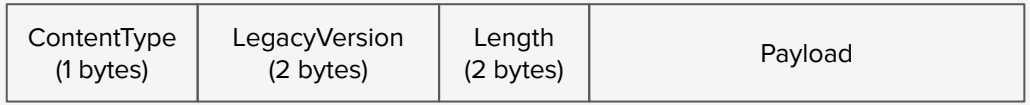
- For an encrypted message, the TLS record is "wrapped" by an application_data (0x17) TLS record



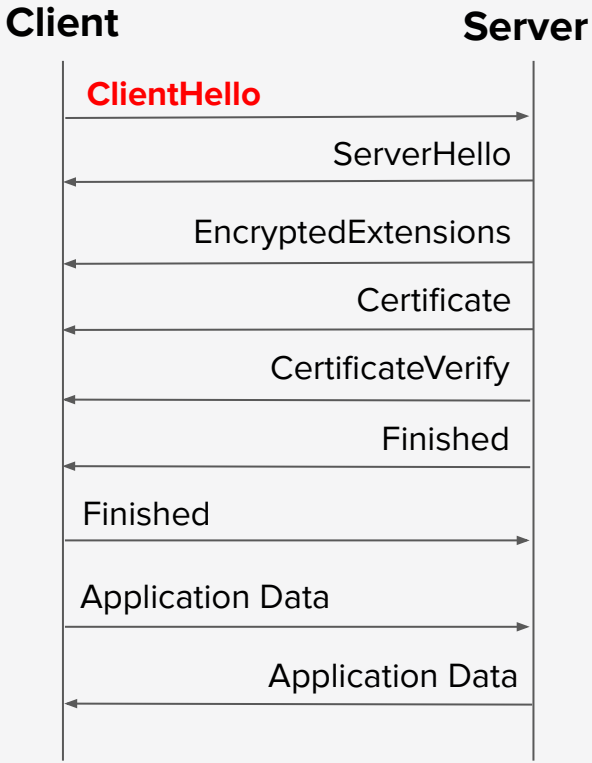
TLS Record Protocol

```
func TLSServer() {
  listener, _ := net.Listen("tcp", ":443")
  for {
    conn, _ := listener.Accept()
    go func(conn net.Conn) {
      // Read TLS Record header
      tlsHeaderBuffer := make([]byte, 5)
      conn.Read(tlsHeaderBuffer)
      length := binary.BigEndian.Uint16(tlsHeaderBuffer[3:5])
      // Read TLS Record payload
      payloadBuffer := make([]byte, length)
      io.ReadFull(conn, payloadBuffer)

      tlsRecord := &TLSRecord{
        ContentType:         ContentType(tlsHeaderBuffer[0]),
        LegacyRecordVersion: ProtocolVersion(binary.BigEndian.Uint16(tlsHeaderBuffer[1:3])),
        Length:              length,
        Fragment:            payloadBuffer,
      }
      switch tlsRecord.ContentType {
        // Handle Record (handshake, application_data, etc)
      }
    }(conn)
  }
}
```

- The server listens on port 443
- Parse record header (5 bytes) for each connection
- Parse payload according to the length
- Handle the record according to the ContentType



ClientHello



- First message sent by the client to initiate a TLS handshake
- Contains client's capabilities and preferences

TLS Record payload

HandshakeType (0x01)	Length (3 bytes)	Handshake Message (ClientHello)

```
uint16 ProtocolVersion;
opaque Random[32];
uint8 CipherSuite[2];

struct {
    ProtocolVersion legacy_version = 0x0303; /* TLS1.2 */
    Random random; /* 32 bytes random */
    opaque legacy_session_id<0..32>; /* for compatibility */
    CipherSuite cipher_suites<2..2^16-2>;
    opaque legacy_compression_methods<1..2^8-1>; /* zero */
    Extension extensions<8..2^16-1>;
} ClientHello;
```

ClientHello

```
> openssl s_client -connect localhost:443 -tls1_3 (more options…)
...
write to 0x60000202c000 [0x12300e600] (183 bytes => 183 (0xB7))
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

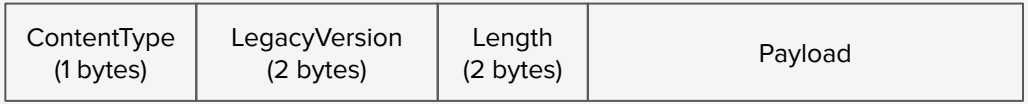
ClientHello

> openssl s_client -connect localhost:443 -tls1_3 (more options…)

```
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

ContentType

```
type ContentType uint8
const (
  InvalidRecord          ContentType = 0x00
  ChangeCipherSpecRecord ContentType = 0x14
  AlertRecord            ContentType = 0x15
  HandshakeRecord        ContentType = 0x16
  ApplicationDataRecord  ContentType = 0x17
)
```



ClientHello

> openssl s_client -connect localhost:443 -tls1_3 (more options…)

```
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

Legacy Version (TLS1.2 or TLS1.0)

```
type ProtocolVersion  uint16
const (
  TLS10 ProtocolVersion = 0x0301
  TLS11 ProtocolVersion = 0x0302
  TLS12 ProtocolVersion = 0x0303
  TLS13 ProtocolVersion = 0x0304
)
```

ContentType (1 bytes)	LegacyVersion (2 bytes)	Length (2 bytes)	Payload

ClientHello

```
> openssl s_client -connect localhost:443 -tls1_3 (more options…)
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

Payload Length

0x00b2 = 16 * 11 + 2 = 178 bytes

Rest of the data is the payload

ContentType (1 bytes)	LegacyVersion (2 bytes)	Length (2 bytes)	Payload

ClientHello

> openssl s_client -connect localhost:443 -tls1_3 (more options…)

```
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

HandshakeType

```
type HandshakeType uint8
const (
  ClientHello         HandshakeType = 0x01
  ServerHello         HandshakeType = 0x02
  EncryptedExtensions HandshakeType = 0x08
  Certificate         HandshakeType = 0x0b
  CertificateVerify   HandshakeType = 0x0f
  Finished            HandshakeType = 0x14
)
```

HandshakeType (0x01)	Length (3 bytes)	Handshake Message (ClientHello)

ClientHello

> openssl s_client -connect localhost:443 -tls1_3 (more options…)

