AES-CCM ECC Cipher Suites for TLS

CS290G Network Security

Qin ZHOU

Content

- Transfer Layer Security (TLS)
- Handshake Protocol: ECDHE_ECDSA
- Record Protocol: AEAD_AES_CCM

Transfer Layer Security (TLS)

- Transfer Layer Security (TLS) is application protocol independent. Successor of SSL

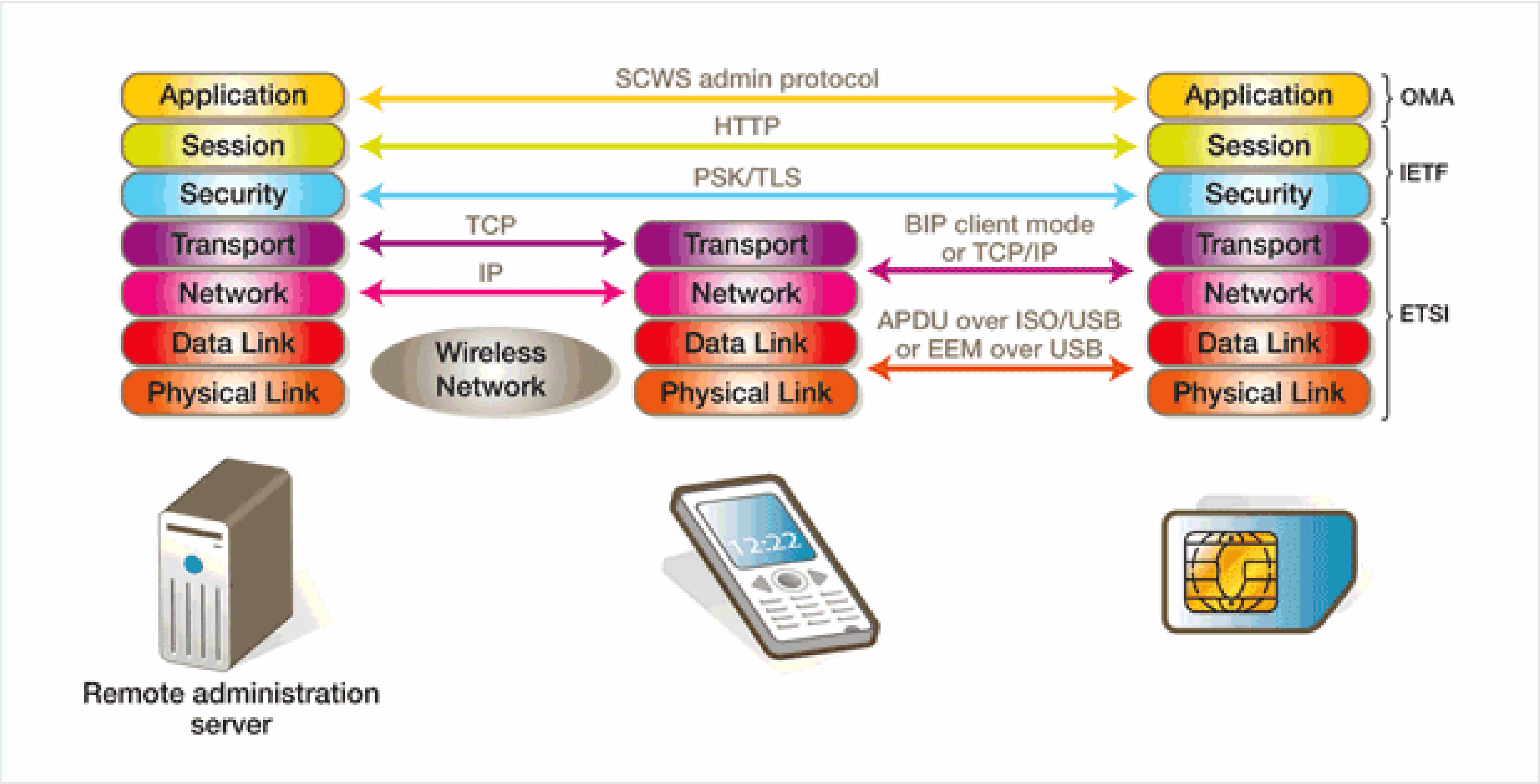

Figure 1: OSI Network Model with TLS layer

Transfer Layer Security (TLS)

