Chapter 4

Key exchange protocols

Cryptographic hash functions, MACs

Digital signature

Key Exchange protocols

In TLS transmitted data is encypted with symmetric block cipher AES

Before encryption with AES can start, a symmetric key must be agreed between the communicating parties. For this there exists algorithms, which are called key exchange protocols

- **RSA Exchange** has been the most used method in the past years
- **ECDHE** has replaced recently 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)

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

RSA key exchange

One of the communicating parties creates a random symmetric key K and sends the key to the other party using recipients public RSA key.

Possible security concerns

1. **If the hacker manages to factor recipients public RSA key, he is able to decrypt the messages containing session keys. Public keys are permanent and encryption of TLS sessions of the recipients stay broken for a long time.**

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

3. **The pseudorandom number generator used in creating key K has a backdoor and the symmetric keys produced by the generator are not random.** (A generator with backdoor was revealed in 2013)

Diffie – Hellman key exchange

System parameters are: 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 b

Alice calculates a public key Ya using a as exponent and sends Ya to Bob in an unsecure channel

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

Bob calculates a public key Yb using b as exponent and sends Yb to Alice in an unsecure channel

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

Both can now use their private keys to calculate the shared key K from the messages they have received.

Alice calculates

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

Bob calculates

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

To avoid Man in the Middle attack users often have their own system parameters g and p.
In this case the public key of Bob is the triple (p_B ,g_B,Y_B), which Alice gets from certificate of Bob.
The procedure uses Bob's system parameters. Only Alice creates a new random private key for the session.
Bob is in this version probably a web-server, which has fixed public and private keys.

Finding group generators g of multiplicative group Z_p^* *)

Mathematics of Z_p^*, where p is a prime. (Z_p^*= {1,2,…,p - 1} where product a*b is multiplication mod p)

1. All elements **a** of Z_p^* = {1,2,…., p -1) generate a cyclic subgroup of $\mathbf{Z_p^*}$.
i.e the set of powers {a, a^2, …., a^k}, where the last power equals 1 form a subgroup of order k.

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 $\mathbf{Z_p^*}$. (known as 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 in order are d_1, d_2, …, 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}, …, g^{dn} equals 1 (mod p)

***) Definition: A set G with operation * defined between its elements is called a group, if**
G1) a*b ϵ G for all a,b ϵ G
G2) (a*b)*c = a*(b*c) for all a,b ϵ G
G3) G has a neutral element e , for which a*e = e*a = a for all a ϵ G
G4) Every element a of G have an inverse element a^{-1} for which a^{-1}*a = a*a^{-1} = e

Example of cyclic group is Z_{11}^* : table of powers x

Z_{11}^*

		powers 1,2, ..., 10 of elements											
		1	2	3	4	5	6	7	8	9	10	Order of elemen	
e	1	1	1	1	1	1	1	1	1	1	1	1	
l	2	2	4	8	5	10	9	7	3	6	1	10	generator
e	3	3	9	5	4	1	3	9	5	4	1	5	
m	4	4	5	9	3	1	4	5	9	3	1	5	
e	5	5	3	4	9	1	5	3	4	9	1	5	
n	6	6	3	7	9	10	5	8	4	2	1	10	generator
t	7	7	5	2	3	10	4	6	9	8	1	10	generator
s	8	8	9	6	4	10	3	2	5	7	1	10	generator
	9	9	4	3	5	1	9	4	3	5	1	5	
	10	10	1	10	1	10	1	10	1	10	1	2	

Fermat's theorem
If p is prime,
$a^{p-1} \bmod p = 1$

is visualized in the power table:

For all elements x of Z_{11}^*
$x^{10} \pmod{11} = 1$

The table of powers of elements 1 – 10 show that there are 4 generators: 2,6,7, and 8.
Elements 3,4,5 and 9 generate subgroups of 5 elements.
Element 10 generates a subgroup of 2 elements and finally element 1 alone forms a subgroup.

Exampe of finding a group generator in a simple case, where p is small.

