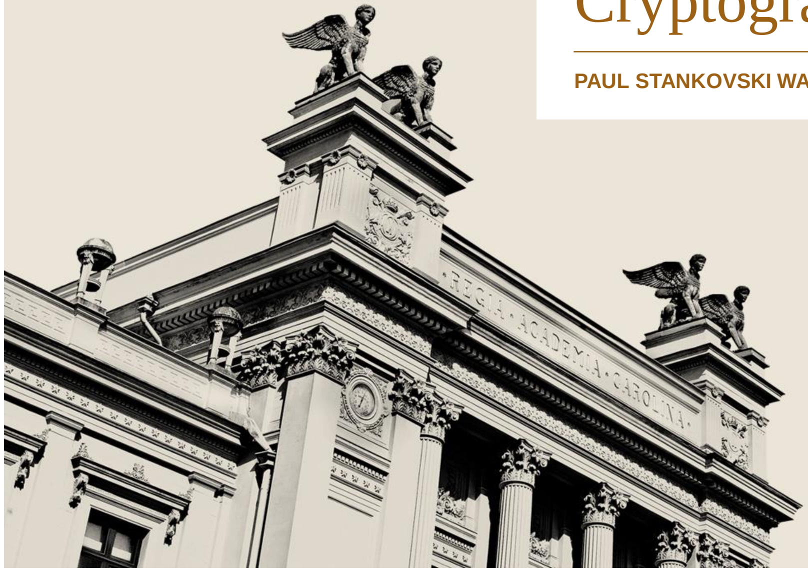
LUND UNIVERSITY

EITA25 Computer Security (Datasäkerhet)
Cryptography

PAUL STANKOVSKI WAGNER, EIT, 2020-01-23/24

©Martin Hell

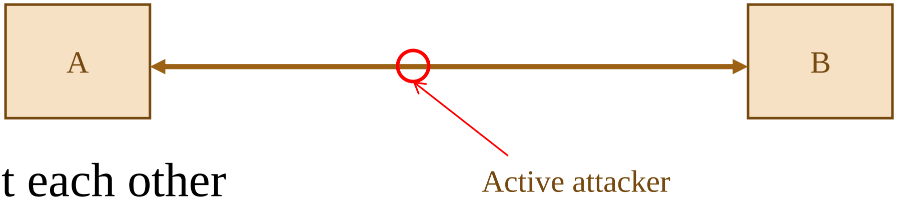


Cryptography

- Introduction to the basic concepts
- Define and see examples of
  - Stream ciphers
  - Block ciphers
  - Hash functions
  - Message authentication codes
  - Public key encryption
  - Digital signatures
  - Digital certificates

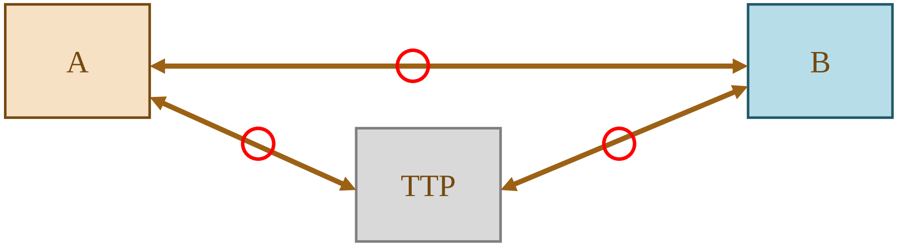
Classic Paradigm

- Insecure communication links



- A and B trust each other
  - Together they try to avoid attacks from outsiders
- Cryptography can give them
  - data confidentiality
  - data integrity
  - message authentication

New Paradigm



- The insiders have no reason to trust each other
- *Trusted Third Party* TTP
- *Nonrepudiation* services generate evidence for resolving a dispute