- TLS Handshake Protocol
- TLS Record Protocol

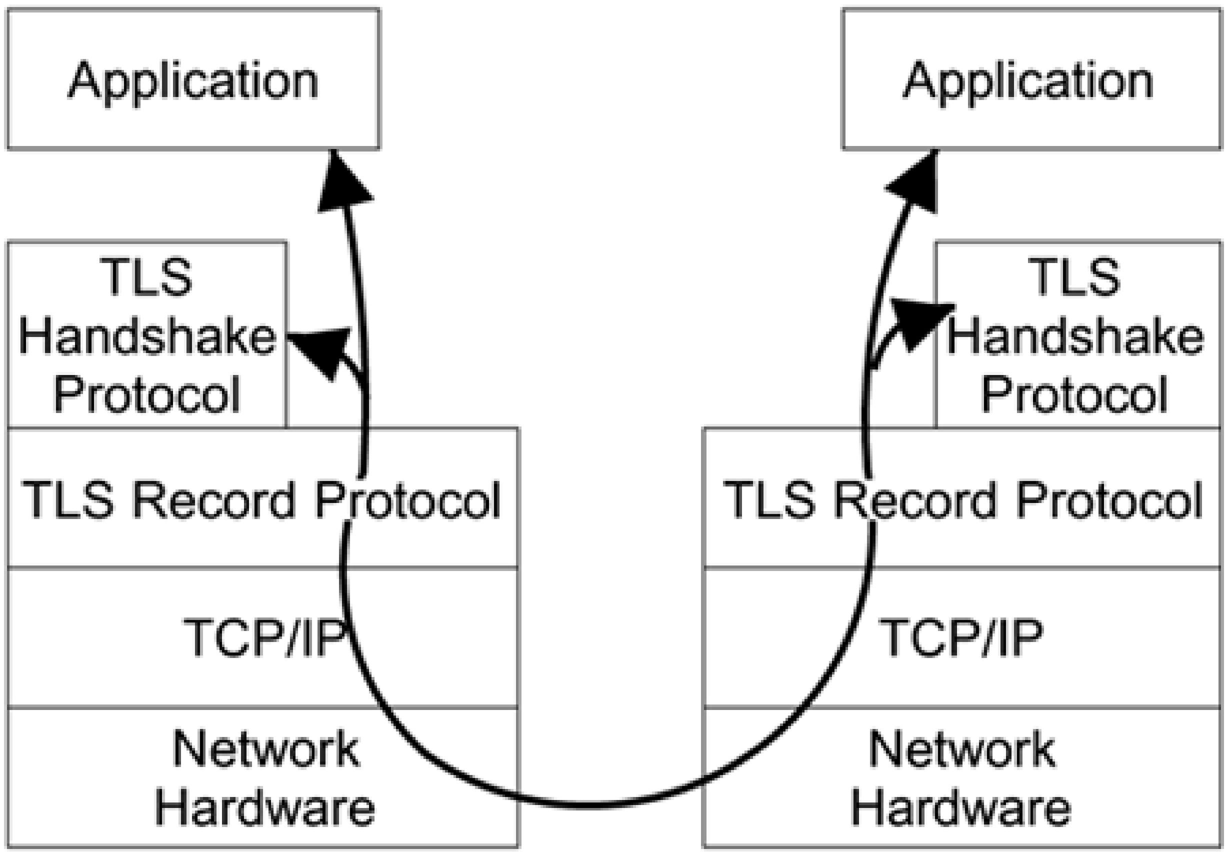

Figure 2: TLS two layers structure

TLS Handshake Protocol

- Responsible for authentication and key exchange necessary to establish or resume secure sessions;
- Manages the following:
  - 1. Cipher suite negotiation
  - 2. Authentication of the server and optionally, the client
  - 3. Session key information exchange

Message Flow for a Full Handshake

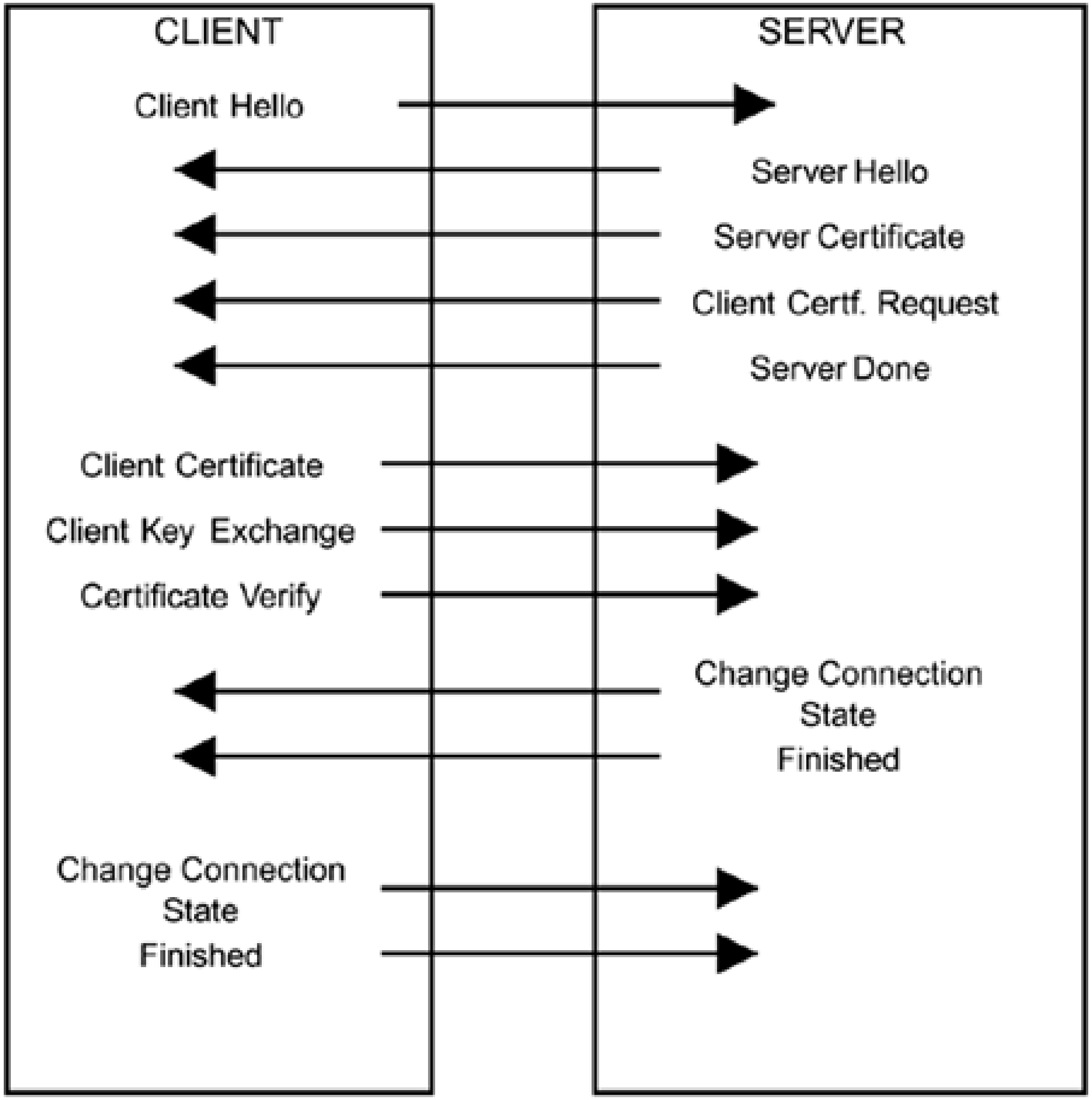

Figure 3: TLS Handshake Steps

TLS Record Protocol

- Responsible for securing application data using the keys created during the Handshake and verifying its integrity and origin
- Manages the following:
  - 1. Dividing outgoing messages and reassembling incoming messages
  - 2. Compressing and decompressing (optional)
  - 3. Applying a Message Authentication Code (MAC) and verifying
  - 4. Encrypting messages and decrypting

