Chapter 4

Key exchange protocols

Cryptographic hash functions, MACs

Digital signature

Key exchange protocols

RSA Key Exchange

DHE

ECDHE

Key Exchange protocols

In TLS data is encypted with symmetric block cipher AES

Before encryption with AES starts, the symmetric key must be agreed between the communicating parties. For this there exists three commonly used algorithms, which are called key exchange protocols

- **RSA Exchange** has been the most common method in the past years
- **ECDHE** has replaced RSA Exchange in many TLS connections
- **DH Exchange** is the third option, which is also widely used (f.e in secure video negotiations encrypted with AES)

DHE = Diffie Hellman key Exhange
ECDHE = Elliptic Curve Diffie Hellman key Exchange

RSA key exchange

One of the communicating parties creates a random symmetric key and sends a copy to the other party encrypted with RSA

Possible security issues

1. **If the hacker manages to factor recipients public RSA key, he may get from the key excange messages the AES keys of present and future sessions.** (RSA public keys are permanent and remain the same at least for the time when servers certificate is valid)

2. **Authorities** could require companies to give private RSA keys of their servers to officials.

Diffie – Hellman key exchange

System parameters: Large (2048 bit) prime **p** and a generator **g** (base for exponentiation)

Alice creates a random private key a

Bob creates a random private key a

A calculates Ya using a as exponent and sends Ya to Bob in unsecure channel

$$Y_a = g^a \bmod p$$

$$Y_b = g^b \bmod p$$

B calculates Yb using b as exponent and sends Yb to Alice in unsecure channel

A calculates key K:

$$K = Y_b^a \bmod p$$
$$= (g^b)^a \bmod p$$
$$= g^{ab} \bmod p$$

B calculates key K:

$$K = Y_a^b \bmod p$$
$$= (g^a)^b \bmod p$$
$$= g^{ab} \bmod p$$

Both use their own private keys to calculate symmetric key K from the messages they have received

To avoid Man in the Middle attack users can have their own system parameters g and p. In this case the public key of Alice would be the triple (p_A,g_A,Y_A), which Bob gets from digitally signed certificate of the server.

Finding group generators g of multiplicative group Z_p*

Basic mathematical facts:

1. All elements of $Z_p^* = \{1,2,\ldots, p-1)$ generate a cyclic subgroup of Z_p^*

2. The order of element a, denoted Ord(a) = size of the subgroup generated by element a.

3. Ord(a) is a divisor of p-1 for any element a in Z_p^*. (based on Lagrange's theorem)

4. If Ord(g) is p-1, element g is called primive root or generator of Z_p^*.

5. If divisors of p - 1 are $d_1, d_2, \ldots, d_n$, where $d_1 = 1$ and $d_n = p-1$ the generators are those elements g of Z_p^*, for which only the last number of sequence $g^{d1}, g^{d2}, \ldots, g^{dn}$ equals 1 (mod p)

Example: p = 29 is a prime. Find some generator g of multiplicative group Z_{29}^*

Solution: The divisors of p – 1 (28) = 1, 2,4,7, 14, 28. Perform generator test for candidates 5 and 2

$\{5^1,5^2,5^4,5^7,5^{14},5^{28}\}$ (mod 29) = {5, 25, 16, 28, 1, 1} => 5 is not a generator, because there are two 1's

$\{2^1,2^2,2^4,2^7,2^{14},2^{28}\}$ (mod 29) = {2, 4, 16, 12, 28, 1} => 2 is a generator, while only the last power equals 1.

Using ”strong primes” as modulus p makes finding generators g easier

If p is very large prime (f.e 1000 bits), it may be impossible to find the divisors of p – 1 , which are needed for generator tests explaned in the previous slide.

Example if p = 265738830135992486377941683556469254997964098756853,
then p – 1 = 265738830135992486377941683556469254997964098756852 , which is even number, which is so large that we cannot factor it and cannot find other divisors than obvious divisor 2.

=> Thus there is no way to perform generator tests for candidate numbers.

Diffie – Hellman prefers using strong primes, where both p and (p-1)/2 are primes. For strong primes p the divisor lisf of p – 1 is short: {1, 2, (p-1)/2, p-1}. Strong primes are very rare.

For example in range [10000,20000] there are 1033 primes, from which 75 are strong primes.
Finding them requires a program which creates random primes p and checks if also (p-1)/2 is prime.

Example of strong prime is p = 200087. The divisors of p-1 are {1,2,100043,200086}. To test is number 5 is a generator of Zp*, we calculate powers 5^1, 5^2, 5^{100043} and 5^{200086} (mod 200087) using Wolfram Alpha. Result is {5,25, 200086, 1}, Because only the last power equals 1 , number 5 is a generator.