Cryptographic Keys

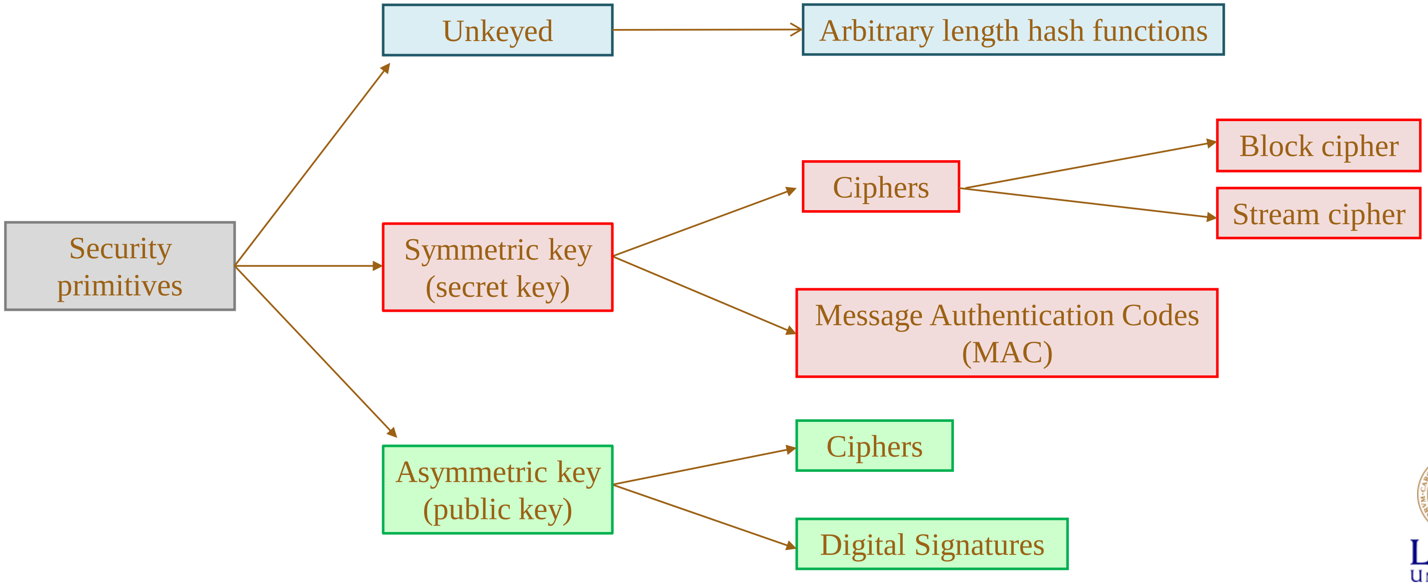
- Cryptographic algorithms use keys to protect data

Key management is the topic of addressing

- Where are keys generated?
- How are keys generated?
- Where are keys stored?
- How do they get there?
- Where are keys used?
- How are they revoked and replaced?

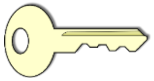
Cryptographic Primitives

Primitives that we will look at

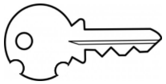

Security primitives
Unkeyed
Arbitrary length hash functions
Symmetric key (secret key)
Ciphers
Block cipher
Stream cipher
Message Authentication Codes (MAC)
Asymmetric key (public key)
Ciphers
Digital Signatures

Symmetric vs. Asymmetric Keys

Symmetric keys (Secret key cryptography)
Same key used for encryption and decryption



Asymmetric keys (Public key cryptography)
Different keys used for encryption and decryption
Encryption key is public
Decryption key is private



Compare with padlock

Example – Symmetric vs. Asymmetric

Symmetric keys (Secret key cryptography)
Same key used for encryption and decryption

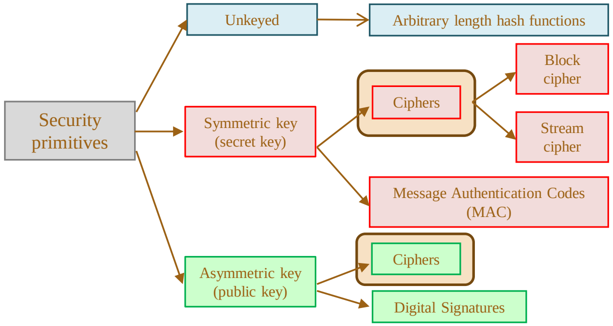
Asymmetric keys (Public key cryptography)
Different keys used for encryption and decryption
Encryption key is public
Decryption key is private