Message Flow for a Full TLS

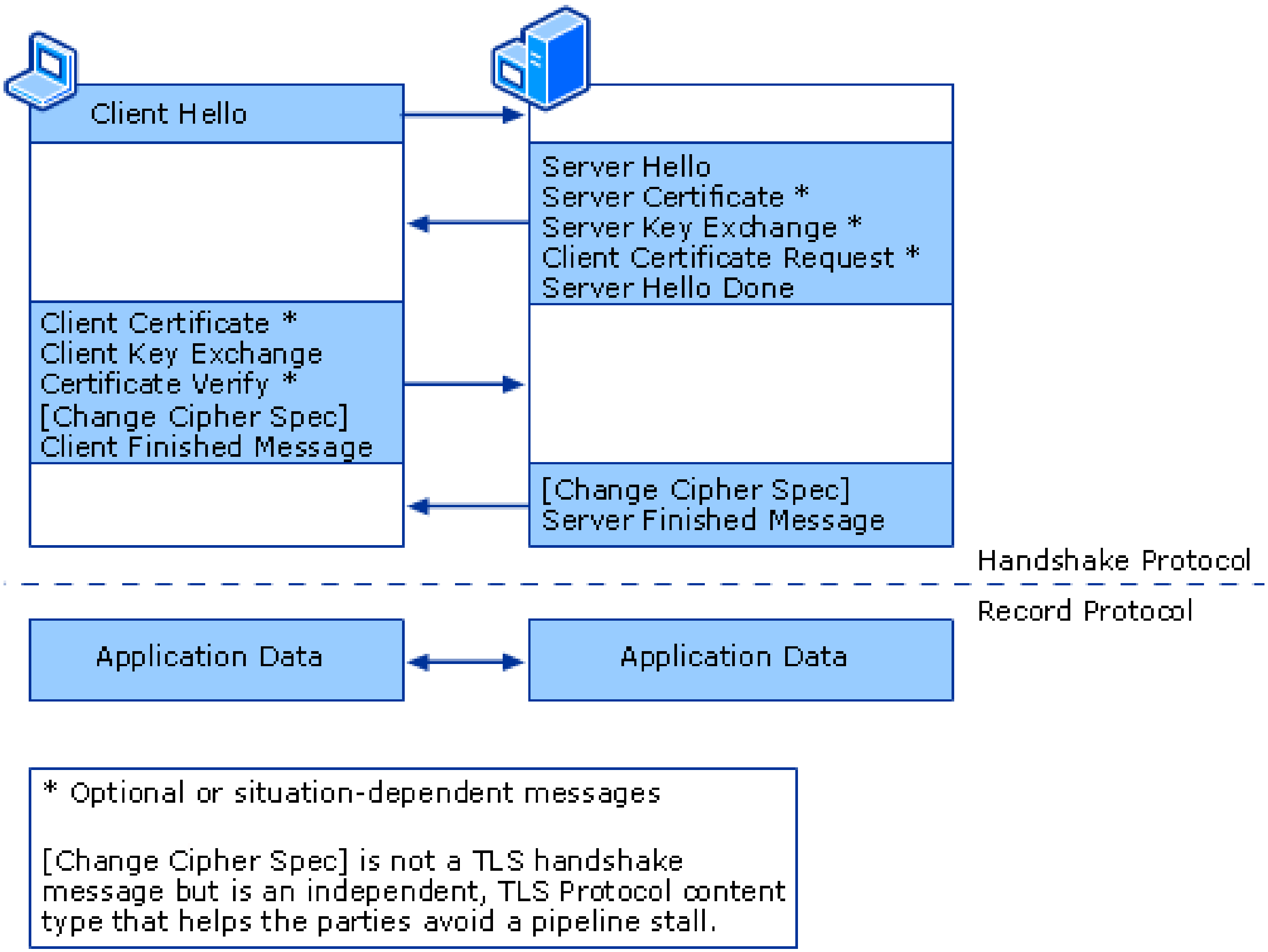

Figure 4: After handshake Record Layer transfer application data

Components of TLS

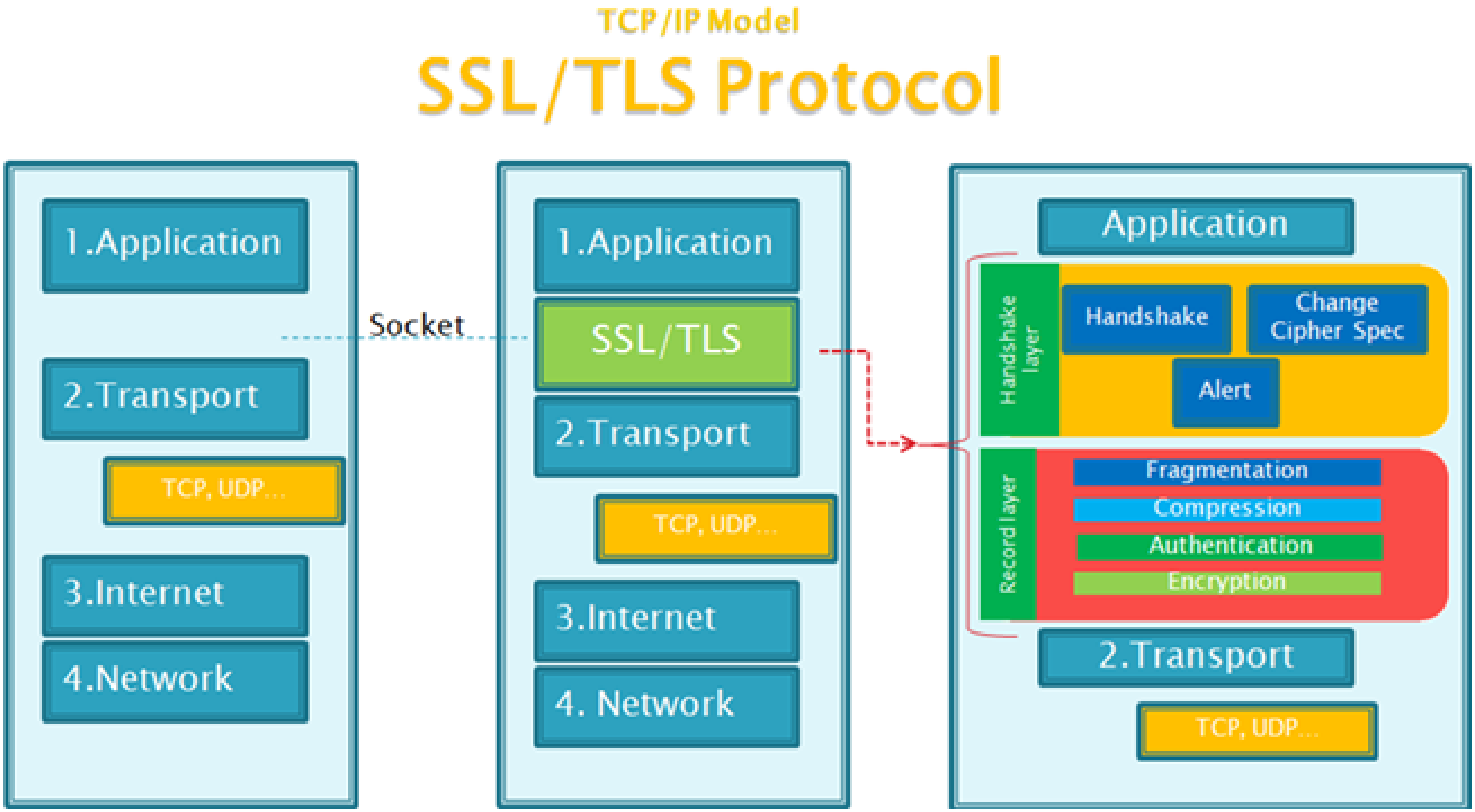

Figure 5: Detailed components of Transfer Layer Security

TLS Ciphersuites

- Ciphersuite: Combination of authentication, encryption and message authentication code (MAC) encryption methods, used to negotiate the communication security settings