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

One of numbers 5 or 2 is a generator of Z_{29}^* . Which ?

Step1: List the divisors of p – 1 = 28 in ascending order: 1, 2, 4, 7, 14 and 28.

Step2: Raise the (candidate) number to all powers of the divisor list. If only the last power p – 1 Is 1, the number is a generator.

Test if **5** is a generator of $\mathbf{Z_{29}^*}$.
$\{5^1,5^2,5^4,5^7,5^{14},5^{28}\}$ (mod 29) = {5, 25, 16, 28, 1, 1} => 5 is not a generator.

Test if **2** is a generator of $\mathbf{Z_{29}^*}$.

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

Using ”strong primes” p makes finding generators easier, when p is large.

If p is very large prime (>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 trivial divisor 2.

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

DH algorithm prefers 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 rare.

For example in range [10000,20000] there are 1033 primes, from which only 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. The powers are {5,25, 200086, 1}, Because only the last power equals 1 , number 5 is a generator.

Security of DH is based on Discrete Logarithm Problem

A hacker can capture 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 them. Solving exponent x from y = g^x mod p is very hard - as hard as factoring large integers in RSA

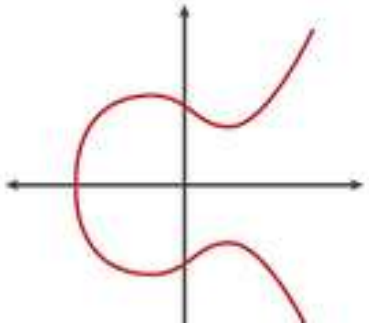
DLP = " Discrete Logarithm Problem" is the name of this "hard problem", which is behind the security of DHE.

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

Usual recommended 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

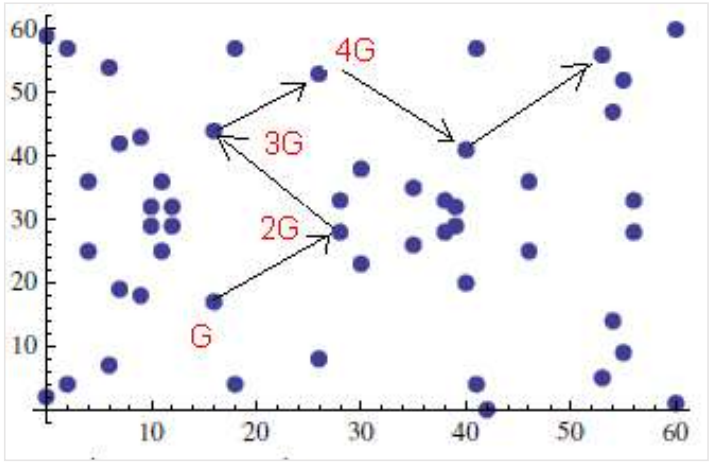
$$y^2 = x^3 + a\,x + b$$

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 $y^2 = x^3 + a\,x + b \pmod p$ where a and b are integers and modulus p is prime.

Example and graph of a discrete elliptic curve

Equation $y^2 = x^3 + 2x + 4 \pmod{61}$, where $0 < x,y < 60$



Points of curve $y^2 = x^3 + 2x + 4 \pmod{61}$

Formulas of addition P + Q

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

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

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

Some points of the curve are generator points. One of them is **G = (4,25)**

This curve has 52 points.

The multiples of G generate all points of the curve {G,2G, 3G,…,51G, O}.

The last element is a symbol O, which is the neutral element added to the curve points.

It works like number zero of normal addition:

P + O = O + P = P for all P of the curve.

