

# TLS 1.3

CHITRANG SRIVASTAVA

# Agenda

- Recap of RSA, DH & Elliptic Curve
- Recap of TLS 1.2
- Introduction to TLS 1.3
- Authenticated Encryption & Additional Data (AEAD)

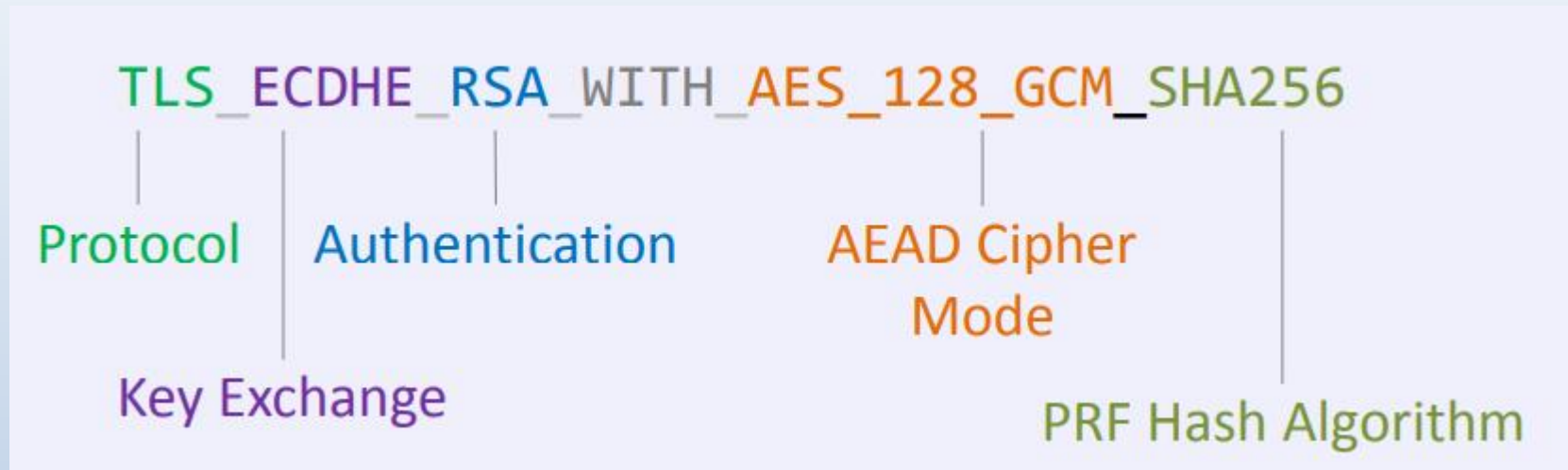
# TLS/SSL Brief History

Developed originally by Netscape

Protocol	Published	Status
SSL 1.0	Unpublished	Unpublished
SSL 2.0	1995	Deprecated in 2011 ( <a href="#">RFC 6176</a> )
SSL 3.0	1996	Deprecated in 2015 ( <a href="#">RFC 7568</a> )
TLS 1.0	1999	Deprecation planned in 2020 <sup>[11]</sup>
TLS 1.1	2006	Deprecation planned in 2020 <sup>[11]</sup>
TLS 1.2	2008 (RFC 5246)	
TLS 1.3	2018 (RFC 8446)	

*Source: Wikipedia*

# CipherSuite

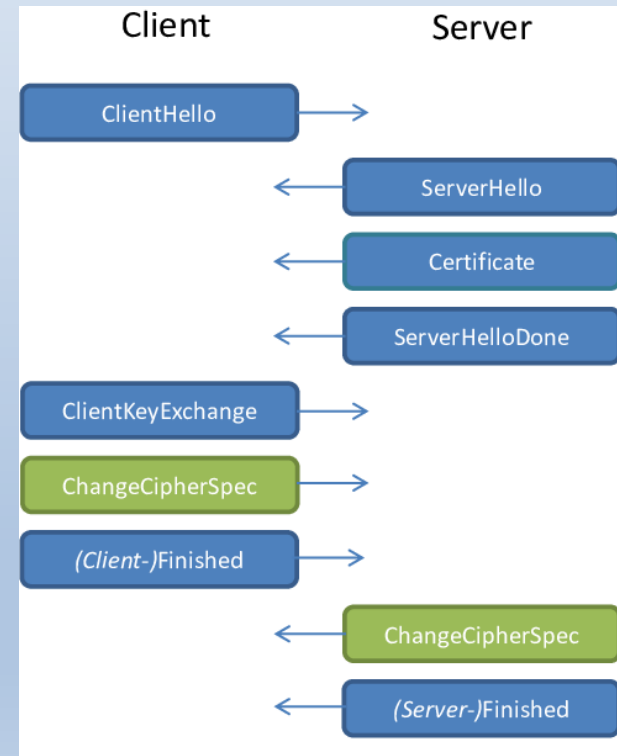


`TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256`

When [GCM](#) is used, there is no per-record HMAC; integrity is **obtained from the GCM mode itself. So the hash function specified in the cipher suite is used only for the PRF and other** handshake-related usages. Source: [StackExchange](#)

# TLS 1.2 Handshake with RSA

- $\text{Pre\_Master\_Secret} = \text{Random 48 bytes generated and encrypted using Server's public key.}$
- $\text{Master Secret} = \text{PRF(PMS, Client.Random, Server.Random, "master secret")}$
- Many keys derived like IV, Read-Write session key, read-write MAC keys,  $\text{PRF(MS, "key expansion")}$ .
- ChangeCipherSec
- Client Finished, hash of all handshake message.
- 2 Round Trips.



# PRF

- **PRF(“secret” + “label” + “non-secret”)**
- `master_secret = PRF(pre_master_secret, "master secret", ClientHello.random + ServerHello.random)`
- `PRF(SecurityParameters.master_secret, "key expansion", SecurityParameters.server_random + SecurityParameters.client_random);`
- `PRF(master_secret, "client finished", Hash(handshake_messages))`

`P_hash(secret, seed) = HMAC_hash(secret, A(1) + seed) +  
                                  HMAC_hash(secret, A(2) + seed) +  
                                  HMAC_hash(secret, A(3) + seed) + ...`

where + indicates concatenation.

A() is defined as:

`A(0) = seed`

`A(i) = HMAC_hash(secret, A(i-1))`

P\_hash can be iterated as many times as necessary to produce the required quantity of data. [Reference](#)

# SHA (Secure Hash Algorithm)

- **Pre-image resistance.**

Given a hash value  $h$  it should be difficult to find any message  $m$  such that  $h = \text{hash}(m)$ .

- **Second pre-image resistance.**

Given an input  $m_1$ , it should be difficult to find a different input  $m_2$  such that  $\text{hash}(m_1) = \text{hash}(m_2)$ .

- **Collision resistance.**

- It should be difficult to find two different messages  $m_1$  and  $m_2$  such that  $\text{hash}(m_1) = \text{hash}(m_2)$ .