Ciphersuite Name	Key Exchange	Cipher	Mac
TLS_NULL_WITH_NULL_NULL	NULL	NULL	NULL
TLS_RSA_WITH_NULL_SHA256	RSA	NULL	SHA256
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256	DHE_DSS	AES_256_CBC	SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_CCM	ECDHE_ECDSA	AES_128_CCM	
TLS_ECDHE_ECDSA_WITH_AES_256_CCM	ECDHE_ECDSA	AES_256_CCM	
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8	ECDHE_ECDSA	AES_128_CCM_8	
TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8	ECDHE_ECDSA	AES_256_CCM_8	

Table 1: Part of TLS ciphersuites, in red are our goals

Content

- Transfer Layer Security (TLS)
- Handshake Protocol: ECDHE_ECDSA
  - Diffe-Hellman Key Exchange Algorithm
  - Digital Signature Algorithm
  - Elliptic Curve Cryptography
  - Elliptic Curve Diffe-Hellman Ephemeral (ECDHE)
  - Elliptic Curve Digital Signature Algorithm (ECDSA)
- Record Protocol: AEAD_AES_CCM

Handshake Protocol: ECDHE_ECDSA

- ECDHE_ECDSA: Ephemeral Elliptic Curve Deffe Halman with Elliptic Curve Digital Signature Authentication
- Steps
  - Review Diffe-Hellman Key Exchange Algorithm and Digital Signature Algorithm
  - Introduce Elliptic Curve Cryptography
  - Illustrate pseudo-code of ECDHE and ECDSA

Content

- Transfer Layer Security (TLS)
- Handshake Protocol: ECDHE_ECDSA
  - Diffe-Hellman Key Exchange Algorithm
  - Digital Signature Algorithm
  - Elliptic Curve Cryptography
  - Elliptic Curve Diffe-Hellman Ephemeral (ECDHE)
  - Elliptic Curve Digital Signature Algorithm (ECDSA)
- Record Protocol: AEAD_AES_CCM

Diffe-Hellman Key Exchange Algorithm

- Basic DH based on intractability of discrete logarithm

	Alice	Evil Eve	Bob
	Alice and Bob exchange a Prime (P) and a Generator (G) in clear text, such that P > G and G is Primitive Root of P **G = 7, P = 11**	**Evil Eve sees** **G = 7, P = 11**	**Alice and Bob exchange a Prime (P) and a Generator (G) in clear text, such that P > G and G is Primitive Root of P** **G = 7, P = 11**
1	Alice generates a random number: X_A $X_A=6$ (Secret)		Bob generates a random number: X_B $X_B=9$ (Secret)
2	$Y_A=G^{X_A}(\text{mod } P)$ $Y_A = 7^6 \ (\text{mod } 11)$ $Y_A = 4$		$Y_B=G^{X_B}(\text{mod } P)$ $Y_B = 7^9 \ (\text{mod } 11)$ $Y_B = 8$
3	Alice receives $Y_B = 8$ in clear-text	Evil Eve sees $Y_A = 4$, $Y_B = 8$	Bob receives $Y_A = 4$ in clear-text
4	Secret Key $=Y_B^{X_A}(\text{mod } P)$ Secret Key $= 8^6 \ (\text{mod } 11)$ **Secret Key = 3**		Secret Key $=Y_A^{X_B}(\text{mod } P)$ Secret Key $= 4^9 \ (\text{mod } 11)$ **Secret Key = 3**

Figure 6: Steps of discrete logarithm diffie-hellman key exchange

Content

- Transfer Layer Security (TLS)
- Handshake Protocol: ECDHE_ECDSA
  - Diffe-Hellman Key Exchange Algorithm
  - Digital Signature Algorithm
  - Elliptic Curve Cryptography
  - Elliptic Curve Diffe-Hellman Ephemeral (ECDHE)
  - Elliptic Curve Digital Signature Algorithm (ECDSA)
- Record Protocol: AEAD_AES_CCM

Digital Signature Algorithm

- DSA Domain Parameter Generation
  - 160bit prime q, 1024bit prime p and g $= h^{(p-1)/q} mod\ p$
- DSA Key Pair Generation
  - Random integer x, $1 \leq x \leq q-1$, x is private key
  - y $= g^x mod\ p$, y is public key

Digital Signature Algorithm

- DSA Signature Generation: sign message m
  - 1. Random integer k, $1 \leq k \leq q - 1$;
  - 2. Compute $X = g^k mod\, p$ and $r = X\, mod\, q$. If $r = 0$ then go to step 1;
  - 3. Compute $k^{-1}\, mod\, q$;
  - 4. Compute $e = SHA1(m)$;
  - 5. Compute $s = k^{-1}\{e + xr\}\, mod\, q$. If $s = 0$ then go to step 1;
  - 6. The signature for message m is (r, s).

Digital Signature Algorithm

- DSA Signature Verification
  - 1. Verify r and s are integers in the interval $[1, q-1]$;
  - 2. Compute $e = SHA1(m)$;
  - 3. Compute $w = s^{-1} \, mod \, q$;
  - 4. Compute $u_1 = ew \, mod \, q$ and$u_2 = rw \, mod \, q$;
  - 5. Compute $X = g^{u_1} y^{u_2} mod \, p$ and $v = X \, mod \, q$;
  - 6. Accept the Signature if and only if $v = r$.