Compare with padlock

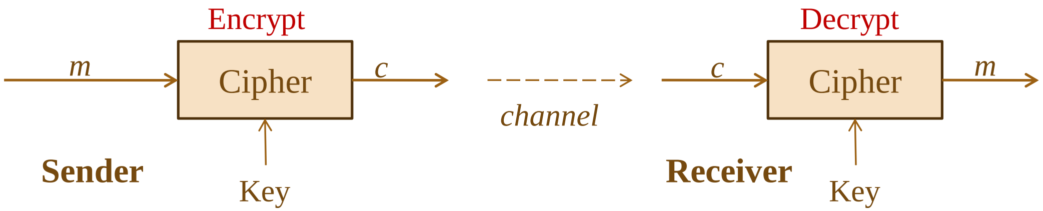
Strength of Encryption Mechanisms

- Empirically secure – Secure based on the fact that no one has broken it for some time.
  - Most common for practically used symmetric primitives and hash functions
  - Typically very efficient
- Provably secure – We prove that breaking a scheme is at least as hard as breaking some well known problem like factoring or discrete log.
  - Most common for asymmetric primitives
  - Also possible for symmetric primitives (but we do not consider those in this course)
- Unconditionally secure – The schemes are secure even if the adversary has unlimited computing power
  - Not common but possible

Plaintext and Ciphertext (Ciphers)


Security primitives
Unkeyed
Arbitrary length hash functions
Symmetric key (secret key)
Ciphers
Block cipher
Stream cipher
Message Authentication Codes (MAC)
Asymmetric key (public key)
Ciphers
Digital Signatures

- The plaintext is the message we want to send
  - We denote it by *m*
- The ciphertext is the data that we actually send
  - We denote it by *c*



Simplified model (without source coding, channel coding, modulation etc.)

Attack Scenarios

- Kerckhoffs' principle:
  - Only the key should be unknown to an adversary
    - Security should not be based on the fact that the algorithm is secret, WHY?
  - Formulated in the 19th century and is for different reasons still sometimes ignored in the 21th century
- A scheme can be analysed under different scenarios
  - Ciphertext only attack (COA)
  - Known plaintext attack (KPA)
  - Chosen plaintext attack (CPA)
  - Chosen ciphertext attack (CCA)
- All scenarios implicitly assume Kerckhoffs' principle
- **Primary attack goal:** Find the secret key
  - However, other goals can be imagined as well

Symmetric Key Cryptography

Some old cryptographic tools

Scytale

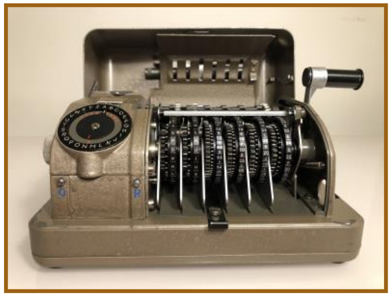
Jefferson’s disk

Enigma

Symmetric Key Cryptography

Some Swedish cryptographic machines

HC-9
AB Transvertex



C-52
Boris Hagelin
Crypto AG

Very Simple Symmetric Schemes (motivate stream ciphers)

We will assume that all keys are chosen from a uniform distribution!

Shift cipher (Caesar cipher)

Plaintext	A	B	C	D	E	F	...	X	Y	Z
Ciphertext	D	E	F	G	H	I	...	A	B	C

Map letter to number, then

Plaintext	0	1	2	3	4	5	...	23	24	25
Ciphertext	3	4	5	6	7	8	...	0	1	2

$$c_t = m_t + 3$$

$$m_t = c_t - 3$$

Key is ”3” (or ”D”)

Problems:

- ✗ Only 26 keys
- ✗ Redundancy in language is preserved

Substitution Ciphers

Substitution cipher

Define a permutation over the alphabet:

Plaintext	A	B	C	D	E	F	…	X	Y	Z
Ciphertext	S	H	D	T	V	B	…	Q	A	O

Table is the key

Problems:

- ✓ Only 26 keys (There are now 26!)
- ✗ Redundancy in language is preserved

Vigenère cipher

Use a shift cipher, but different shifts for n consecutive letters

0					
A	B	C	…	Y	Z
F	G	H	…	D	E

1					
A	B	C	…	Y	Z
T	U	V	…	R	S

....