- SHA1(160 bits), SHA2(224/256 bits), SHA384, SHA-3

- <https://www.youtube.com/watch?v=E4FL9Tv-X-k>

- <https://8gwifi.org/MessageDigest.jsp>

# Message Authentication Code (MAC)

- MAC is a tag of data computed with a key
- HMAC is one such MAC algorithm which is a recipe for turning hash functions into MAC i.e. HMAC-SHA256

This definition is taken from [RFC 2104](#):

$$\text{HMAC}(K, m) = \text{H} \left( (K' \oplus \text{opad}) \parallel \text{H} \left( (K' \oplus \text{ipad}) \parallel m \right) \right)$$
$$K' = \begin{cases} \text{H}(K) & K \text{ is larger than block size} \\ K & \text{otherwise} \end{cases}$$

where

$\text{H}$  is a cryptographic hash function

$m$  is the message to be authenticated

$K$  is the secret key

$K'$  is a block-sized key derived from the secret key,  $K$ ; either by padding to the right with 0s up to the block size, or by hashing down to less than the block size first and then padding to the right with zeros

$\parallel$  denotes [concatenation](#)

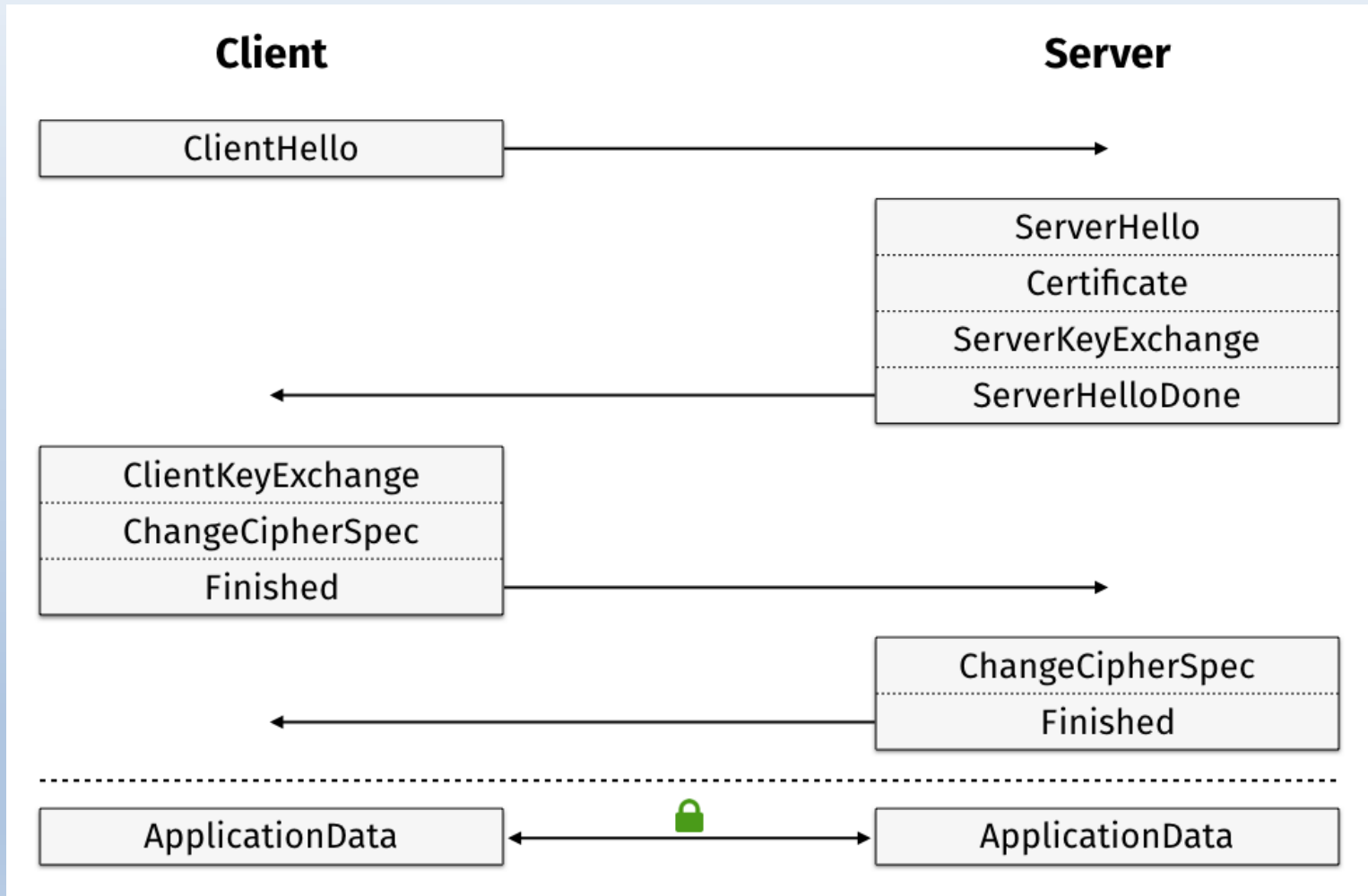
$\oplus$  denotes bitwise [exclusive or](#) (XOR)

$\text{opad}$  is the block-sized outer padding, consisting of repeated bytes valued 0x5c

$\text{ipad}$  is the block-sized inner padding, consisting of repeated bytes valued 0x36



# TLS 1.2 Handshake with DH



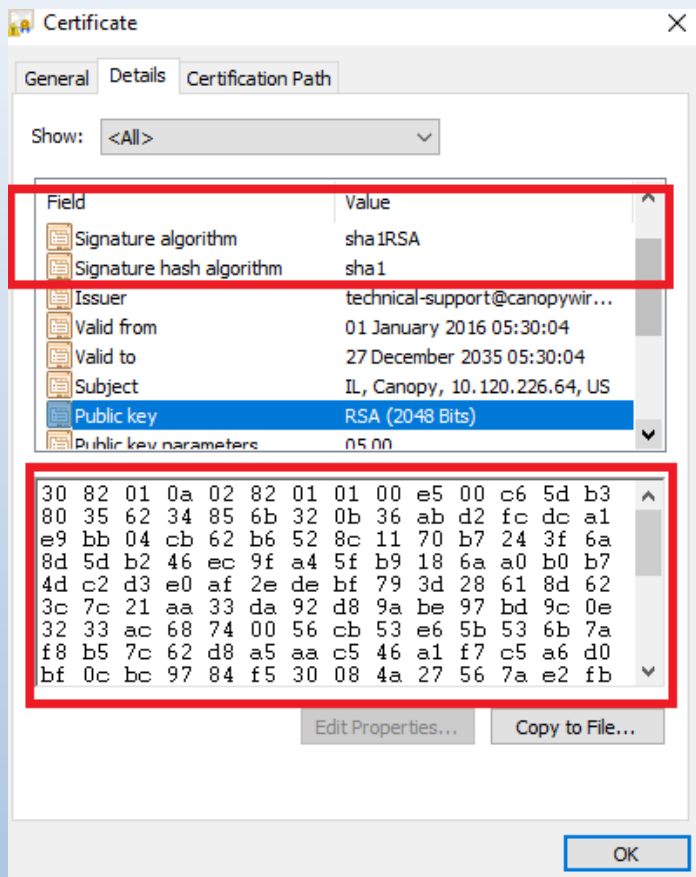
# Generating Keys

- $\text{Master\_Secret} = \text{PRF}(\text{Client.Random} + \text{Server.Random} + \text{Pre-Master-Secret} + \text{"master secret"})$
- $\text{Key\_Expansion} = \text{PRF}(\text{Client.Random} + \text{Server.Random} + \text{Master-Secret} + \text{"key expansion"})$

```
client_write_MAC_secret[] server_write_MAC_secret[]  
client_write_key[] server_write_key[]  
client_write_IV[] server_write_IV[]
```

Messages from **client to server** are **encrypted** with the **client write key**, and the **server** uses the **client write key** to **decrypt** them. Messages from server to client are encrypted with the server write key, and the client uses the server write key to decrypt them.