Digital Signature Algorithm

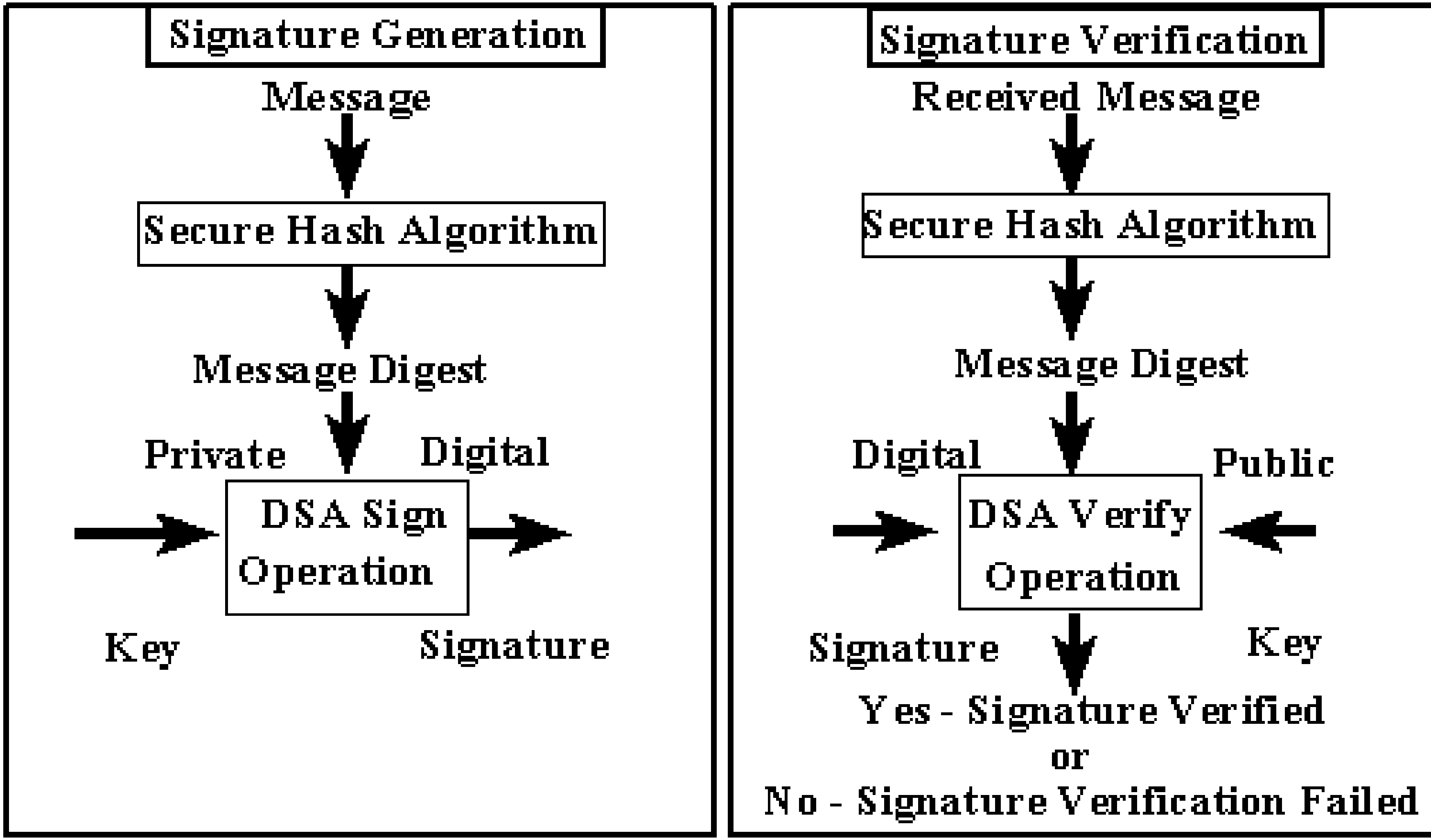

Figure 7: Steps of digital signature algorithm

Content

- Transfer Layer Security (TLS)
- Handshake Protocol: ECDHE_ECDSA
  - Diffe-Hellman Key Exchange Algorithm
  - Digital Signature Algorithm
  - Elliptic Curve Cryptography
  - Elliptic Curve Diffe-Hellman Ephemeral (ECDHE)
  - Elliptic Curve Digital Signature Algorithm (ECDSA)
- Record Protocol: AEAD_AES_CCM

Elliptic Curve Cryptography

- $p > 3$, an odd prime,
- Elliptic curve E over F_p is defined by an equation of form (Weierstrass equation)

$$y^2 = x^3 + ax + b \qquad (1)$$

- Where $a, b \in \mathrm{F}_p$, and $4a^3 + 27b^2 \equiv 0 \ (mod\ p)$
- Set $E(\mathrm{F}_p)$ consists of all points (x, y), $x \in \mathrm{F}_p$, y$\in \mathrm{F}_p$, which satisfy the defining equation (1), together with a special point O called *the point at infinity*

(Another form of Weierstrass equation:

$$y^2 + xy = x^3 + ax^2 + b \qquad (2)$$

For binary finite fields GF(2^m). Ignore now since we will not use it.)

Elliptic Curve: Example

- Let E be the curve

$$y^2 = x^3 + x + 4 \qquad (3)$$

over the field $GF(23)$, then the points on $E(\mathrm{F}_{23})$ are:

- $\{O, (0,2), (0,21), (1,11), (1,12), (4,7), (4,16), (7,3), (7,20), (8,8), (8,15), (9,11), (9,12), (10,5), (10,18), (11,9), (11,14), (13,11), (13,12), (14,5), (14,18), (15,6), (15,17), (17,9), (17,14), (18,9), (18,14), (22,5), (22,19)\}$

- Thus, the order of E is $\#E(GF(23)) = 29$

EC Operations: Point Addition

- $P = (x_1, y_1) \in E(\mathrm{F}_p)$ and $\mathrm{Q} = (x_2, y_2) \in E(\mathrm{F}_p)$, where $P \neq \pm Q$, then $\mathrm{P} + \mathrm{Q} = (x_3, y_3)$, where