n - 1					
A	B	C	…	Y	Z
M	N	O	…	K	L

Letter *t* in message of length *N* is encrypted with table *t (mod n)*

Key is sequence of *n* numbers (or letters)

Problems:

- ✓ Only 26 keys (There are now 26^n)
- ✓ / ✗ Redundancy in language is preserved (n different probability distributions)

The One-Time-Pad (OTP)

- Substitution cipher and Vigenere cipher can be broken with statistics since the underlying language has redundancy!
  - Note that we are talking about a ciphertext only attack
- But what if $n=N$ in a Vigenere cipher? (Length of key is the same as message length)
- Then it is UNBREAKABLE!
- This is called Vernam cipher or One-Time-Pad (OTP)
- Perfect secrecy (unconditionally secure)

Problems:

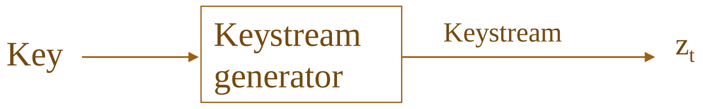
✓ Only 26 keys (There are now 26^N)

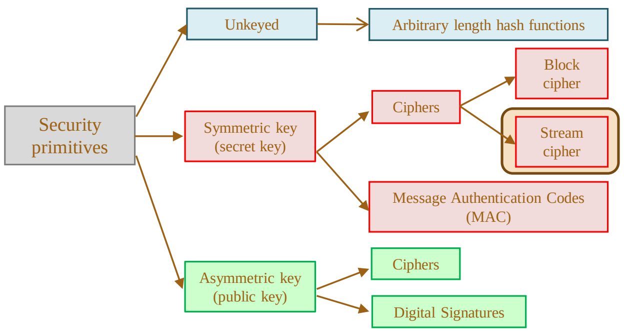
✓ Redundancy in language is preserved (No redundancy at all)

- Secure since number of possible keys is the same as number of possible messages. New problem!

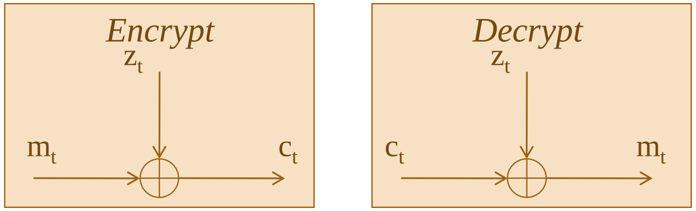
Stream Ciphers

- **A good idea:** Take a short random key and expand it to a long (pseudo)random sequence of bits
- That is a stream cipher!




Security primitives
Unkeyed
Arbitrary length hash functions
Symmetric key (secret key)
Ciphers
Block cipher
Stream cipher
Message Authentication Codes (MAC)
Asymmetric key (public key)
Ciphers
Digital Signatures

Binary additive stream cipher

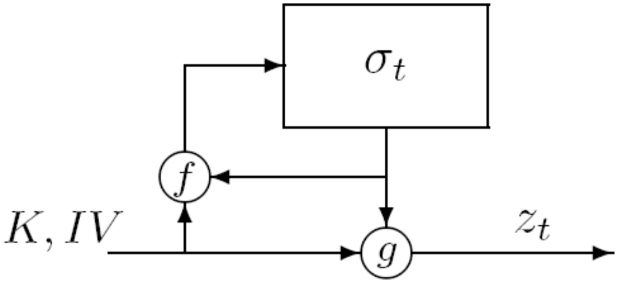


xor function

a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0

$$m_t \oplus \underbrace{z_t \oplus z_t}_{=0} = m_t$$

Inside the Keystream Generator

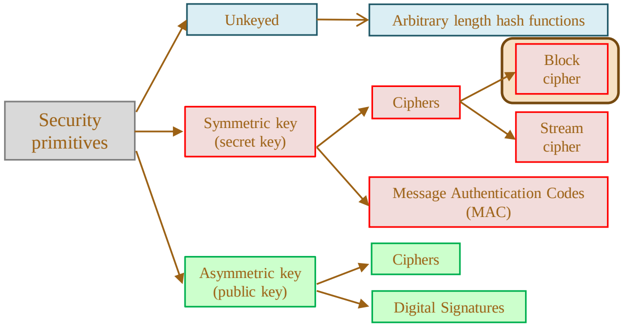


$\sigma_0 = \gamma(K, IV)$	Initialisation function
$\sigma_{t+1} = f(\sigma_t, K, IV)$	State update function
$z_t = g(\sigma_t, K, IV)$	Output function