# Certificate



Signed by CA private key after taking  
SHA1 of Server certificate contents.

<https://shattered.it/>

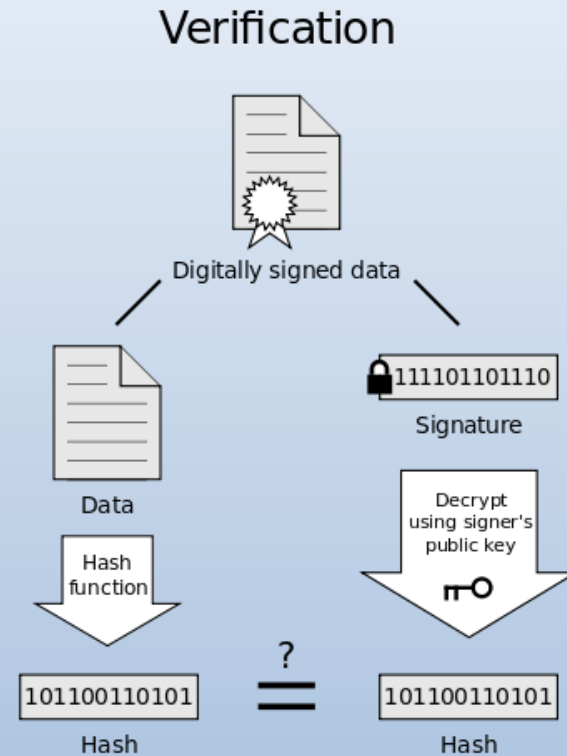
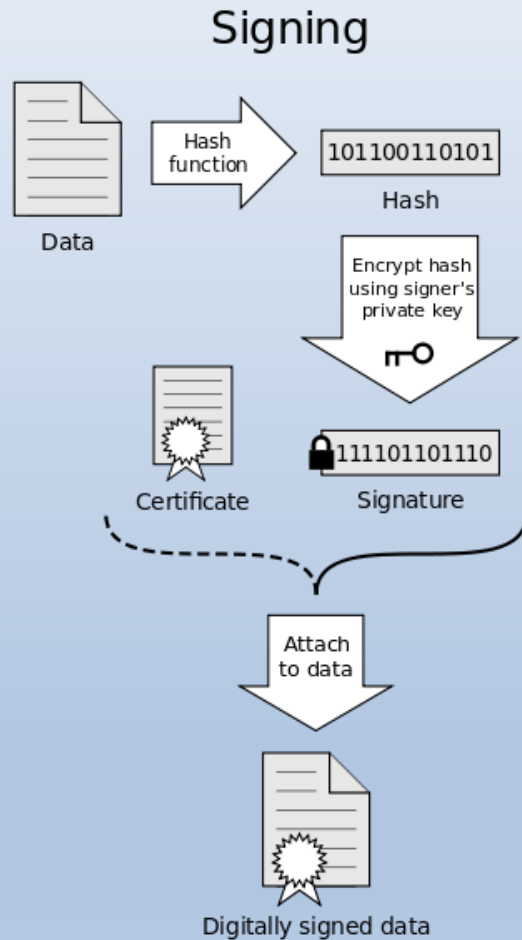
```
certificate:
Data:
  Version: 3 (0x2)
  Serial Number:
    18:1f:4e:c6:be:3e:44:14:f0:6d:51:ae:40:e4:ee:7f:c5:ee
  Signature Algorithm: sha1WithRSAEncryption
  Issuer: C = US, ST = Illinois, O = "Motorola Solutions, I
  Validity
    Not Before: May 31 11:13:08 2019 GMT
    Not After : Apr 20 04:44:52 2003 GMT
  Subject: C = US, CN = 10.110.246.130, O = Canopy, ST = IL
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public-Key: (2048 bit)
    Modulus:
      00:cf:15:ab:42:43:17:b3:39:7c:25:ea:ce:b2:d6:
      ad:b5:a0:4e:2f:47:44:0d:d9:c4:09:ca:e0:54:9d:
      15:6c:b4:d9:3b:00:63:e9:e4:32:12:69:e8:ed:3a:
      8c:62:e4:7f:c9:1f:8f:55:fc:b5:eb:d9:4a:59:e9:
      ad:11:07:a6:0b:c0:ec:25:de:1d:df:5c:c8:13:a8:
      08:ed:22:15:af:b4:44:4c:07:43:c4:3c:ee:8f:ff:
      3b:ee:02:89:96:84:9d:2b:28:0f:20:ae:f1:e4:c8:
      33:4f:ca:49:31:d9:31:22:16:8c:3c:3f:90:2a:4b:
      12:1b:74:91:db:71:b0:94:6e:e7:ea:90:44:14:3f:
      79:37:a8:a0:db:a9:50:a7:ab:7a:9a:c9:fb:f0:cb:
      43:c4:7d:9e:d8:8a:ef:54:dd:c2:78:23:5b:6d:c8:
      b9:0e:00:c8:67:ee:96:21:c8:c2:95:4c:b6:97:b1:
      8b:b1:64:7b:50:cb:53:40:2f:32:3e:52:f0:89:c0:
      e7:28:7f:65:33:b8:9e:15:0b:4d:ec:eb:4c:b7:1d:
      aa:d5:40:1d:55:0c:99:c8:06:ab:b9:7c:49:de:81:
      12:e3:96:72:1b:76:fb:a3:4d:e7:28:7d:c0:b0:b6:
      42:bf:ae:63:4e:33:96:26:1c:a9:cb:54:84:6d:b0:
      d0:77
    Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Basic Constraints:
      CA:FALSE
    Netscape Comment:
      "Canopy generated Certificate"
    X509v3 Key Usage:
      Digital Signature, Non Repudiation, Key Encipherm
    X509v3 Subject Alternative Name:
      TP Address:10.110.246.130, URI:https://10.110.246
  Signature Algorithm: sha1WithRSAEncryption
    3d:46:0d:00:2b:cb:7b:65:80:bd:35:a1:47:72:c9:41:99:b7:
    b9:f4:dd:47:ce:e4:cb:09:24:4b:e8:8d:20:5a:f6:ec:43:a9:
    75:87:95:6c:a1:09:2d:2c:e5:13:24:87:9e:33:41:46:3f:a6:
    cf:e2:80:46:a9:58:20:7a:f8:2c:5c:55:35:58:f9:2a:fc:2b:
    a3:cb:d5:69:af:64:65:6a:01:e6:d1:3f:01:ac:8e:e3:bc:8a:
    f3:61:52:47:f2:af:a2:5b:a8:4b:62:e3:8f:5c:86:59:7b:f3:
    46:15:aa:9d:d2:e5:ea:8d:00:b5:ff:4c:96:2a:02:9d:63:91:
```