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \quad (4)$$

and

$$y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1 \quad (5)$$

EC Operations: Point Addition

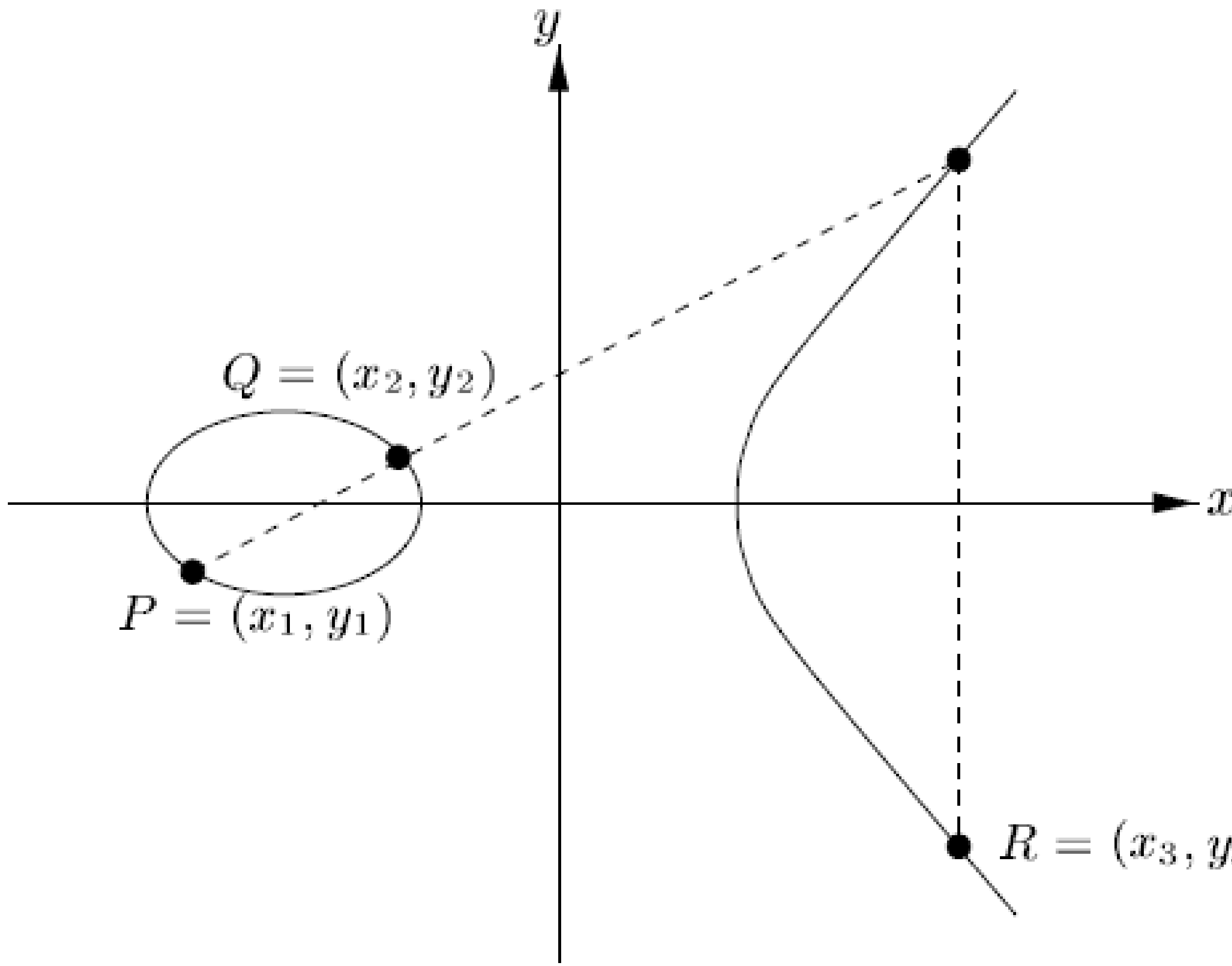

Figure 8: Geometric description of the addition of two distinct elliptic curve points: $P + Q = R$

EC Operations: Point Doubling

- $P = (x_1, y_1) \in E(\mathrm{F}_p)$, where $P \neq -P$, then $2\mathrm{P} = (x_3, y_3)$, where

$$x_3 = \left(\frac{3{x_1}^2 + a}{2y_1}\right)^2 - 2x_1 \qquad (6)$$

and

$$y_3 = \left(\frac{3{x_1}^2 + a}{2y_1}\right)(x_1 - x_3) - y_1 \qquad (7)$$

EC Operations: Point Doubling

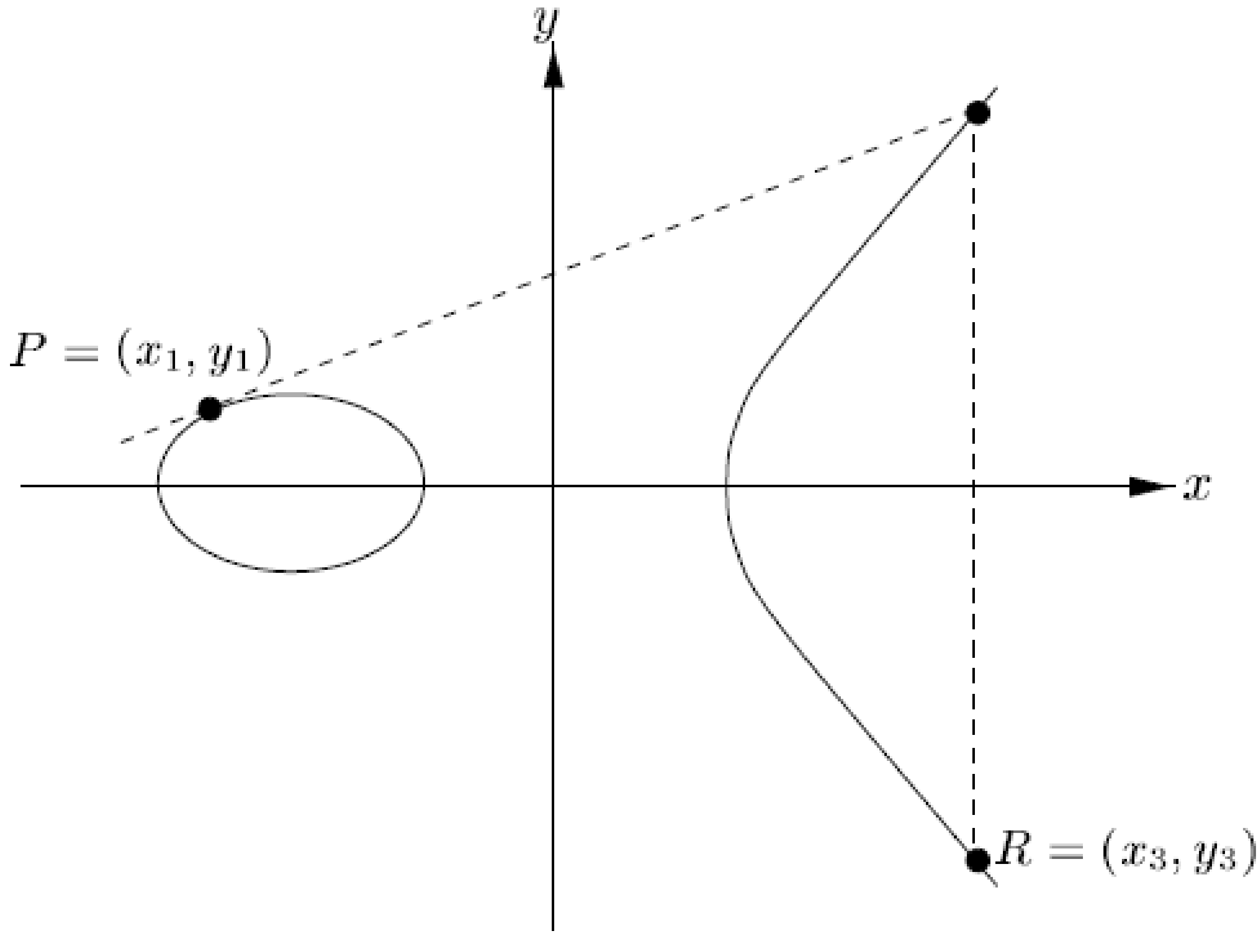

Figure 9: Geometric description of the doubling of an distinct elliptic curve point: $P + P = R$