- IV (Initialization Vector)
  - Allows reuse of key
  - Must be unique for each encryption with same key
  - Always assumed to be known to everyone
- State can be: shift register, large table, counter etc
- Well-known stream ciphers: RC4, SNOW, A5/1, E0, Salsa20, ChaCha20

LUND UNIVERSITY

Block Ciphers


Security primitives
Unkeyed
Arbitrary length hash functions
Symmetric key (secret key)
Ciphers
Block cipher
Stream cipher
Message Authentication Codes (MAC)
Asymmetric key (public key)
Ciphers
Digital Signatures

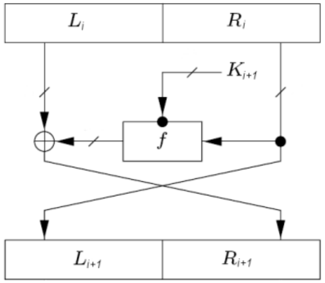
- Return to substitution cipher

Plaintext	**A**	**B**	**C**	**D**	**E**	**F**	**…**	**X**	**Y**	**Z**
Ciphertext	**S**	**H**	**D**	**T**	**V**	**B**	**…**	**Q**	**A**	**O**

- Substitution cipher is a block cipher
  - Still, redundancy is a problem
  - Block length too small → full table (key) is easily recovered if some plaintext is known
- Increase block size to e.g., 64, 128, 192 or 256 bits
  - Now table is too large to fit in memory
- Solution: Use mathematical tools to map plaintext symbols to ciphertext symbols (and back)!
  - Still preserved redundancy, but we will solve that soon...

Two Variants of Block Cipher Design Ideas

Feistel structure

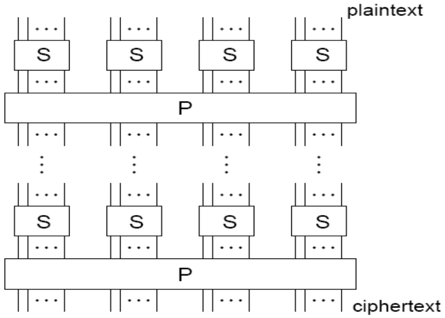


Encrypt
$$\begin{cases} L_{i+1} = R_i \\ R_{i+1} = L_i \oplus f(K_{i+1}, R_i) \end{cases}$$

Decrypt
$$\begin{cases} R_i = L_{i+1} \\ L_i = R_{i+1} \oplus f(K_{i+1}, L_{i+1}) \end{cases}$$

Decryption can be done using the same structure, but with keys in reverse order

Substitution Permutation Network (SP-network)

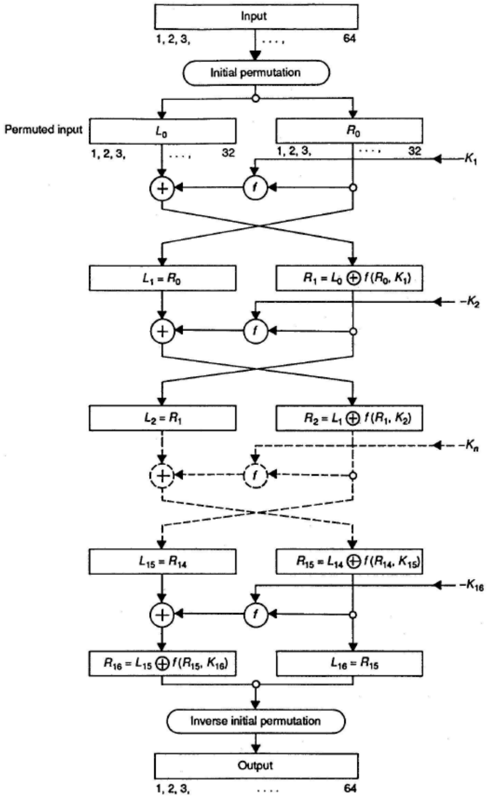


- Repeated substitutions and permutations
- Confusion and diffusion
- Go backwards to decrypt