ECDHE key exchange

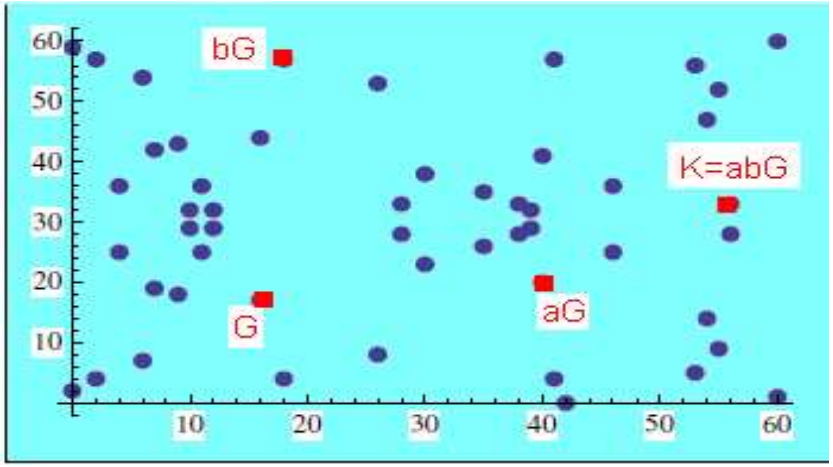
Curve and generator point G are given as system parameters

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

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

3. Both of them calculate the symmetric key K= a*b*G

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



TLS uses standardized curves, where modulus q and number of points have at least 250 bits

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

Solving x from P = x*G, if points G and P are known, is called ECDLP , ”Elliptic curve Discrete Logarithm Problem” . It is a similar ”Hard Problem” as DLP behind DH exchange.

Example of ECDHE using curve $y^2 = x^3 + 2x + 4 \pmod{61}$. Generator point G = (4,25)

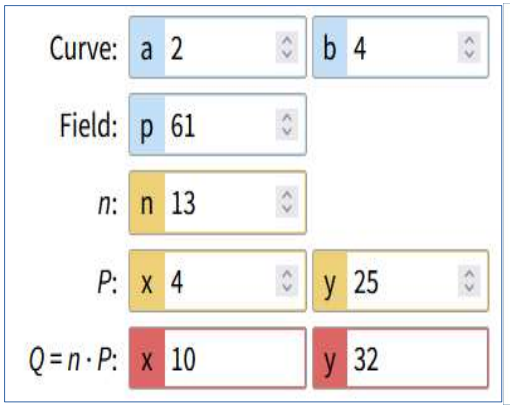
An online calulator for n*P is found at https://andrea.corbellini.name/ecc/interactive/modk-mul.html

Alice chooses private key a = 13 and Bob uses private key of 23. Calculate the shared key using ECDHE.

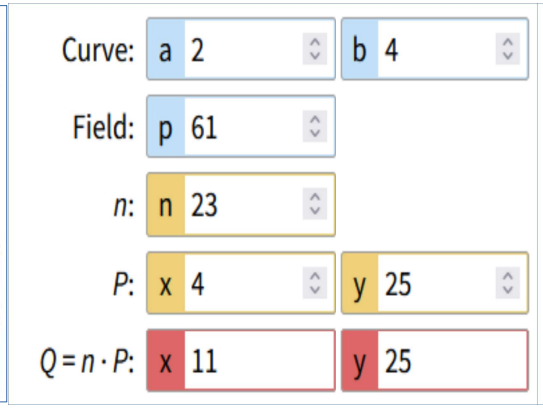
Alices: public key Ya = 13*(4,25) = (10,32), Bob's public key Yb =23*(4,25) = (11,25).
Shared key calculations: Alice calculates: K = 13*(11,25) =(10,29), Bob: K = 23*(10,32) = (10,29)

All needed calculations are shown in following screen captions of the online calculator

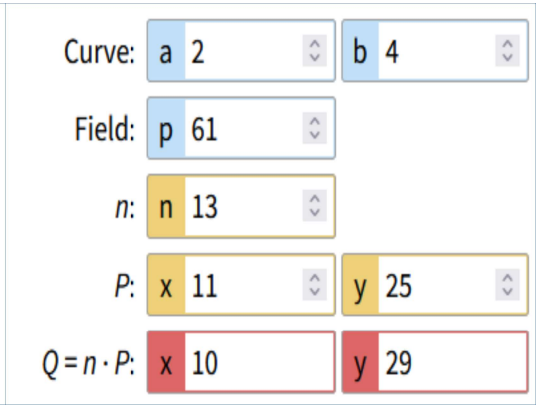
Pic1: 11*(4,25)