Property of Elliptic Curve

- $E(\mathrm{F}_p)$ is an abelian group of rank 1 or 2, which means $E(\mathrm{F}_p)$ is isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$, where n_2 divides n_1, n_2 divides $q-1$, for unique positive integers n_1 and n_2
- $\mathbb{Z}_n$ denotes the cyclic group of order n. If $n_2 = 1$, then $E(\mathrm{F}_q)$ is isomorphic to $\mathbb{Z}_{n_1}$, and is cyclic
- When $E(\mathrm{F}_q)$ is isomorphic to $\mathbb{Z}_{n_1}$ (rank=1 now), and there exists a point $P \in E(\mathrm{F}_q)$ such that $E\left(\mathrm{F}_q\right) = \{kP : 0 \leq k \leq n_1 - 1\}$. Point P is called a generator of $E(\mathrm{F}_q)$

Cyclic Elliptic Curve: Example

- Consider the elliptic curve $E(\mathrm{F}_{23})$ defined in previous example. Since $\#E(GF(23)) = 29$, which is prime, $E(\mathrm{F}_{23})$ is cyclic and any point other than O is a generator of $E(\mathrm{F}_{23})$. For example, when $P = (0,2)$ is a generator:

$$1P = (0,2), 2P = (13,12),$$
$$3P = (11,9), 4P = (1,12),$$
$$\ldots, 28P = (0,21), 29P = O$$

How to replace Discrete Logarithm with Elliptic Curve

- Two parties agree on elliptic curve settings (a sextuple $T = (p, a, b, G, n, h)$
- p: GField's prime;
- a, b: two parameters for $y^2 = x^3 + ax + b$ (or other curve equations);
- G: a base point of $E(\mathrm{F}_p)$;
- n: order of G;
- h: cofactor of G;

How to replace Discrete Logarithm with Elliptic Curve

- Given the set of EC domain parameters, generate EC key pair (W, s)
- s is private key, which is an integer in range $[1, r-1]$
- W is public key, which is a point on $E(\mathrm{F}_p)$, where $W = sG$
- It is hard to find s if we only know W and G. Just like in discrete logarithm it is hard to find a if we only know $\mathrm{A} = g^a \; mod \; p$, g, and p

Advantage of ECC: smaller key sizes

- Using more complex math to shorten the key sizes, while complex math does not mean adding time complexity in algorithm

Symmetric	ECC	DH/DSA/RSA
80	163	1024
112	233	2048
128	283	3072
192	409	7680
256	571	15360

Table 2: Comparable Key Sizes (in bit)

Content

- Transfer Layer Security (TLS)
- Handshake Protocol: ECDHE_ECDSA
  - Diffe-Hellman Key Exchange Algorithm
  - Digital Signature Algorithm
  - Elliptic Curve Cryptography
  - Elliptic Curve Diffe-Hellman Ephemeral (ECDHE)
  - Elliptic Curve Digital Signature Algorithm (ECDSA)
- Record Protocol: AEAD_AES_CCM

Elliptic Curve Diffe-Hellman Ephemeral (ECDHE)

- Using ECKAS-DH1 scheme with the identity map as key derivation function (KDF)

ECKAS-DH1: Elliptic Curve Key Agreement Scheme, Diffie-Hellman version

- Secret value derivation primitive: ECSVDP-DH, or ECSVDP-DHC
- A sequence of shared secret keys $K_1, K_2, \ldots, K_t$, shall be generated by each party by following steps
  - 1. Establish the valid set of EC domain parameters;
  - 2. Select a valid private key s for the operation;
  - 3. Obtain the other party's purported public key w' for the operation;
  - 4. Compute a shared secret value z from private key s and other party's public key w' with the selected secret value derivation primitive;
  - 5. Convert the shared secret value z to an octet string Z using FE2OSP.
  - 6. For each shared secret key to be agreed on
    - Establish or otherwise agree on key derivation parameters Pi for the key;
    - Derive a shared secret key Ki from the octet string Z and the key derivation parameters Pi with the selected key derivation function

Content

- Transfer Layer Security (TLS)
- Handshake Protocol: ECDHE_ECDSA
  - Diffe-Hellman Key Exchange Algorithm
  - Digital Signature Algorithm
  - Elliptic Curve Cryptography
  - Elliptic Curve Diffe-Hellman Ephemeral (ECDHE)
  - Elliptic Curve Digital Signature Algorithm (ECDSA)
- Record Protocol: AEAD_AES_CCM

Elliptic Curve Digital Signature Algorithm (ECDSA)

- Elliptic Curve Digital Signature Algorithm (ECDSA)

Content

- Transfer Layer Security (TLS)
- Handshake Protocol: ECDHE_ECDSA
- Record Protocol: AEAD_AES_CCM
  - AEAD: Authenticated encryption with associated data
  - AES: Advanced encryption standard
  - CCM: Counter with Cipher Block Chaining – Message Authentication Code

Authenticated Encryption with Associated Data (AEAD)

- Many cryptographic applications require both confidentiality and message authentication
- Confidentiality: data is available only to those authorized to obtain it, usually realized through encryption;
- Message authentication: data has not been altered or forged by unauthorized entities;
- AEAD algorithm will provide both by using a single cryptoalgorithm

Content

- Transfer Layer Security (TLS)
- Handshake Protocol: ECDHE_ECDSA
- Record Protocol: AEAD_AES_CCM
  - AEAD: Authenticated Encryption with Associated Data
  - AES: Advanced Encryption Standard
  - CCM: Counter with Cipher Block Chaining – Message Authentication Code

Advanced Encryption Standard (AES)

- Originally called Rijndael algorithm;

Content

- Transfer Layer Security (TLS)
- Handshake Protocol: ECDHE_ECDSA
- Record Protocol: AEAD_AES_CCM
  - AEAD: Authenticated Encryption with Associated Data
  - AES: Advanced Encryption Standard
  - CCM: Counter with Cipher Block Chaining – Message Authentication Code

CCM: Counter with Cipher Block Chaining – Message Authentication Code

- Generation-encryption process:

Prerequisites:

block cipher algorithm;

key K;

counter generation function;

formatting function;

MAC length Tlen.