```
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

Handshake Message Length

0x00ae = 16 * 10 + 14 = 174 bytes

```
type HandshakeType    uint8
msgType := HandshakeType(tlsRecord.Fragment[0])
// extract 3 bytes (Big Endian)
handshakeLength :=
  (uint32(tlsRecord.Fragment[1]) << 16) |
  (uint32(tlsRecord.Fragment[2]) << 8)  |
  uint32(tlsRecord.Fragment[3])
```

HandshakeType (0x01)	Length (3 bytes)	Handshake Message (ClientHello)

ClientHello

```
> openssl s_client -connect localhost:443 -tls1_3 (more options…)
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

Handshake Message Payload

```
type (
  CipherSuite uint16
  ClientHelloMessage struct {
    LegacyVersion           ProtocolVersion
    Random                  []byte
    LegacySessionID         []byte
    CipherSuites            []CipherSuite
    LegacyCompressionMethod []byte
    Extensions              []Extension
  }
)
```

HandshakeType (0x01)	Length (3 bytes)	Handshake Message (ClientHello)

ClientHello

```
> openssl s_client -connect localhost:443 -tls1_3 (more options…)
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

ClientHello LegacyVersion

```
type (
  ProtocolVersion  uint16
  ClientHelloMessage struct {
    LegacyVersion           ProtocolVersion
    Random                  []byte
    LegacySessionID         []byte
    CipherSuites            []CipherSuite
    LegacyCompressionMethod []byte
    Extensions              []Extension
  }
)
```

ClientHello

```
> openssl s_client -connect localhost:443 -tls1_3 (more options…)
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

Random (32 bytes)

```
type (
  ClientHelloMessage struct {
    LegacyVersion           ProtocolVersion
    Random                  []byte
    LegacySessionID         []byte
    CipherSuites            []CipherSuite
    LegacyCompressionMethod []byte
    Extensions              []Extension
  }
)
```

ClientHello

```
> openssl s_client -connect localhost:443 -tls1_3 (more options…)
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

Legacy Session ID

- Not used in TLS1.3
- Length (1 byte): 0x20 = 32 bytes

```
type (
  ClientHelloMessage struct {
    LegacyVersion           ProtocolVersion
    Random                  []byte
    LegacySessionID         []byte
    CipherSuites            []CipherSuite
    LegacyCompressionMethod []byte
    Extensions              []Extension
  }
)
```

ClientHello

```
> openssl s_client -connect localhost:443 -tls1_3 (more options…)
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

Cipher Suites

- Cipher suites supported by the client
- Length (2 bytes): 0x06 = 6 bytes

```
type (
  CipherSuite uint16
  ClientHelloMessage struct {
    LegacyVersion           ProtocolVersion
    Random                  []byte
    LegacySessionID         []byte
    CipherSuites            []CipherSuite
    LegacyCompressionMethod []byte
    Extensions              []Extension
  }
)
const (
  TLS_AES_128_GCM_SHA256       CipherSuite = 0x1301
  TLS_AES_256_GCM_SHA384       CipherSuite = 0x1302
  TLS_CHACHA20_POLY1305_SHA256 CipherSuite = 0x1303
)
```

ClientHello

```
> openssl s_client -connect localhost:443 -tls1_3 (more options…)
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

Legacy Compression Method

- Length (1 byte): 0x01 = 1 byte
- No compression (0x00)
  - Only no compression is allowed

```
type (
  ClientHelloMessage struct {
    LegacyVersion           ProtocolVersion
    Random                  []byte
    LegacySessionID         []byte
    CipherSuites            []CipherSuite
    LegacyCompressionMethod []byte
    Extensions              []Extension
  }
)
```

ClientHello

```
> openssl s_client -connect localhost:443 -tls1_3 (more options…)
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

Extensions

- Extension enhances TLS capabilities
- Length (2 bytes): 0x005f = 105 bytes

```
type (
  ExtensionType uint16
  Extension struct {
    ExtensionType ExtensionType
    Length        uint16
    Data          []byte
  }
  ClientHelloMessage struct {
    LegacyVersion           ProtocolVersion
    Random                  []byte
    LegacySessionID         []byte
    CipherSuites            []CipherSuite
    LegacyCompressionMethod []byte
    Extensions              []Extension
  }
)
```

ClientHello

```
> openssl s_client -connect localhost:443 -tls1_3 (more options…)
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

Supported Groups Extension (0x000a)

- Supported elliptic curves
- Extension length: 0x0006 = 6 bytes
- Entries length: 0x0004 = 4 bytes

```
type (
  NamedGroup uint16
  SupportedGroupExtension struct {
    Length         uint16
    NamedGroupList []NamedGroup
  }
)
const (
  secp256r1 NamedGroup = 0x0017
  x25519    NamedGroup = 0x001d
)
```

ClientHello

```
> openssl s_client -connect localhost:443 -tls1_3 (more options…)
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

Signature Algorithms Extension (0x000d)

- Supported signature algorithms
- Extension length: 0x0006 = 6 bytes
- Entries length: 0x0004 = 4 bytes

```
type (
  SignatureScheme uint16
  SignatureAlgorithmsExtension struct {
    Length                      uint16
    SupportedSignatureAlgorithms []SignatureScheme
  }
)
const (
  ED25519                SignatureScheme = 0x0807
  ECDSA_SECP256R1_SHA256 SignatureScheme = 0x0403
)
```

ClientHello

```
> openssl s_client -connect localhost:443 -tls1_3 (more options…)
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

Supported Versions Extension (0x002b)

- Supported TLS versions
- Extension length: 0x0003 = 3 bytes
- Entries length: 0x02 = 2 bytes

```
type (
  ProtocolVersion  uint16
  ClientSupportedVersionsExtension struct {
    SelectedVersions []ProtocolVersion
  }
)
const (
  TLS10 ProtocolVersion = 0x0301
  TLS11 ProtocolVersion = 0x0302
  TLS12 ProtocolVersion = 0x0303
  TLS13 ProtocolVersion = 0x0304
)
```