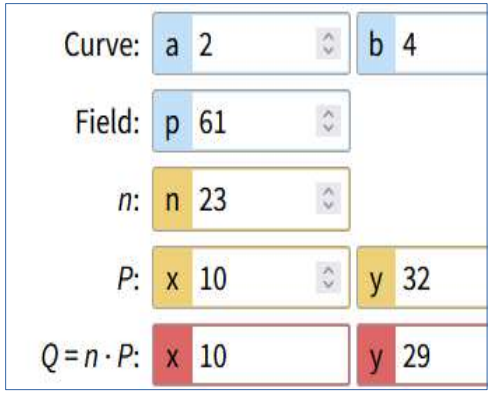
Pic2: 23*(4,25)



Pic3: 13*(11,25)



Pic4: 23*(10,32)



Instructions for the online calculator: Fill in the curve parameters a,b,p (p=prime modulus). Fill also n and point P's coordinates. Click somewhere outside the cell area to update the answer cells Q = n*P

Summary of concepts in section Key Exchange Protocols

Protocol is a standard procedure, 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 protocol used for agreeing on symmetric key between two parties.

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

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

Cryptographic hash functions

Message Authentication codes MAC’s

Digital signature

Hash functions

Definition: Hash functions are one-way functions, which produce a **fixed length digest** from the message. Hash values are also called ”message digests”

Uses of hash functions:

1.Hash values are used to **ensure the integrity of data transmission** **which means ensuring** that data is not changed during transmission.

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

3. **Digital signature** **is** calculated using hash value of message.

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

Requirements for a secure hash function

Hash function is denoted below as h(m), where argument m is message

1. **h(m)** is a **one-way function**. It is not 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 with same hash surely exist, 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.

Known hash functions

Chinese scientists showed in 2005 the vulnerability of many common hash functions. They found a method to generate another message with a same hash value as the original message.

- **MD5 - broken in 2005**
- **SHA-1 (160 bit) - not recommended**

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

Example: The sha256 hash of the message "It is tuesday evening" is in integer form 66234561002411177584498349418789448918599187388512797175649903214146302670 34 and hexadecimal form 0ea4 be49 0ee7 d159 4948 a6dd bb7d 64ca 3ade 2afb 5d27 1024 1043 c9db e1ef d89a

Secure hash length. ”Birthday paradox”.

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

Reason for requirement for double length 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

message authentication codes

- MAC = hash function with symmetric key K as extra input . The other input is message m

- MAC ensures not only the integrity of message, but also authenticates the sender of message

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

HMAC-SHA

Is the most common MAC which is calculated from message m
ja and symmetric key K using following formula

$$HMAC(K, m) = sha(K \oplus opad || sha(K \oplus ipad || m)$$

⊕ = XOR sum

K = symmetric key

Ipad and Opad are 64 bit constants, in hexadecimal form

Ipad = 5c5c5c5c , Opad = 36363636

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

HMAC-SHA is used as pseudorandom password generator in devices, which produce one-time passwords. There is more information about HMAC in the last chapter.