Security of DHE is based on Discrete Logarithm Problem

A hacker can get the public messages Ya = g^a mod p ja Yb = g^b mod p by listening the unsecure channel, but he cannot solve private keys **a** and **b** from these number.

Solving exponent x from y = g^x mod p is as hard as factoring large integers in RSA

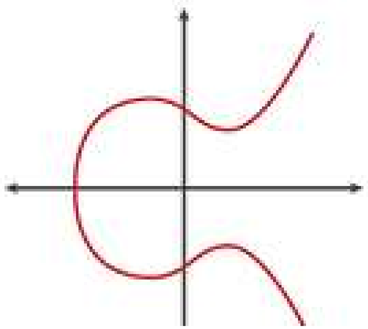
DLP = " Discrete Logarithm Problem" is the name of this "hard problem", which is behind the security of DHE.

DLP: Solve integer x from y = g^x mod p, where integers y, g and prime p are given

Usual recommendation for secure size of p is 2048 bits

ECDHE   Elliptic Curve Diffie Hellman Exchange

A more recent key exhange is ECDHE , which has replaced RSA exchange in many web services (for example in some Finnish banks)



Elliptic curves used in ECC are curves

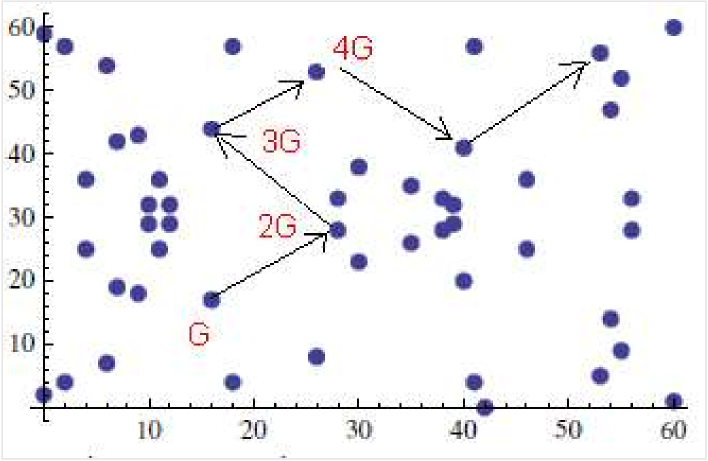
$$\mathbf{y^2 = x^3 + a\,x + b}$$

If P and Q are points of the curve, it is possible to define an addition operation which gives a third point of the curve called the sum P + Q

In ECC cryptography discrete elliptic curves are used. Discrete curve consists of integer pairs (x,y) , which satisfy equation $y^2 = x^3 + a\,x + b \pmod q$
where a and b are integers and modulus q is prime.

Example and graph of a discrete elliptic curve

Equation $y^2 = x^3 + 2x + 4$ (mod 61), where $0 < x,y < 60$



Points of curve $y^2 = x^3 + 2x + 4$ (mod 61)

In this course manual calculations are not required.

Some points of the curve are generating elements. One of them is **G = (2,4)**

The multiples of G generate all points of the curve {G,2G, 3G,…,60G, O}

Formulas of addition P + Q

$$k = (y_2 - y_1)*(x_2 - x_1)^{-1} \bmod q$$

$$X_{P+Q} = k^2 - x_1 - x_2 \quad \bmod q$$

$$y_{P+Q} = y_1 - k\,(x_{P+Q} - x_1) \bmod q$$

ECDHE key exchange

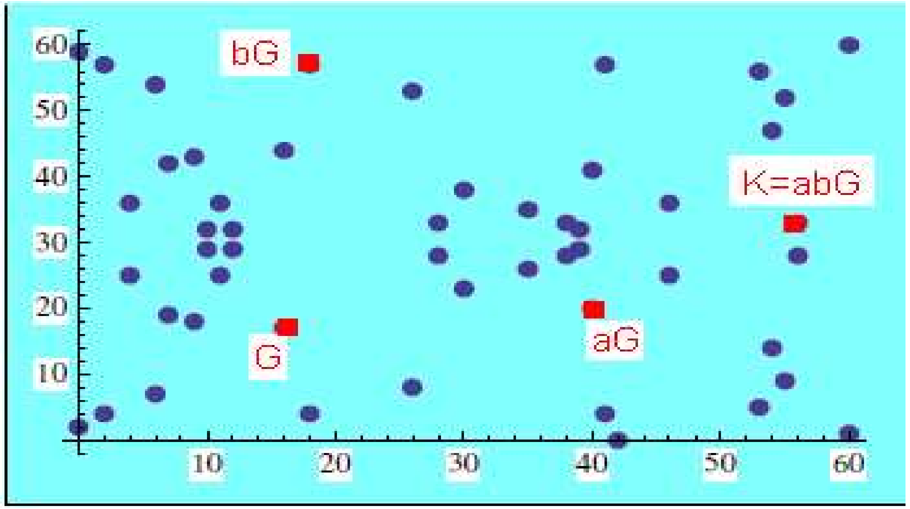
Curve and G are given as system parameters