ClientHello

```
> openssl s_client -connect localhost:443 -tls1_3 (more options…)
...
16 03 01 00 b2 01 00 00-ae 03 03 b5 28 8a e8 56
d7 18 3f fe cc 3d df ba-47 81 e4 dd f5 83 34 af
94 84 1d ba 89 26 e5 c6-cb 89 39 20 20 63 4d 59
fc a8 70 8e db fa 7c 5a-a4 e7 5a 99 92 58 9f 11
13 3e d8 12 cb db e7 40-12 b1 ed 92 00 06 13 02
13 03 13 01 01 00 00 5f-00 0b 00 04 03 00 01 02
00 0a 00 06 00 04 00 1d-00 17 00 23 00 00 00 16
00 00 00 17 00 00 00 0d-00 06 00 04 08 07 04 03
00 2b 00 03 02 03 04 00-2d 00 02 01 01 00 33 00
26 00 24 00 1d 00 20 59-4f 1b 75 88 f5 fe b8 c9
7d 7d eb 01 c6 df 05 58-a4 66 a0 b8 6b 87 ce ba
35 4b dd 33 46 b4 49
```

Key Share Extension (0x0033)

- ECDH parameter (client's preference)
- ECDH public key
- Extension Length: 0x0026 = 38 bytes
- Entries length: 0x0024 = 36 bytes
- Public Key length: 0x0020 = 32 bytes

```
type (
  NamedGroup uint16
  KeyShareEntry struct {
    Group           NamedGroup
    Length          uint16
    KeyExchangeData []byte
  }
  KeyShareExtension struct {
    Length       uint16
    ClientShares []KeyShareEntry
  }
)
const x25519 NamedGroup = 0x001d
```

ClientHello

Parsed result

- Supported TLS versions: TLS1.3
- Supported Cipher Suites
  - TLS_AES_256_GCM_SHA384
  - TLS_CHACHA20_POLY1305_SHA256
  - TLS_AES_128_GCM_SHA256
- Supported Elliptic Curve Named Group
  - x25519
  - secp256r1
- Shared Key
  - Named Group: x25519
  - Key: 594f1b7588f5feb8c97d7deb01c6df0558a466a0b86b87ceba354bdd3346b449
- Supported Signature Algorithms
  - ed25519
  - ecdsa_secp256r1_sha256

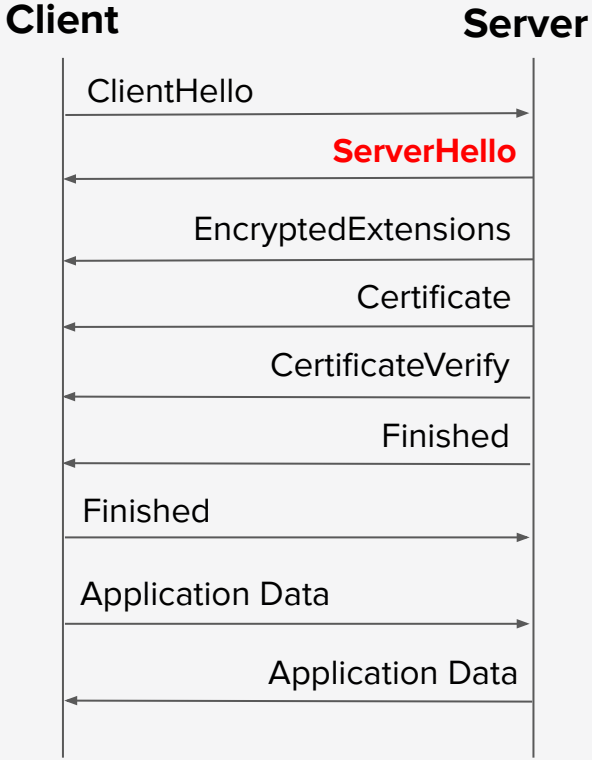
ClientHello

Parsed result

- Supported TLS versions: TLS1.3
- Supported Cipher Suites
  - TLS_AES_256_GCM_SHA384
  - TLS_CHACHA20_POLY1305_SHA256
  - TLS_AES_128_GCM_SHA256
- Supported Elliptic Curve Named Group
  - x25519
  - secp256r1
- Shared Key
  - Named Group: x25519
  - Key: 594f1b7588f5feb8c97d7deb01c6df0558a466a0b86b87ceba354bdd3346b449
- Supported Signature Algorithms
  - ed25519
  - ecdsa_secp256r1_sha256

The server agrees on the presented cipher suite and algorithms, and constructs the **ServerHello** message based on this agreement

ServerHello



- Purpose
  - Respond to ClientHello and establish the agreed parameters for the TLS connection
  - Share the server's public key for key exchange

- ECDH (Elliptic Curve Diffie-Hellman) Key Exchange
  1. Client and server agree on a curve (e.g., x25519)
  2. Each party generates a private-public key pair on the curve
  3. They exchange their public keys (KeyShare Extension)
  4. Each party computes the shared secret using their own private key and the other's public key. Both parties arrive at the same value
  5. The shared secret is used to derive symmetric keys for encryption and authentication

ServerHello

- Public key is shared to client by the KeyShare Extension
- Go “crypto/ecdh” package to generate an ECDH private-public key pair