# Certificate Signing Request(CSR)

- Generate a key\_pair(private, public)
- Fill up details like CN/C/ST/L/O/OU etc.
- Take a Hash of above data , Sign everything with your private key and append this data as “Signature Algorithm”. Signing the CSR proves ownership of the private key corresponding to the public key in the CSR. 1

```
csr100@IN01-760Y0G2:/mnt/c/Users/csr100/Downloads/Canopy_certs/2$openssl req -text -noout -verify -in myECC.csr
verify OK
Certificate Request:
Data:
  Version: 1 (0x0)
  Subject: C = IN, ST = KA, L = BLR, O = Cambium Networks Inc, OU = Cambium BLR, CN = Chitrang Srivastava ECC secp384r1
  Subject Public Key Info:
    Public Key Algorithm: id-ecPublicKey
    Public-Key: (384 bit)
    pub:
      04:2c:a5:10:8f:b7:75:75:88:d4:fc:22:2b:41:ca:
      7c:31:1c:82:39:7e:ef:e7:54:0b:75:0b:c1:7b:c7:
      2a:5a:c9:d9:3e:ed:23:46:28:64:87:5d:69:0b:2b:
      a4:1a:75:18:ac:d6:8d:43:44:b0:f8:31:3c:59:ac:
      eb:4b:1c:23:e5:4f:be:bd:56:e3:a9:7a:05:a2:e3:
      b3:66:a7:24:8b:3d:5d:b2:c3:40:01:9c:f7:54:00:
      3d:96:1b:cd:9e:6b:fe
    ASN1 OID: secp384r1
    NIST CURVE: P-384
  Attributes:
    a0:00
  Signature Algorithm: ecdsa-with-SHA256
    30:65:02:31:00:db:80:3c:93:ec:56:d3:21:82:ed:4a:fd:f0:
    8d:41:78:eb:08:eb:22:c1:6a:e3:d8:f6:5a:e5:43:a4:b8:f6:
    6a:03:00:03:0c:ba:7d:bf:2d:44:58:03:9b:ce:70:c2:20:02:
    30:0f:9e:57:b2:db:fa:1c:aa:f2:a5:b2:fb:ac:58:7c:74:16:
    b8:99:45:49:81:3f:9b:81:f2:15:41:0f:b5:b9:69:5a:80:cf:
    a2:4c:48:06:34:6d:f6:c3:57:23:82:f6:df
```

# Signing



If the hashes are equal, the signature is valid.

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 1234605616436508555 (0x112233445566778b)

Signature Algorithm: sha256WithRSAEncryption

Issuer: C = IN, ST = KA, O = Cambium Networks Inc, OU = Cambium BLR, CN = PMP 450 BLR

Validity

Not Before: Sep 3 10:32:43 2019 GMT

Not After : Aug 31 10:32:43 2029 GMT

Subject: C = IN, ST = KA, O = Cambium Networks Inc, OU = Cambium BLR, CN = Chitrag Srivastava ECC secp384r1

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (384 bit)

pub:

04:2c:a5:10:8f:b7:75:75:88:d4:fc:22:2b:41:ca:  
7c:31:1c:82:39:7e:ef:e7:54:0b:75:0b:c1:7b:c7:  
2a:5a:c9:d9:3e:ed:23:46:28:64:87:5d:69:0b:2b:  
a4:1a:75:18:ac:d6:8d:43:44:b0:f8:31:3c:59:ac:  
eb:4b:1c:23:e5:4f:be:bd:56:e3:a9:7a:05:a2:e3:  
b3:66:a7:24:8b:3d:5d:b2:c3:40:01:9c:f7:54:00:  
3d:96:1b:cd:9e:6b:fe

ASN1 OID: secp384r1

NIST CURVE: P-384

X509v3 extensions:

X509v3 Basic Constraints:

CA:FALSE

Netscape Comment:

OpenSSL Generated Certificate

X509v3 Subject Key Identifier:

A4:96:0A:E1:4E:30:AC:1D:37:14:EB:5D:C0:A8:44:6E:C3:63:5A:C5

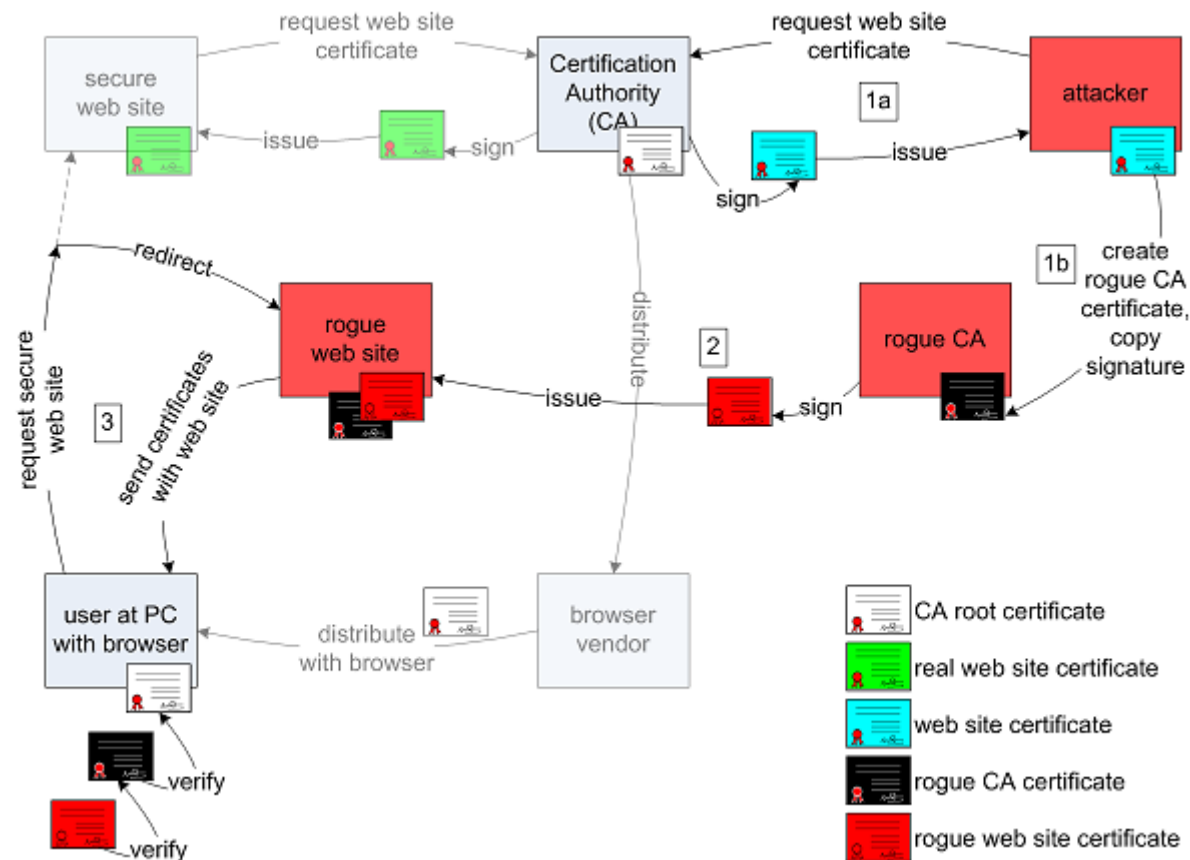
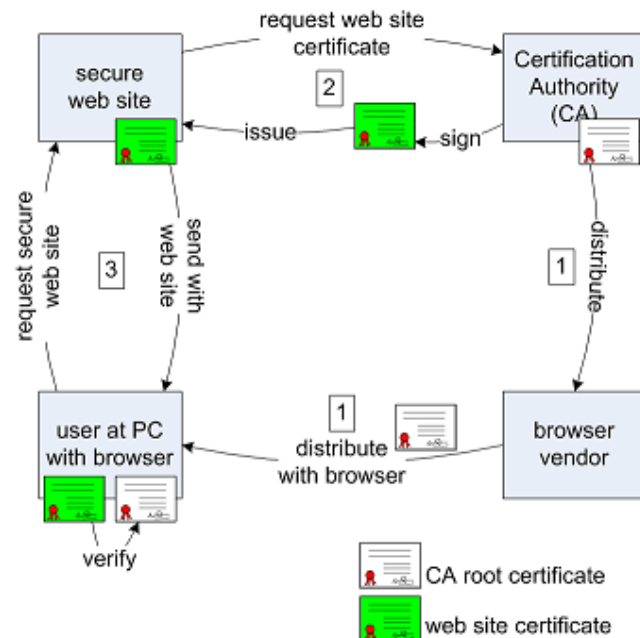
X509v3 Authority Key Identifier:

keyid:EB:54:31:9C:00:9A:E3:48:BB:42:51:BC:14:38:3F:75:47:54:03:87

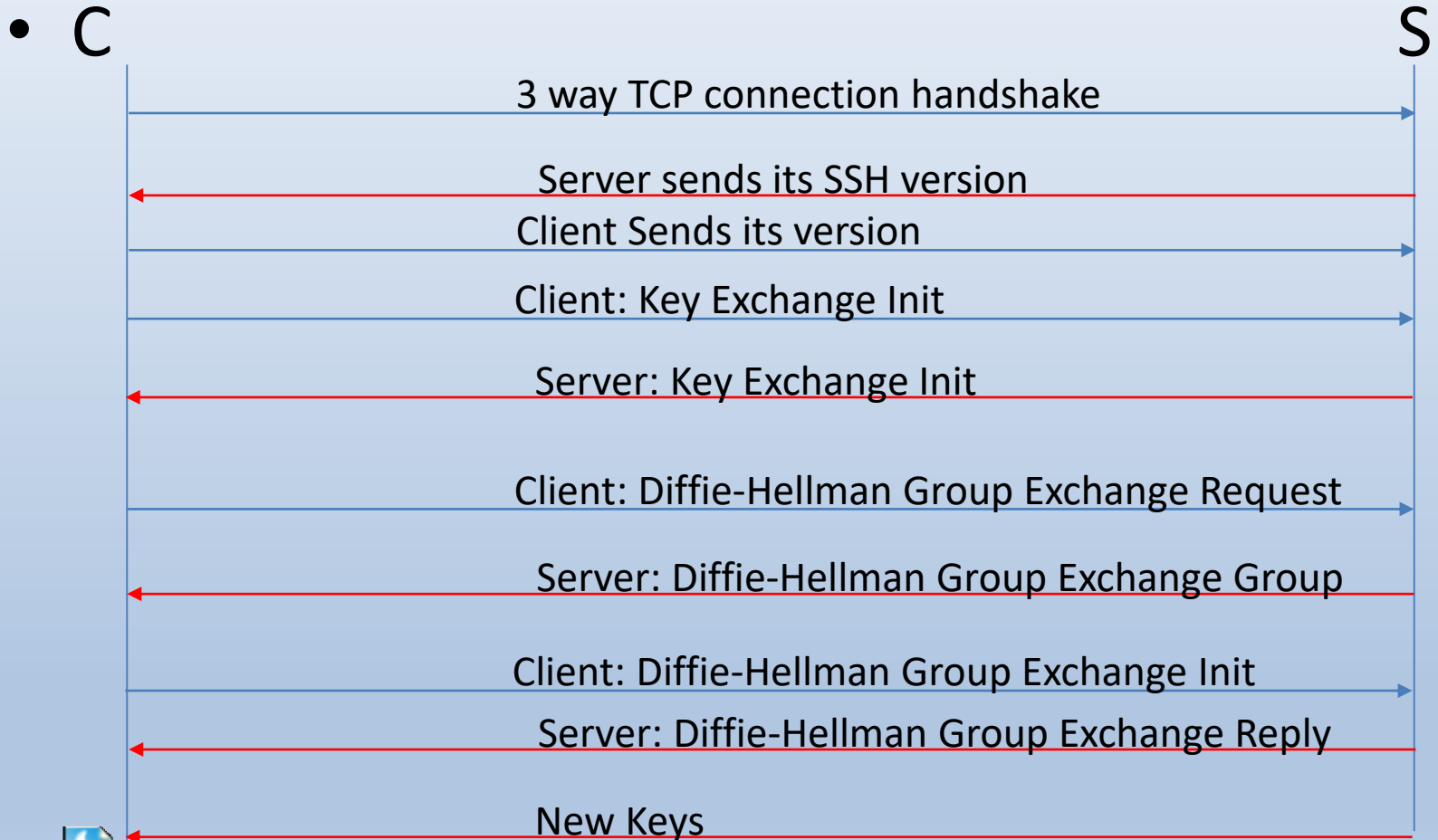
Signature Algorithm: sha256WithRSAEncryption

72:23:67:81:9b:96:35:12:97:f3:30:af:73:7e:99:7d:d4:ec:  
ad:fb:41:d7:60:68:9d:06:2b:8e:b5:c3:c4:d5:74:40:cb:f7:  
b8:78:0a:3c:cc:0f:ea:8c:54:2b:22:0c:36:72:a2:a5:16:25:  
f7:dc:d0:74:28:b9:05:50:57:70:3c:9a:80:30:be:32:79:2b:  
58:13:cc:f3:52:ed:d2:2a:be:3c:84:27:21:cf:5b:90:1e:c6:  
33:a1:54:11:3a:87:49:6e:94:b9:da:18:69:12:30:c9:df:bc:  
8a:1b:de:22:6d:72:08:9e:6d:39:9a:09:2c:27:35:1f:eb:c7:  
ee:f1:87:7b:ec:d4:59:3e:11:6f:04:1b:1f:e5:41:16:6a:cc:  
79:7a:bf:2a:6e:82:53:41:f6:72:ec:1e:c7:ac:08:ce:14:0b:  
21:c4:17:0a:00:89:cb:df:7d:44:42:aa:bf:d7:9d:e3:3d:a3:  
87:3e:78:2c:e6:7a:f5:f3:b2:f4:fd:2c:a3:d5:39:83:5a:50:

# SHA-1 Collision Attack



# How SSH works

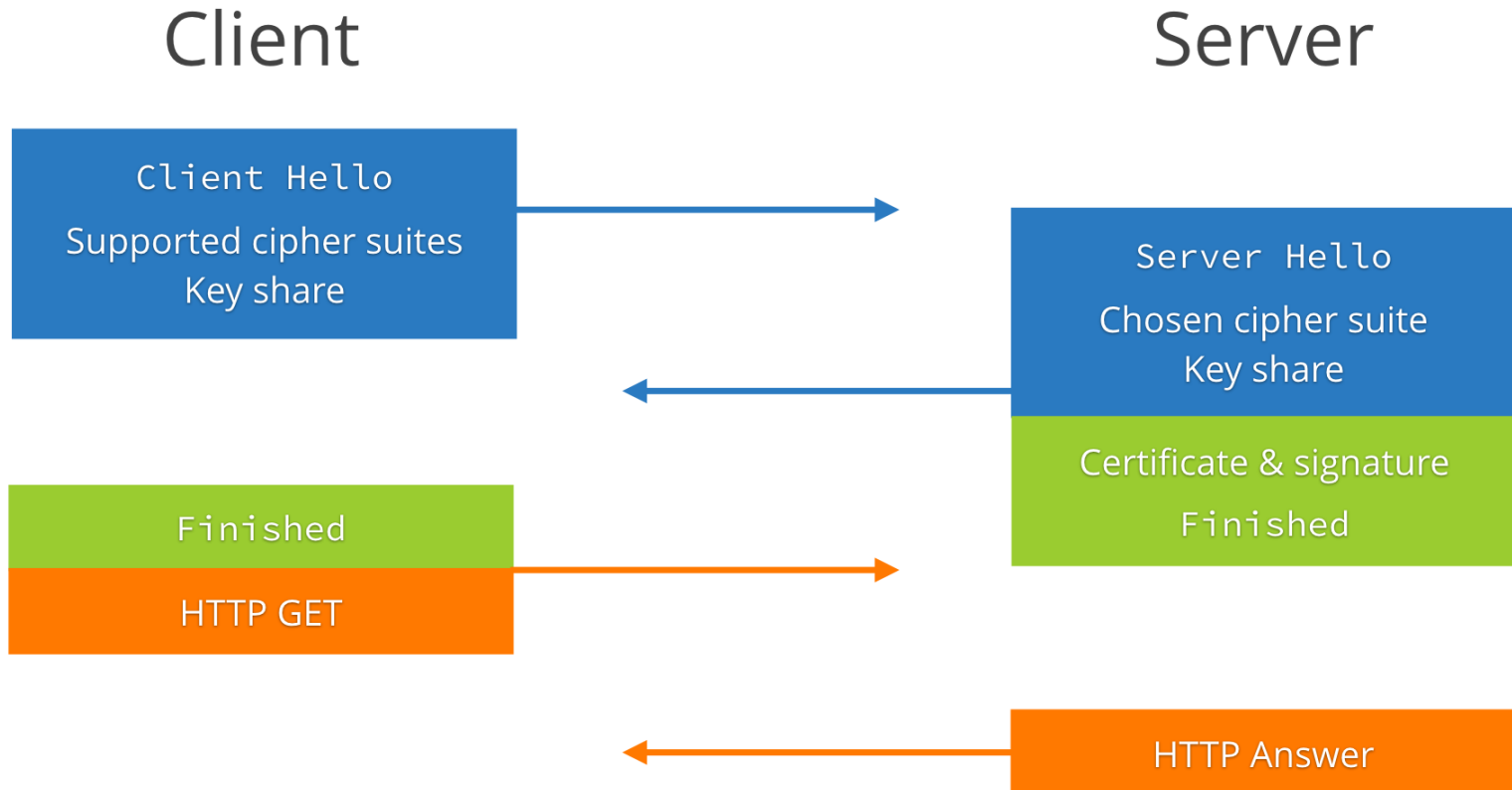


ssh-server-client.pcapng

[RFC 4253](#)



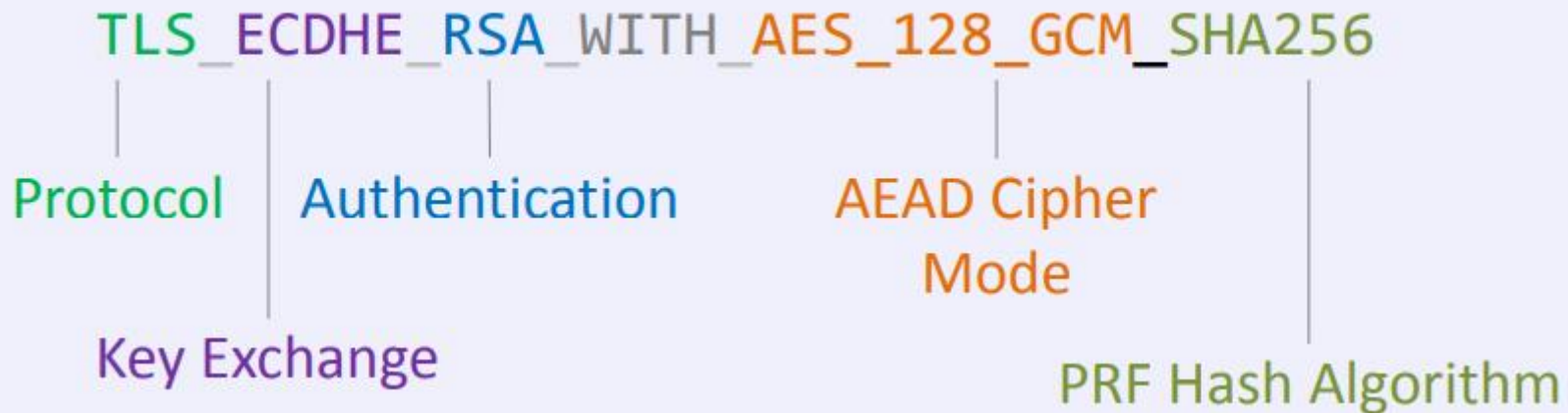
# TLS 1.3 Handshake



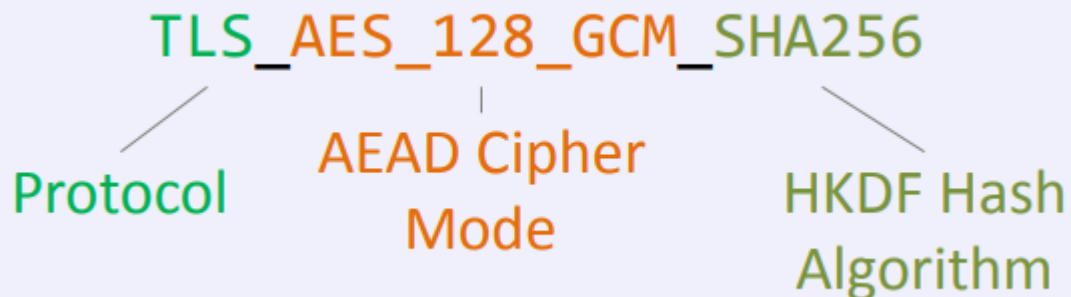
# Major Difference between TLS 1.2 & TLS 1.3

- |  |
|--|
| • Simple cipher suites (just 5 of them ) specifying encryption algorithm & HKDF.   |
| • Perfect Forward Secrecy is mandatory.  |
| • Certificate is sent encrypted in <b>ServerHello</b> as opposed to plain text in TLS 1.2                                      |
| • 1-RTT, 0-RTT as opposed to 2-RTT or more in TLS 1.2  |
| • PRF is replaced by HKDF.   |
| • Only support AEAD cipher suites. AES in GCM mode and Chach20-Poly1305. Encrypt-then-MAC, Mac-then-Encrypt is all phased out. |
| • Compression, Renegotiation is removed  |

# CipherSuites Difference



## TLS 1.3



# Downgrade Attack

The diagram illustrates a downgrade attack on a TLS connection. It shows two browser security warnings side-by-side. The left warning, for a standard connection, shows 'Identity verified' and 'The connection uses TLS 1.2'. The right warning, after an attack, shows 'Identity verified' but 'The connection uses SSL 3.0'. A red circle highlights a message at the bottom of the right warning: 'The connection had to be retried using an older version of the TLS or SSL protocol. This typically means that the server is using very old software and may have other security issues.' A green bracket on the right points to the top warning with the text 'Standard connection to Facebook.com using TLS 1.2'. A red bracket on the left points to the bottom warning with the text 'Attack downgrades connection to use SSL v3'.