1. Alice chooses random private key **a** and sends Bob its multiple point **aG** in unsecure channel

2. Bob chooses random private key **b** and sends Alice its multiple point b**G** in unsecure channel

3. Both calculate the symmetric key (K= points abG)

Alice: K = a(bG) Bob: K = b(aG)



TLS uses standard curves, where modulus q has at least 250 bits and nr of points is likewise

Security: The hacker can listen the channel and read points aG and bG, but is not able to calculate private keys a and b

Solving x from xG is called ECDLP , ”Elliptic curve Discrete Logarithm Problem” .
It is a similar ”Hard Problem” of mathematics as factoring behind RSA

Concepts:

Protocol
is a standard, which defines or makes possible connections between devices or software. One party sends a message to the other party, who reacts to the message and possibly answers according to the protocol.

Key exchange protocol
= standard for agreeing on symmetric key

DLP = mathematically hard problem, which is the basis of security of Diffie Hellman Exchange

ECDLP = mathematically hard problem, which is the basis of security of ECDHE exchange

Cryptographic hash functions and MAC’s

Digital signature

Hash functions

Definition: Hash functions are one-way functions, which produce a fixed length digest from the message

Uses of hash functions:

1.Hash value **ensures the integrity of data transmission** (that data is not changed during transmission).

2. **Password files** of servers do not store passwords, but their **hash values**

3. **Digital signature** is calculate using hash value of message as input .

4. **Checksums** are shorter hash values, which for example antivirus software calculates for folders and files.

Hash values are also called ”message digests”

Requirements for a secure hash function

Notation: Hash function is h(m), where argument m is message

1. h(m) is a **one-way function**. Is must not be possible to calculate the message m from its hash value h(m).

2. **Collision resistance 1**. If the hash h(m1) for one message m1 is known, it should be impossible to generate another message m2 with same hash value.
(Other messages surely exists, but there should be no method for finding them)

3. **Collision resistance 2**. It should be impossible to generate two message m1 and m2 with same hash value.

Some known hash functions

”Chinese Xiaoun Wang broke in 2005 almost all hash functions in sight - and paradoxally the community of cryptographers loved it”

Chinese scientist in 2005 showed the vulnerability of many common hash functions by showing how to generate another message with a given hash value

- **MD5** - **broken**
- **SHA-1 (160 bit)** - **not recommended**

- **<u>Following hash functions are regarded as safe</u>**
- **SHA256**
- **SHA384**
- **SHA512**

Example of SHA256

```
Hash["Tämä on koeviesti, josta lasketaan tiiviste","SHA256"]
32134986657396586487192643852384396035054468584117070131813446773863278924851
```

Secure hash length and ”birthday paradox”

The length of secure hash should have 2 x key length of secure block cipher

Reason is ”**birthday paradox**”

What is the minimum number of students in the class, in which there is > 50% probability that the class has two students with the same birthday? Answer: 23

The calculation of probability confirms the result

$$P = 1 - \prod_{k=1}^{23} \frac{365 - k + 1}{365} = 0.51$$

Similar calculation can be done for probabilty of collisions of hash values:

If the size of ”hash space” is n, there will be collisions at probability of 50% among $\sqrt{n}$ hash values
=> The number of 256 bit hashes is 2^{256} . The square root is 2^{128} which is the number of hashes where appears collision at >50% probability.

MAC -functions

MAC = message authentication code

- MAC = hash function with symmetric key K as extra input in addition to message m
- MAC ensures not only the integrity of message, but also authenticates the sender of message

* MAC is use in the same way as digital signatures at the end of transmitted data packets to quarantee integrity and sender's authenticity

HMAC-SHA

= most common MAC which is calculated from message m ja and symmetric key K

HMAC(K, m) = sha(K⊕opad||sha(K⊕ipad ||m)

⊕ = XOR sum

K = symmetric key

Ipad and Opad are 64 bit constants, in hexadecimal form

Ipad = 5c5c5c5c , Opad = 36363636

|| means concatenation of strings (“auto”||”mat” = “automat”)

sha-HMAC – is used as pseudorandom number generator in devices, which produce one-time passwords.
(More about this in the last chapter)

Digital signature

- A digital signature enables secure online transactions. Juridically digitally signed document is equally valid as document manually signed at presence of wittnesses.