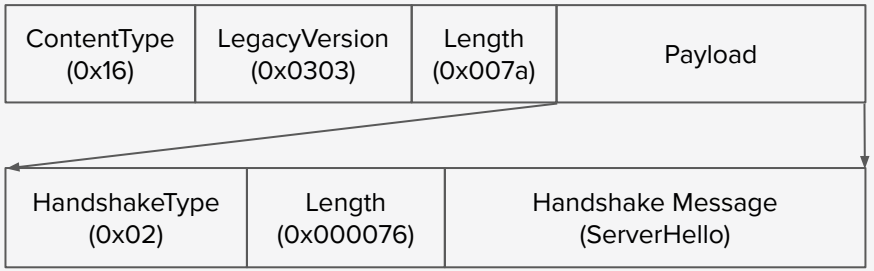
```
import (
  "crypto/ecdh"
  "crypto/rand"
)

ecdhServerPrivateKey, _ := ecdh.X25519().GenerateKey(rand.Reader)
ecdhServerPublicKey := ecdhServerPrivateKey.PublicKey()
serverKeyShareExtension := KeyShareExtension{
  // 4 bytes for NamedGroup, and Length
  Length: 4 + uint16(len(ecdhServerPublicKey.Bytes())),
  ClientShares: []KeyShareEntry{
    {
      Group:           x25519, // 0x001d
      Length:          uint16(len(ecdhServerPublicKey.Bytes())),
      KeyExchangeData: ecdhServerPublicKey.Bytes(),
    },
  },
}
```

ServerHello

- Construct a ServerHello handshake message, and send it to the client as a TLS record

```
serverHello := ServerHelloMessage{
  LegacyVersion:     TLS12, // 0x0303
  RandomBytes:       [32]byte(randomData),
  SessionID:         clientHello.LegacySessionID,
  CipherSuite:       TLS_AES_128_GCM_SHA256, // 0x1301
  CompressionMethod: 0x00, // No compression
  Extensions: []Extension{
    {
      // 0x002b
      ExtensionType: SupportedVersionsExtensionType,
      Length:        2,
      Data:          []byte{0x03, 0x04}, // TLS1.3
    },
    {
      ExtensionType: KeyShareExtensionType,// 0x0033
      Length:        serverKeyShareExtension.Length,
      Data:          serverKeyShareExtension.Bytes(),
    },
  },
}
```

```
16 03 03 00 7a
02 00 00 76 03 03 43 35-7b 97 c4 4e 00 f9 0a fc
7c 27 3b 34 4b 4f 35 c9-8d ef 01 ae 27 0d e4 1e
90 cd 90 dc 68 a2 20 20-63 4d 59 fc a8 70 8e db
fa 7c 5a a4 e7 5a 99 92-58 9f 11 13 3e d8 12 cb
db e7 40 12 b1 ed 92 13-01 00 00 2e 00 2b 00 02
03 04 00 33 00 24 00 1d-00 20 d3 56 e6 58 9f 6c
48 52 53 4d 0e 1c 54 57-c5 30 07 8c 91 b1 79 e2
61 85 4a 9d c4 33 51 9f-28 23
```



Key Schedule

- Overview
  - (server private key, client public key) => ECDH shared secret
  - ECDH shared secret => Symmetric keys (Shared keys)
  - Provides forward secrecy and key independence
- Key Derivation Process
  - Different keys for handshake and application data encryption/decryption
  - HKDF (HMAC-based key derivation function)
    - Combination of the “Extract” and “Expand” steps
  - TranscriptHash
    - Hash of concatenated handshake messages
  - Wrapper functions (utilities) of HKDF-Expand
    - HKDF-Expand + Key Label => **HKDF-Expand-Label**
    - HKDF-Expand-Label + TranscriptHash => **Derive-Secret**

Key Schedule

HKDF-Extract(key=0, salt=0) → Early-Secret

Derive-Secret(key=Early-Secret, label="derived", messages=[]) → Secret-State

HKDF-Extract(0, Secret-State) → Handshake-Secret

Derive-Secret(Handshake-Secret, "s hs traffic", [ClientHello, ServerHello])
→ server_handshake_traffic_secret

HKDF-Expand-Label(key=server_handshake_traffic_secret,
label="key", context=empty, length=16) → **Server Handshake Key**

HKDF-Expand-Label(key=server_handshake_traffic_secret,
label="iv", context=empty, length=12) → **Server Handshake IV**

IV = Initial Vector, used in AES (block ciphers)

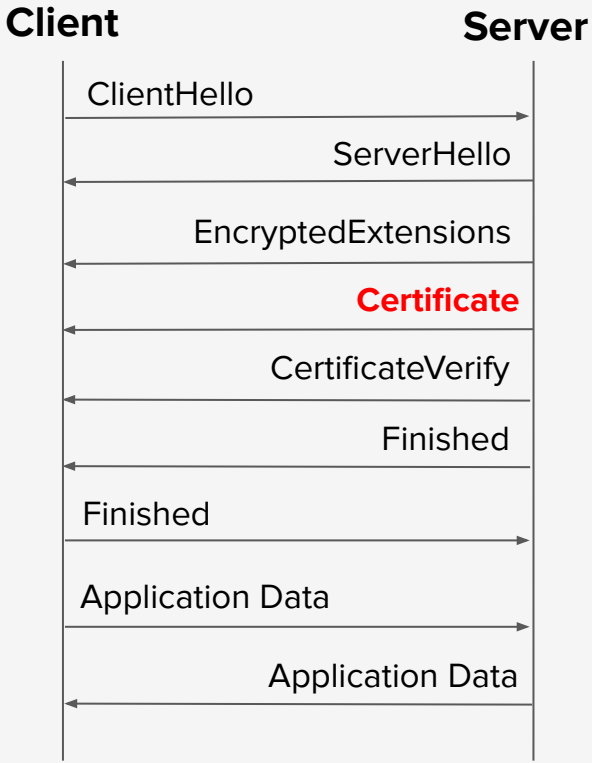
Also generate client_handshake_traffic_secret with a different label "c hs traffic" to decrypt a later client message ("Finished" handshake message)

Key Schedule

```
import (
  "crypto/ecdh"
  "crypto/sha256"
  "golang.org/x/crypto/hkdf"
)
// ECDH Shared secret
clientPublicKey, _ := ecdh.X25519().NewPublicKey(clientPublicKeyBytes) // From KeyShare Extension
sharedSecret, _ := ecdhServerPrivateKey.ECDH(clientPublicKey)        // Generated in ServerHello
// Early Secret
zero32 := make([]byte, sha256.New().Size())
earlySecret := hkdf.Extract(sha256.New, zero32, zero32)
secretState := DeriveSecret(earlySecret, "derived", [][]byte{})
// Handshake Secret
handshakeSecret := hkdf.Extract(sha256.New, sharedSecret, secretState)
secretState = DeriveSecret(handshakeSecret, "derived", [][]byte{})
// Handshake Traffic Secret for server
serverHandshakeSecret := DeriveSecret(handshakeSecret, "s hs traffic", [][]byte{clientHello, serverHello})
// Key and IV for server
serverWriteKey := HKDFExpandLabel(serverHandshakeSecret, "key", []byte{}, 16)
serverWriteIV := HKDFExpandLabel(serverHandshakeSecret, "iv", []byte{}, 12)
```