Digital signature

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

The goal of digital signature is to ensure that

1) document or message is not changed

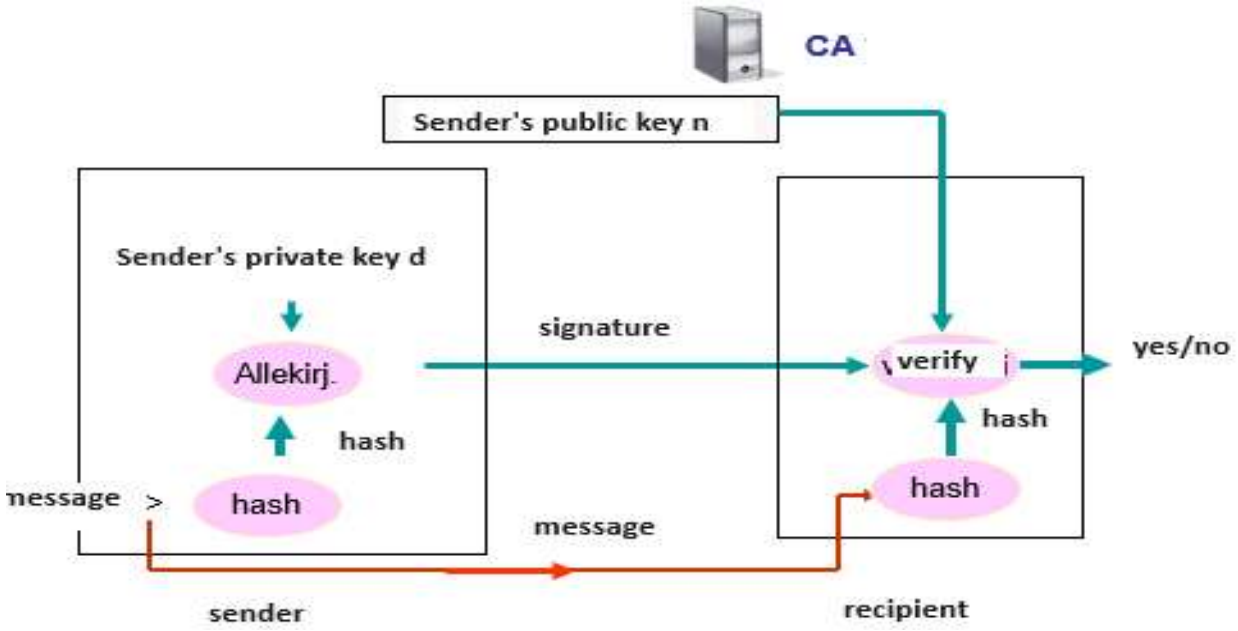
2) sender or author is authenticated

Digital signature algorithm usually uses hash function and public key encryption

Typical signature used in TLS protocol is **sha256RSA.**

Other signatures, for example ECDSA is also possible

Diagram of digital signature



Digital signature is a number, which is the hash value of message encrypted with senders private key.
The recipient receives a pair (message m, signature S). Usually pair (m,S) is encrypted with recipients public key.
In that case the recipient needs to decrypt the pair before verification process.

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**.

Calculation of sha256RSA signature

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

Signature S is calculated in two steps

1. Calculate the sha256 hash value of the message m
2. Encrypt the hash value with senders private RSA key d.

The result is the signature.

Calculated example of digital signature:
Alice sends Bob a message ”It is tuesday evening” with shaRSA digital signature.
Her RSA public key is n = 308911 and private key d = 133073.
Calculate the message-signature pair which she sends to Bob.

Step 1: Alice calculates the SHA256 hash of the message

WolframALpha command SHA256 "It is tuesday evening" gives

6623456100241117758449834941878944891859918738851279717564990321414630267034

Step2: Alice encrypts the has with her private key

In reality the public key n is much larger than 256 bit hash value. The verification of the signature does not work properly if hash value > public key os sender.
We change in this demonstration the hash value in such a way that we take only four first digits from the beginning: 6623.

Signature **S = 6623^133073 mod 308911 =138680**
and the sent pair is (”It is tuesday evening”, 138680)

How the recipient verifies the signature?

The recipient Bob has received the pair (m,S) = (”It is tuesday evening”,138680)

Step 1. Bob decrypts the signature with senders public key n = 308911 and public exponent e = 65537

Hash value decrypted from signature is $\mathbf{S^e \bmod n = 138680^{65537} \bmod 308911 = 6623}$

Step 2. Recipient calculates SHA of the received message directly with W.A
The first four digits of the result 66234561002... are 6623

SHA256 "It is tuesday evening"

Step 3. Comparison: **HASH values match => message is unchanged and sender authenticated**