**Standard connection to Facebook.com using TLS 1.2**

**Attack downgrades connection to use SSL v3**

**Standard connection to Facebook.com using TLS 1.2**

**Attack downgrades connection to use SSL v3**

The connection had to be retried using an older version of the TLS or SSL protocol. This typically means that the server is using very old software and may have other security issues.

# Downgrade Attack

- ClientHello(CH) is dropped and then client send CH with lower SSL version of Cipher Suites.
- Sever Replies TLS 1.3 ServerRandom last 8 bytes has DOWNGRD01
- ServerHello is compromised ;SCSV(Signalling Cipher Supported Version)

its presence in the Client Hello message serves as

a backwards-compatible signal from the client to the server.

For backard compatibility, ClieHello version remain 1.2 instead a new “supported\_version” extension is added which list 1.3 and hence TLS 1.3 server knows that client wants 1.3, TLS 1.2 will simply ignore it and do TLS 1.2 More use [cases](#) and [here](#)

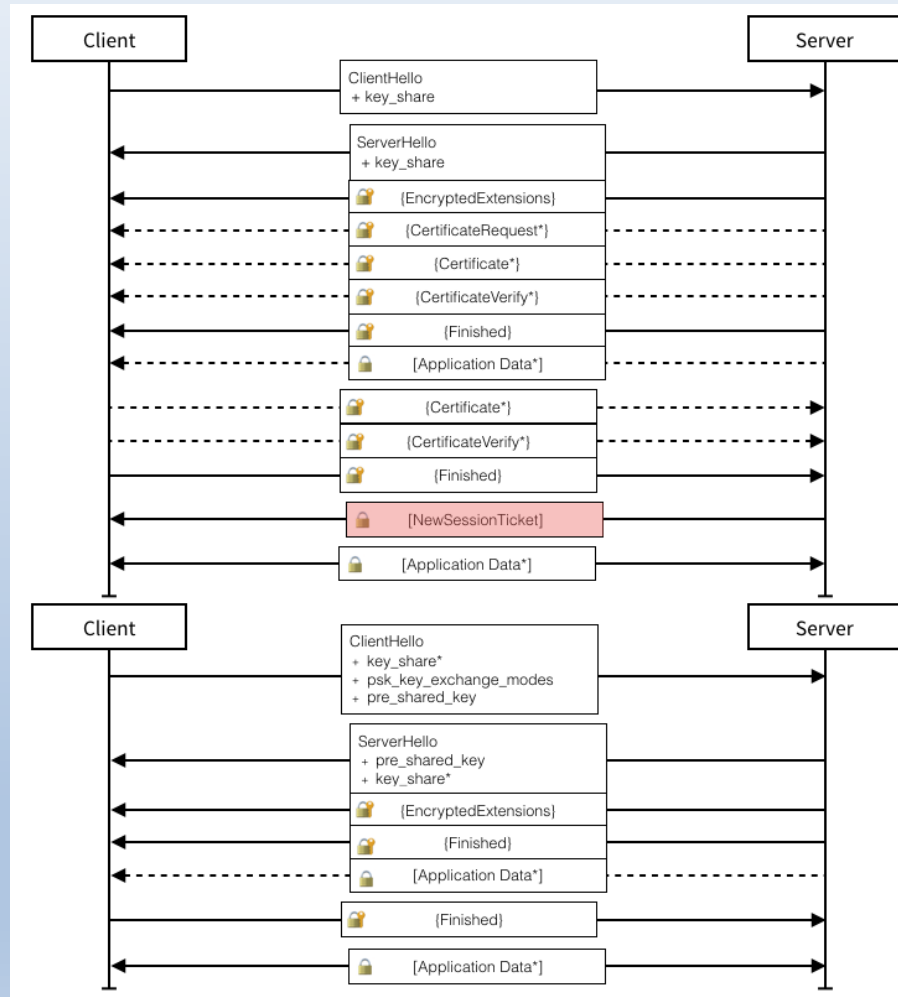
# HKDF

- Client\_public\_key + server\_private\_key + SHA(ClientHello+ServerHello) is fed to HKDF
- Extract & Expand

# 0-RTT

- Opens up risk for replay [attack](#).
- During 1<sup>st</sup> session establishment, server gives client 'Session Ticket' which client uses in subsequent connection.
- The client also sends a key share, so that client and server can switch to a new fresh key for the actual HTTP response and the rest of the connection.

# TLS 1.3 Session Resumption



Reference:  
<https://www.davidwong.fr/tls13/>



# TLS Proxy

93	1.413892000	10.42.1.4	74.125.132.138	TLSv1	238	Client Hello
95	1.442269000	74.125.132.138	10.42.1.4	TLSv1	1444	Server Hello
97	1.442966000	74.125.132.138	10.42.1.4	TLSv1	1187	Certificate, Server
99	1.469835000	10.42.1.4	74.125.132.138	TLSv1	224	Client Key Exchange
100	1.499698000	74.125.132.138	10.42.1.4	TLSv1	292	New Session Ticket,

Compression method: (1 method)

Extensions Length: 50

▼ Extension: server\_name

Type: server\_name (0x0000)

Length: 20

▼ Server Name Indication extension

Server Name list length: 18

Server Name Type: host\_name (0)

Server Name length: 15

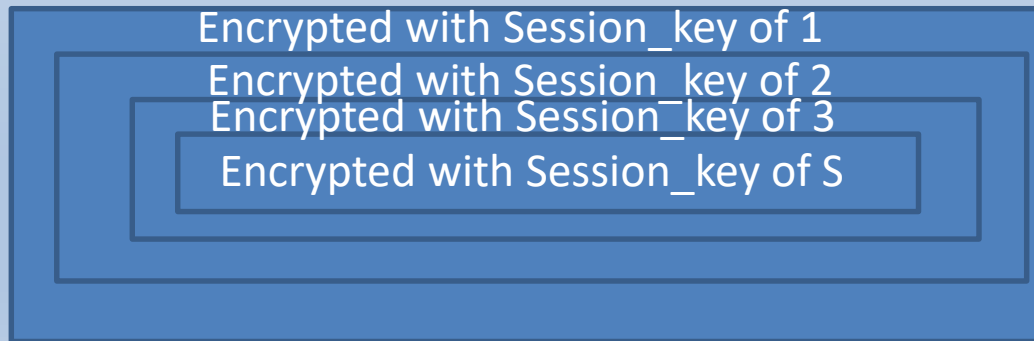
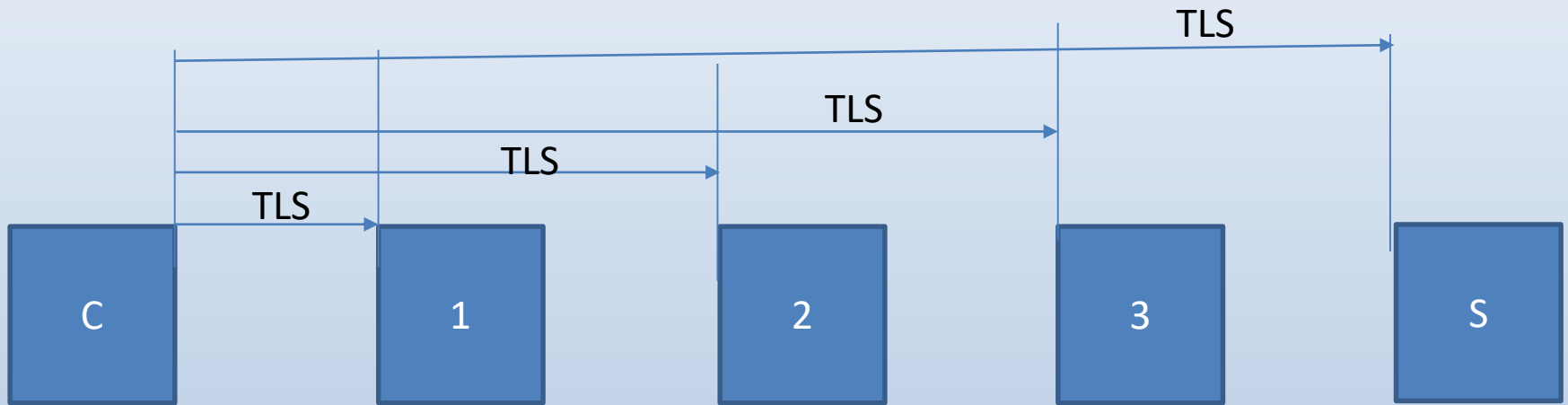
Server Name: plus.google.com