Generate **clientHandshakeSecret** similarly for later decryption

Certificate



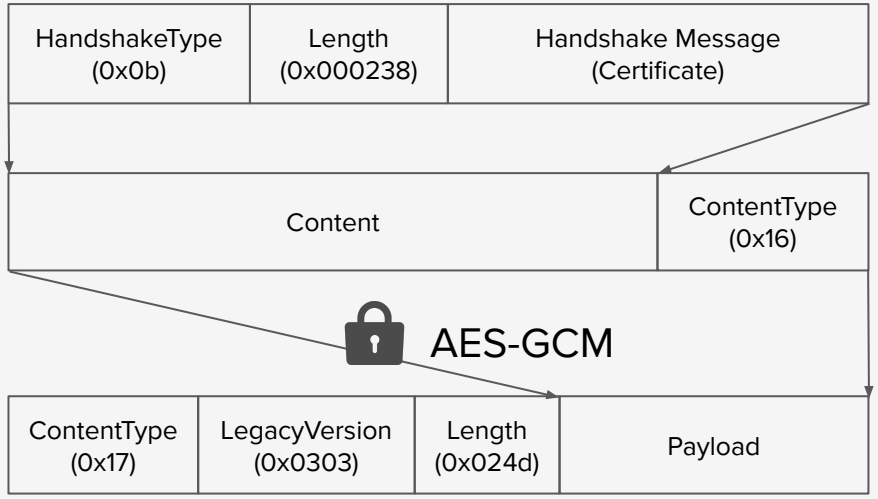
- Overview
  - Contains the server's certificate chain
  - Allows the client to authenticate the server's identity
  - Encrypted by the generated symmetric key
- Server's Certificate
  - Contains the server's public key
  - Signed by a CA to prove its authenticity
    - Used self-signed certificate this time

```
# Generate a private key
openssl ecparam -genkey -name prime256v1 -out server.key
# Create CSR
openssl req -new -key server.key -out server.csr \
  -subj "/C=JP/ST=Tokyo/L=Tokyo/O=MyOrg/OU=MyUnit/CN=localhost"
# Signing the public key
openssl req -x509 -sha256 -days 365 -key server.key \
  -in server.csr -out server.crt
```

Certificate

```
import "crypto/tls"

serverCert, _ := tls.LoadX509KeyPair(
  "server.crt", "server.key")
certificateMessage := CertificateMessage{
  CertificateRequestContext: []byte{},
  CertificateList: []CertificateEntry{
    {
      CertType: X509, // 0x01
      CertData: serverCert.Certificate[0],
    },
  },
}
// <- Wrap data structure
// <- Encrypt with AES-GCM using server Key and IV
// ...
encryptedCertificate := TLSRecord{
  ContentType:         ApplicationDataRecord, // 0x17
  LegacyRecordVersion: TLS12, // 0x0303
  Length:              uint16(len(encryptedRecord)),
  Fragment:            encryptedRecord,
}
```

```
0b 00 00 02 38 00 00 02-34 00 02 2. .. .. .. ..
.. .. .. .. .. .. .. ..-.. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. 16
```




```
17 03 03 02 4d
7d 04 95 0c 55 b5 66 b0-e6 83 46 c. .. .. .. ..
.. .. .. .. .. .. .. ..-.. .. .. .. .. .. .. ..
```

Certificate

Encryption with AES-GCM

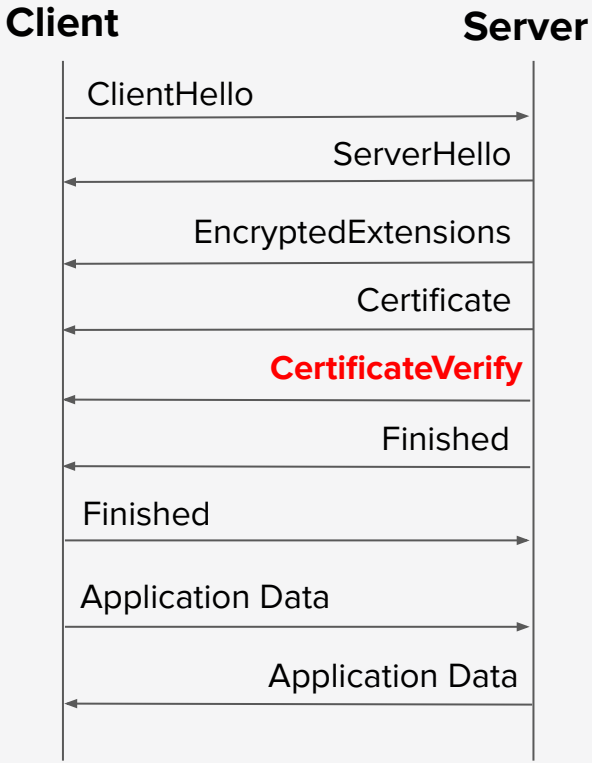
- Provides **confidentiality**, **integrity**, and **authenticity**
- Uses a shared secret key to encrypt data and generates an authentication tag
- The authentication tag ensures both integrity and authenticity
- Uses a unique nonce for each encryption operation

```
import (
  "crypto/aes"
  "crypto/cipher"
)
```

```
func EncryptTLSInnerPlaintext(key, iv []byte, tlsInnerPlainText []byte, seqNum uint64) []byte {
  block, _ := aes.NewCipher(key)
  aead, _ := cipher.NewGCM(block)
  // AEAD (Authenticated Encryption with Associated Data)
  tlsCipherTextLength := len(tlsInnerPlainText) + aesgcm.Overhead()
  additionalData := []byte{
    byte(ApplicationDataRecord),          // 0x17
    byte(TLS12 >> 8), byte(TLS12 & 0xff), // 0x0303
    byte(tlsCipherTextLength >> 8), byte(tlsCipherTextLength),
  }
  encryptedRecord := aead.Seal(nil, calculateNonce(iv, seqNum), tlsInnerPlainText, additionalData)
  return encryptedRecord
}
```