Input:

valid nonce N;

valid payload P of length Plen bits;

valid associated data A;

Output:

ciphertext C.

CCM: Counter with Cipher Block Chaining – Message Authentication Code

- Generation-encryption process:

<u>*Steps:*</u>

1. Apply the formatting function to (N, A, P) to produce the blocks B0, B1, …, Br.
2. Set Y0= CIPHK(B0).
3. For i = 1 to r, do Yi = CIPHK(Bi $\oplus$ Yi-1).
4. Set T=MSBTlen(Yr).
5. Apply the counter generation function to generate the counter blocks Ctr0, Ctr1, …, Ctrm, where $\lceil\rceil$128Plenm=.
6. For j=0 to m, do Sj= CIPHK(Ctrj).
7. Set S= S1 || S2 || …|| Sm.
8. Return C=(P $\oplus$ MSBPlen(S)) || (T $\oplus$ MSBTlen(S0)).

CCM: Counter with Cipher Block Chaining – Message Authentication Code

- Decryption-verification process:

Prerequisites:
block cipher algorithm;
key K;
counter generation function;
formatting function;
valid MAC length Tlen.

Input:
nonce N;
associated data A;
purported ciphertext C of length Clen bits;

Output:
either the payload P or INVALID.

CCM: Counter with Cipher Block Chaining – Message Authentication Code

- Generation-encryption process:

Steps:

1. If Clen≤Tlen, then return INVALID.

2. Apply the counter generation function to generate the counter blocks Ctr0, Ctr1, …, Ctrm, where ⌈128)(TlenClenm−=.

3. For j=0 to m, do Sj= CIPHK(Ctrj).

4. Set S= S1 || S2 || …|| Sm.

5. Set P=MSBClen-Tlen(C) ⊕ MSBClen-Tlen(S).

6. Set T=LSBTlen(C) ⊕ MSBTlen(S0).

7. If N, A, or P is not valid, as discussed in Section 5.4, then return INVALID, else apply the formatting function to (N, A, P) to produce the blocks B0, B1, …, Br.

8. Set Y0= CIPHK(B0).

9. For i = 1 to r, do Yj = CIPHK(Bi ⊕ Yi-1).

10. If T≠MSBTlen(Yr), then return INVALID, else return P.

Implementation

- Set up TCP/IP Link
- AES Component
- Record Layer: AEAD_AES_128_CCM
- ECDHE Component
- ECDSA Component
- Handshake Layer: ECDHE_ECDSA
- Test
- To Do List

Set up TCP/IP Link

- Using Windows Sockets API (Winsock)
- Basic client and server code from MSDN
  - http://msdn.microsoft.com/en-us/library/windows/desktop/ms737591(v=vs.85).aspx
  - http://msdn.microsoft.com/en-us/library/windows/desktop/ms737593(v=vs.85).aspx

Structure of Client

```
Initialize socket;
Clientphase = InitialC (or TestC);
While(){
        clientphase:
                TestC: test the socket api
                InitialC: initialize all parameters
                HandshakeRelatedC: do handshake
                RecordPrepareC: generate keys with master secret
                RecordLayerC: transfer the data and verify the
                              received data
                ExitC: exit the while loop
}
Close the socket;
```

Structure of Server

```
Initialize socket;
Serverphase = InitialS (or TestS);
While(){
        serverphase:
                TestS: test the socket api
                InitialS: initialize all parameters
                HandshakeRelatedS: do handshake
                RecordPrepareS: generate keys with master secret
                RecordLayerS: transfer the data and verify the
                                received data
                ExitS: exit the while loop
}
Close the socket;
```

Implementation

- Set up TCP/IP Link
- AES Component
- Record Layer: AEAD_AES_128_CCM
- ECDHE Component
- ECDSA Component
- Handshake Layer: ECDHE_ECDSA
- Test
- To Do List

AES Component

- From this link: http://code.google.com/p/lostinactionscript/downloads/detail?name=AES.zip&can=2&q=

Implementation

- Set up TCP/IP Link
- AES Component
- Record Layer: AEAD_AES_128_CCM
- ECDHE Component
- ECDSA Component
- Handshake Layer: ECDHE_ECDSA
- Test
- To Do List

Record Layer: AEAD_AES_128_CCM

- Implemented according many references.

Implementation

- Set up TCP/IP Link
- AES Component
- Record Layer: AEAD_AES_128_CCM
- ECDHE Component
- ECDSA Component
- Handshake Layer: ECDHE_ECDSA
- Test
- To Do List

Test

1. Did a complete single package test.

2. Since we only have record layer, server will set the security parameters and send to client. Then both of them use these parameters to generate the keys.

Test: Client Output

C:\windows\system32\cmd.exe

```
Security Parameters Received
InitialC Finished
RecordPrepareC Finished
Plaintext Sent
This is a client test!

Client RecordLayer Send Finished
Server Ciphertext Received:     51
outputUint8s: 23        255     255     38      0       0       0       0
0       0       0       0       0       1       145     61      46      138
176     182     32      192     96      240     113     3       90      238
253     32      116     101     115     116     33      20      163     186
1       147     157     126     232     97      232     229     9       44
84      7       148
Plaintext Valid? true
Plaintext
Recv:
This is a server test!
RecordLayer Finished
Press any key to continue . . .
```

Test: Server Output

C:\Windows\system32\cmd.exe

```
Security Parameters Sent
InitialS Finished
RecordPrepareS Finished
Plaintext Sent
This is a server test!

Server RecordLayer Send Finished
Client Ciphertext Received:     51
outputUint8s: 23        255     255     38      0       0       0       0
0       0       0       0       0       58      16      85      230     164
43      191     237     2       192     29      139     185     38      74
241     32      116     101     115     116     33      201     9       169
45      83      52      104     11      177     76      178     225     99
246     85      242
S:
outputUint8s: 110       120     60      149     132     66      204     205
99      224     126     231     208     67      36      133
Plaintext Valid? true
Plaintext
Recv len:       22
This is a client test!
RecordLayerS Finished
请按任意键继续. . .
```

To Do List

1. Optimize the current code. There are a lot of questions that need to be solved. I have kept them in a google doc;
2. Expand the code to normal data transfer requirement. For example, transfer a big html page, or provide an API for http service;
3. Finish the handshake protocol.

Thank you!