- <https://docs.mitmproxy.org/stable/concepts-howmitmproxyworks/>
- <https://arxiv.org/pdf/1407.7146.pdf>

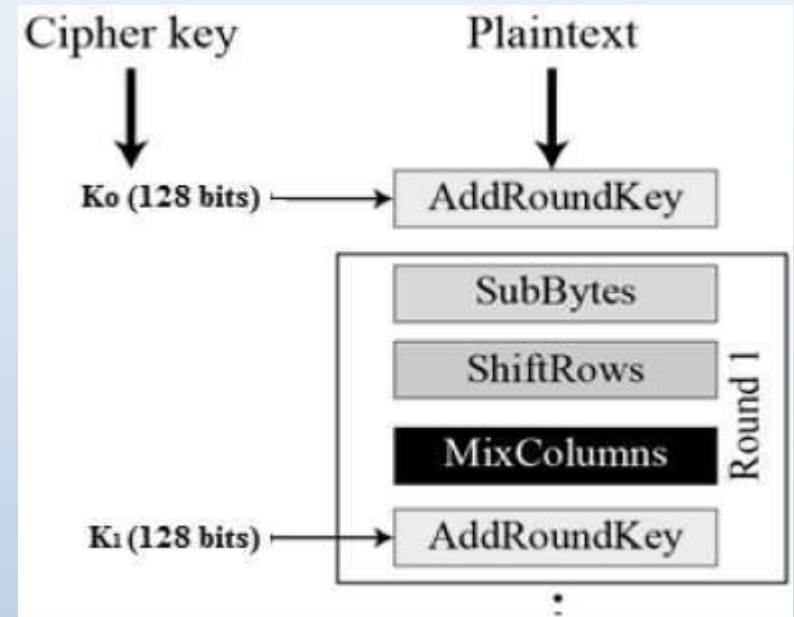
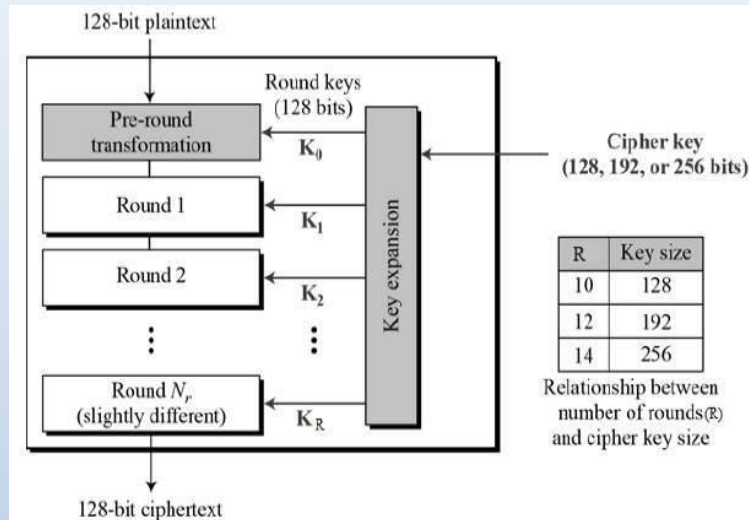
# TLS Proxy

- Each & every connection have to be monitored while in TLS 1.2 certificate message is in plaintext and monitoring can be selective.
- Static DH /RSA is not allowed.

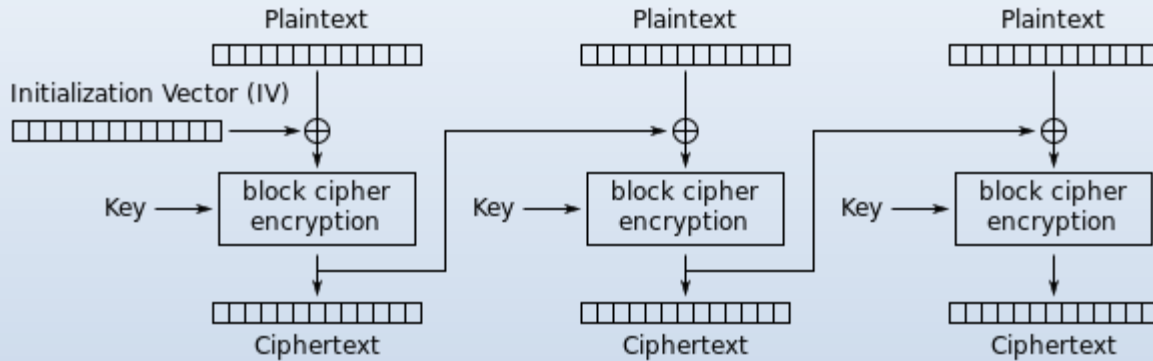
# TOR



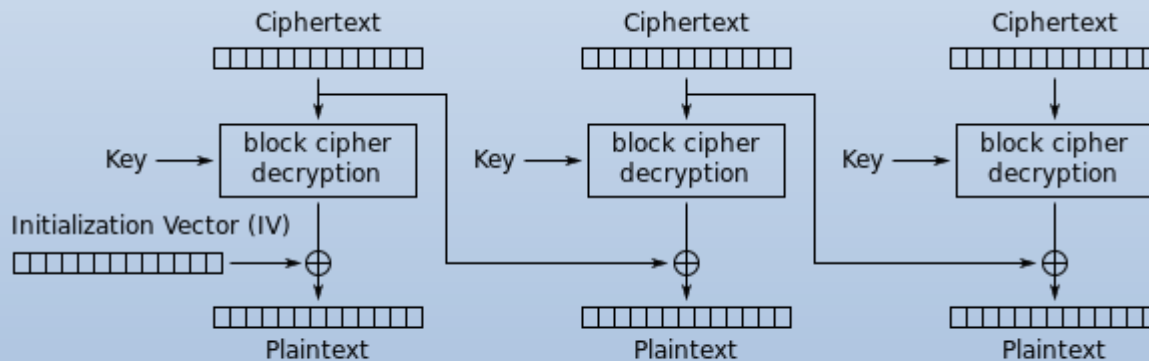
# AES(Advance Encryption Standard)



# AES(Advance Encryption Standard)



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

# AEAD(Authenticated Encryption Additional Data)