Same as the non-encrypted TLS Header

CertificateVerify



- Overview
  - Proves that the server possesses the private key corresponding to the public key in its certificate
- Signature
  - The server signs a hash of the handshake messages exchanged so far
  - The client verifies the signature using the server's public key from the certificate
- Used Algorithm
  - ecdsa_secp256r1_sha256
  - Supported Signature Algorithms Extension

CertificateVerify

```
import (
  "bytes"
  "crypto/aes"
  "crypto/ecdsa"
  "crypto/rand"
  "crypto/sha256"
  "crypto/tls"
)

signatureTarget := bytes.Repeat([]byte{0x20}, 64) // protection for chosen-prefix collision attack
signatureTarget = append(signatureTarget, []byte("TLS 1.3, server CertificateVerify")...)
signatureTarget = append(signatureTarget, 0x00) // separator
signatureTarget = append(signatureTarget, TranscriptHash([][]byte{
  clientHello,
  serverHello,
  encryptedExtensions,
  certificate,
})...)

signatureHash := sha256.Sum256(signatureTarget)

serverCert, _ := tls.LoadX509KeyPair("server.crt", "server.key")
privateKey := serverCert.PrivateKey.(*ecdsa.PrivateKey)
signature, _ := ecdsa.SignASN1(rand.Reader, privateKey, signatureHash[:])
```

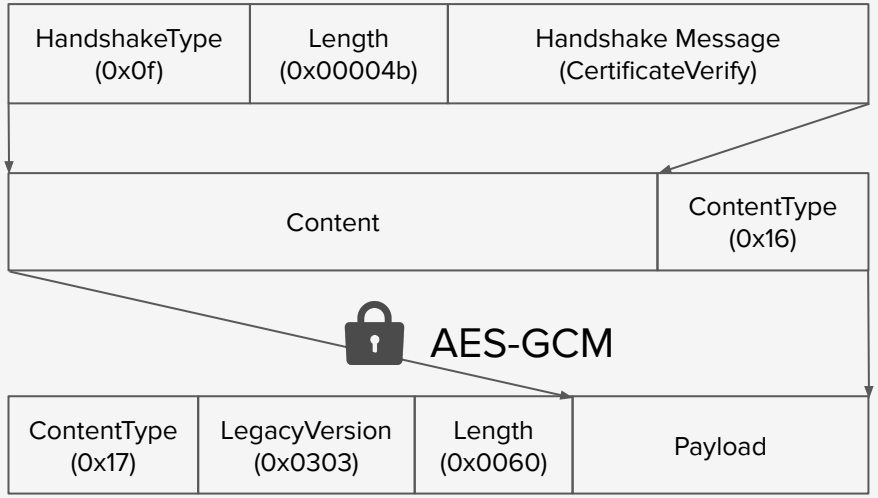
- TranscriptHash of (non-encrypted) handshake messages
- Hash by SHA-256
- Signs by ECDSA private key corresponding to the server certificate

CertificateVerify

- Construct CertificateVerify message from the signature
- Encrypt with AES-GCM
- Wrap it as Application Data TLS record

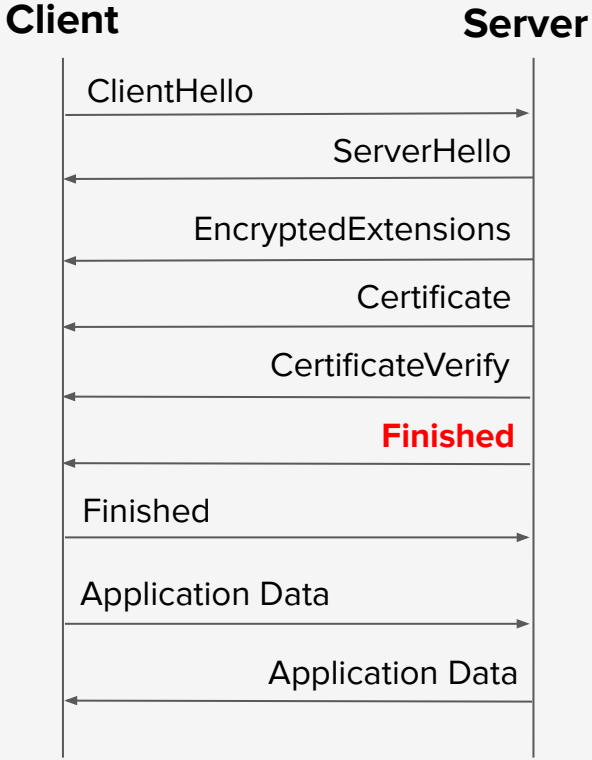
```
certificateVerifyMessage := CertificateVerifyMessage{
  Algorithm: ECDSA_SECP256R1_SHA256, // 0x0403
  Signature: signature, // 0x304502...
}
// <- Wrap data structure
// <- Encrypt with AES-GCM using server Key and IV
// ...
encryptedCertificateVerify := TLSRecord{
  ContentType:         ApplicationDataRecord, // 0x17
  LegacyRecordVersion: TLS12, // 0x0303
  Length:              uint16(len(encryptedRecord)),
  Fragment:            encryptedRecord,
}
```

```
0f 00 00 00 4b 04 03 00-47 30 45 02 .. .. .. ..
.. .. .. .. .. .. .. ..-.. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. 16
```




```
17 03 03 00 60
65 68 4d 25 cc 93 9d 27-f0 .. .. .. .. .. .. ..
.. .. .. .. .. .. .. ..-.. .. .. .. .. .. .. ..
```

Finished (Server)