The goal of digital signature is to ensure that

1) document/message is not changed

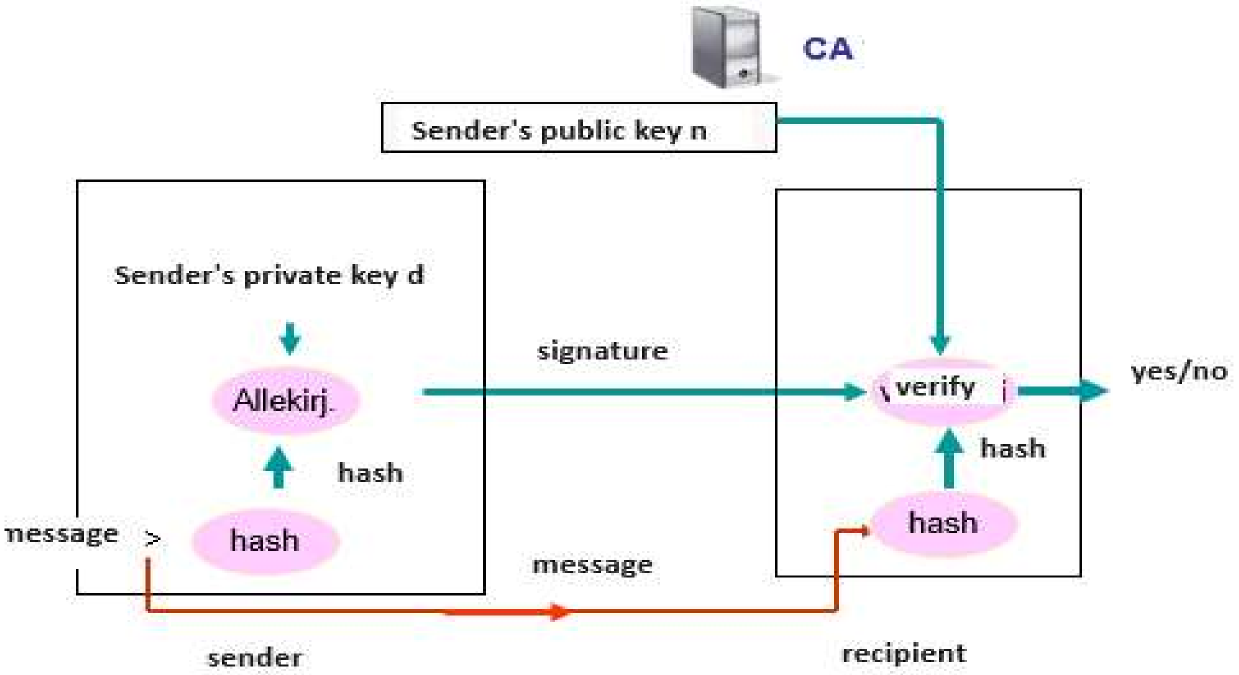
2) sender/author is authenticated

Digital signature algorithm usually uses hash function and public key encryption

Typical signature in TLS is **sha256RSA.**

Other signature ECDSA is also possible

Diagram of digital signature



The recipient receives the pair (m,S), where
m = message
S = signature

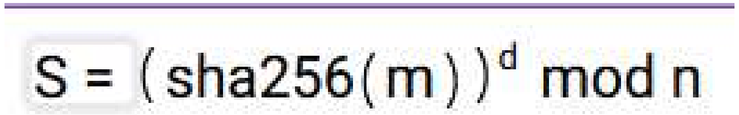
(In addition to diagram:
Usually both m
S are encrypted with
recipients public key)

Digital signature is a number, which is the hash value of message encrypted with senders private key

The recipient verifies the signature in the following way:

1) Decrypts the signature with sender’s public key. Result is the hash value of message.
2) He calculates another hash from the message.
3) If both hash values match, message is unchanged and sender authenticated.

Formula of sha256RSA signature



$$S = (\mathrm{sha256}(m))^{d} \bmod n$$

S = signature
m = message
sha256(m) = hash of the message
d = senders private key
n = senders public key

Example. Message is **”Tämä on koeviesti”**. Calculate signature.

Sender’s keys : n= 308911 , Private key d = 133073

In the example SHA256 is too long compared to senders RSA keys. We make it easier modifying the hash by taking only 4 numbers from the beginning.

WolframAlpha:

SHA "Tämä on koeviesti"

gives 9126…

Calculate the signature using 9126 as h(m)

$$S = sha(m)^d \bmod n = 9126^{133073} \bmod 308911 = 298954$$

Signature S = 298954

Example cont...Show how recipient verifies the signature

1. Recipient decrypts the signature using senders public key n and e = 65537

$$S^e \bmod n = 298954^{65537} \bmod 308911 = 9126$$

2. Recipient calculates SHA of the received message directly

SHA(”Tämä on koeviesti”) gives **9126**...

3. Recipient compares the hash value from signature and hash value calculated the message itself.

HASH values match => message is unchanged and sender authenticated