- Overview
  - Verifies the integrity of the handshake messages
  - Confirms the successful completion of the handshake
  - The client verifies the server's Finished message to ensure handshake integrity
  - If the verification succeeds, the client sends its own Finished message

Finished (Server)

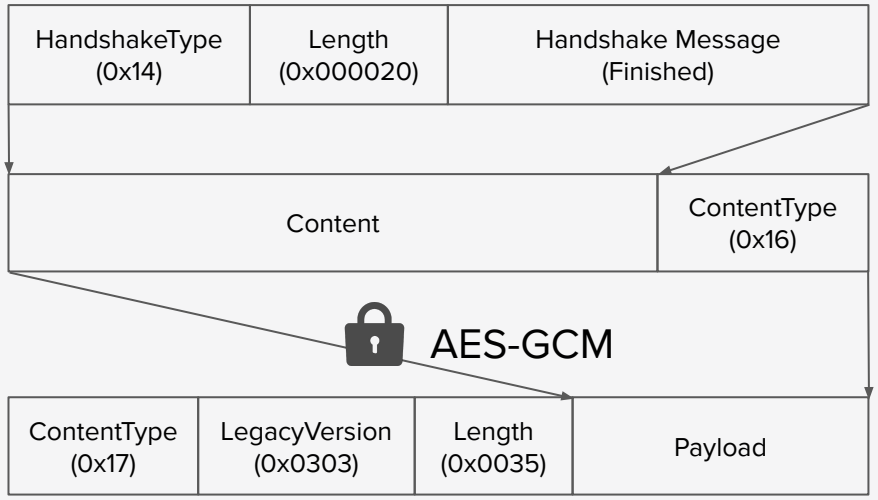
```
import (
  "crypto/hmac"
  "crypto/sha256"
)

finishedKey := HKDFExpandLabel(
  serverHandshakeTrafficSecret,
  "finished",
  []byte{},
  sha256.New().Size(),
)
h := hmac.New(sha256.New, finishedKey)
h.Write(TranscriptHash([][]byte{
  clientHello,
  serverHello,
  encryptedExtensions,
  certificate,
  certificateVerify,
}))
verifyData := h.Sum(nil)
```

- Expands a new key from the server_handshake_traffic_secret
- TranscriptHash of the handshake messages exchanged so far
- Calculates an authentication code by HMAC
  - Provides integrity and authenticity
  - Computes using a hash function and a shared secret key

Finished (Server)

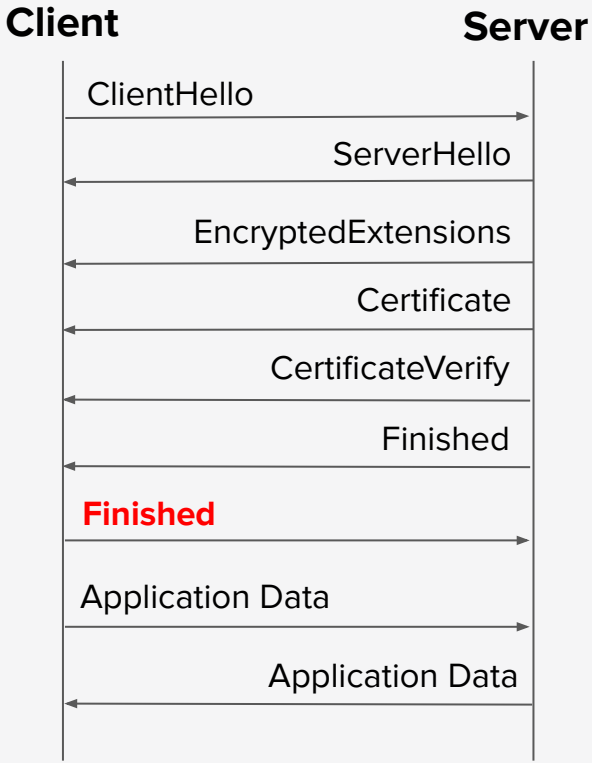
```
finished := FinishedMessage{
  VerifyData: verifyData,
}
// <- Wrap data structure
// <- Encrypt with AES-GCM using server Key and IV
// ...
encryptedFinished := TLSRecord{
  ContentType:         ApplicationDataRecord, // 0x17
  LegacyRecordVersion: TLS12, // 0x0303
  Length:              uint16(len(encryptedRecord)),
  Fragment:            encryptedRecord,
}
```

```
14 00 00 20 6f 75 07 88-40 b5 5f .. .. .. .. ..
.. .. .. .. .. .. .. ..-.. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. 16
```




```
17 03 03 00 35
54 97 df 0d c7 d9 23 f7-5e f0 .. .. .. .. .. .. ..
.. .. .. .. .. .. .. ..-.. .. .. .. .. .. .. ..
```

Finished (Client)



- Overview
  - Verifies the integrity of the handshake messages
  - Confirms the successful completion of the handshake
  - Sent by the client after verifying the server's Finished message
  - The server verifies the client's Finished message to ensure handshake integrity
- Decrypted first with Client handshake Key and IV

```
import (
  "crypto/aes"
  "crypto/cipher"
)
block, _ := aes.NewCipher(clientWriteKey)
aesgcm, _ := cipher.NewGCM(block)
decryped := aesgcm.Open(nil, calculateNonce(clientWriteIV, seqNum),
  encryptedTLSInnerText,
  []byte{0x17, 0x03, 0x03, 0x00, 0x35}, // TLS Record header
)
```

Finished (Client)

Received data

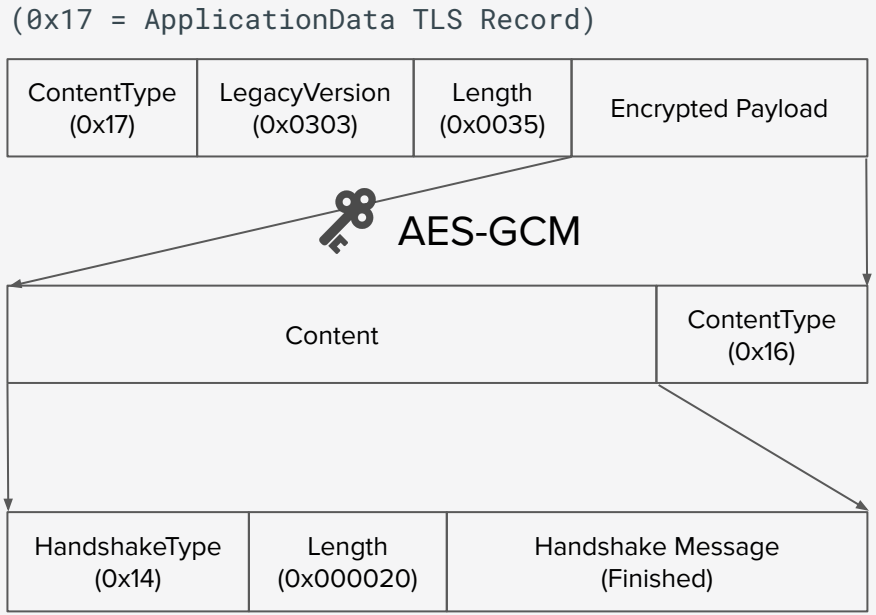
```
17 03 03 00 35 12 1e 80-f5 51 a4 bd 24 c2 6c 0f
29 80 60 cf c6 a1 1b 15-31 54 51 1a 8a b8 78 9e
0a 42 59 5d 87 97 9c a2-a2 b6 61 f0 ed 05 52 9e
5a 02 b2 d8 24 82 ac 62-75 2b
```

Decryption with AES-GCM (aesgcm.Open)
using the Client handshake Key and IV

```
14 00 00 20 aa 6c 54 15-aa 69 d9 d1 09 9a 81 9c
8e 93 02 22 53 38 79 95-9d 0f 00 24 35 2a a9 36
c3 fa ca e5 16
```

Extract the HMAC value (Finished payload)
calculated in the client side

```
aa 6c 54 15 aa 69 d9 d1-09 9a 81 9c 8e 93 02 22
53 38 79 95 9d 0f 00 24-35 2a a9 36 c3 fa ca e5
```



=> Verifies that this value coincides with the HMAC value calculated in the server side

Finished (Client)

```
import (
  "crypto/hmac"
  "crypto/sha256"
)
finishedKey := HKDFExpandLabel(
  clientHandshakeTrafficSecret,
  "finished",
  []byte{},
  sha256.New().Size(),
)
h := hmac.New(sha256.New, finishedKey)
h.Write(TranscriptHash([][]byte{
  clientHello,
  serverHello,
  encryptedExtensions,
  certificate,
  certificateVerify,
  serverFinished,
}))
verifyData := h.Sum(nil)
```

Compare ⇔

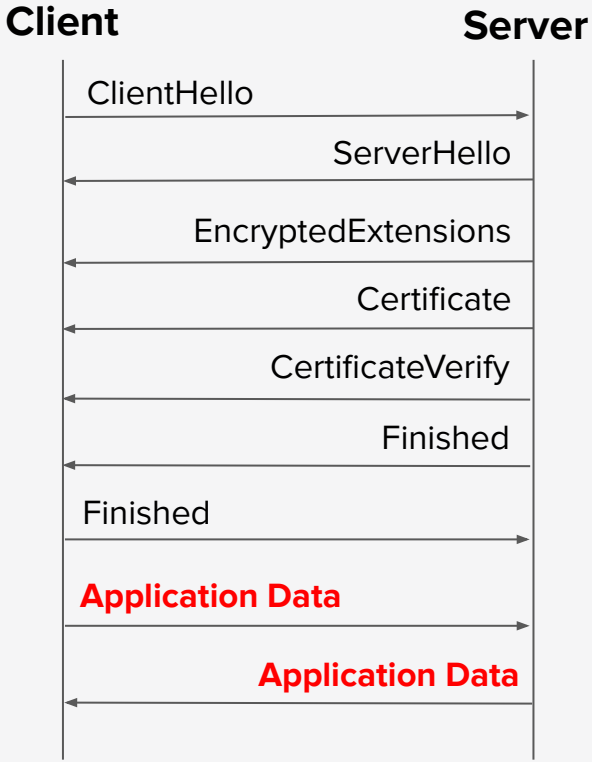
- Similar calculation as the server Finished
  - Uses client_handshake_traffic_secret
  - Includes the server Finished message in TranscriptHash
- Comparison with the HMAC from the client Finished
  - Confirms the integrity of the handshake messages

TLS handshake completed! 🎉

HMAC value calculated in the client

```
aa 6c 54 15 aa 69 d9 d1-09 9a 81 9c 8e 93 02 22
53 38 79 95 9d 0f 00 24-35 2a a9 36 c3 fa ca e5
```

After TLS Handshake



- Derive different secrets and keys for encrypting/decrypting application data
  - Master secret
  - Client application traffic secret / Server application traffic secret
- HTTP data is encrypted with the derived keys

Summary

Summary

- Explored the full handshake process in TLS 1.3. We saw ..
  - Key Agreement and Exchange: ECDHE
  - Key Schedule: Extraction and Expansion with HKDF
  - Symmetric Encryption: AES-GCM (AEAD)
  - Authentication: Digital signatures and certificates (ECDSA)
  - Handshake integrity: HMAC
- Implementation with Go
  - “crypto/*” packages
- Realized the important security properties
  - **Confidentiality**, **Integrity**, and **Authenticity**
- There are more topics
  - Session resumption, 0-RTT, and more

Summary

References

- RFC 8446: https://datatracker.ietf.org/doc/html/rfc8446
- Ivan Ristic, “Bulletproof TLS and PKI, Second Edition”, Feisty Duck
  （邦訳: 齋藤孝道 監訳,『プロフェッショナルTLS&PKI 改題第2版』, ラムダノート）
- 結城浩,『暗号技術入門 第3版』, SBクリエイティブ
- 古城 隆, 松尾 卓幸, 宮崎 秀樹, 須賀 葉子,『徹底解剖 TLS 1.3』, 翔泳社
- https://zenn.dev/satoken/articles/golang-tls1_3 (golangで作るTLS1.3プロトコル)