Feistel Cipher: DES

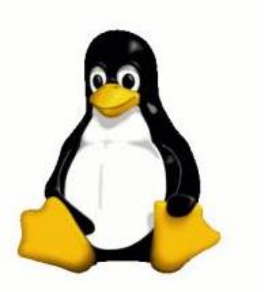
- Block size: 64 bits
- 16 rounds
- Key size: 56 bits
- Can be ”broken” in a day or so
- Standard 1977 – 1998
- 1998 – 2002: 3DES

AES has been standard since 2002 and is an example of a SP-network

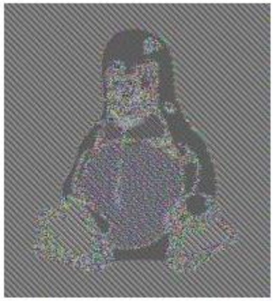


Modes of Operation – ECB

- Electronic code book mode (ECB)
  - $c_i = eK(m_i)$
  - $m_i = dK(c_i)$
- All blocks encrypted independently of each other



Original

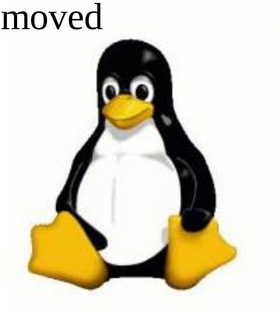


Encrypted with ECB mode

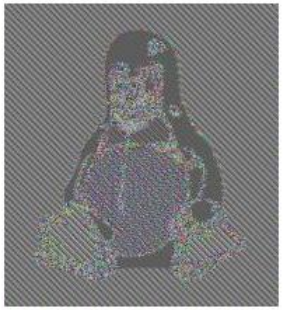
- Redundancy preserved!

Modes of Operation – CBC

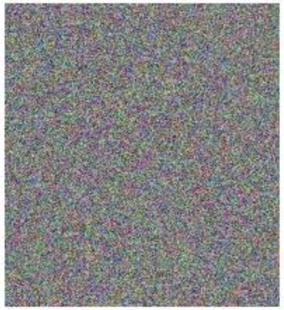
- Cipher Block Chaining (CBC)
  - $c_i = eK(m_i \oplus c_{i-1}),\ c_{-1} = IV$
  - $m_i = dK(c_i) \oplus c_{i-1}$
- Redundancy removed



Original



Encrypted with ECB mode



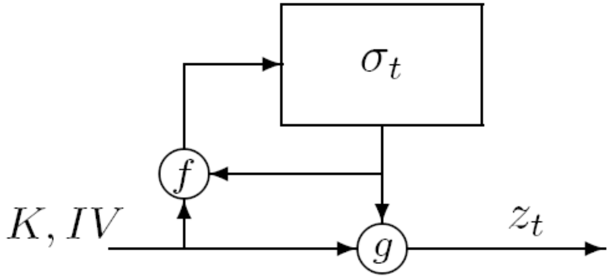
Encrypted with CBC mode

Modes of Operation – OFB

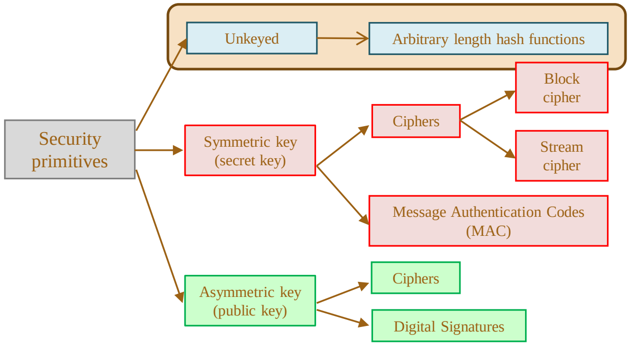
- Output feedback mode
  - Turns the block cipher into a stream cipher
  - $z_t = eK(z_{t-1}),\ z_{-1} = IV$
  - $c_t = m_t \oplus z_t$
  - $m_t = c_t \oplus z_t$

Advanced state update function *f*, but very simple keystream generation function *g*.
Counter mode has opposite property.

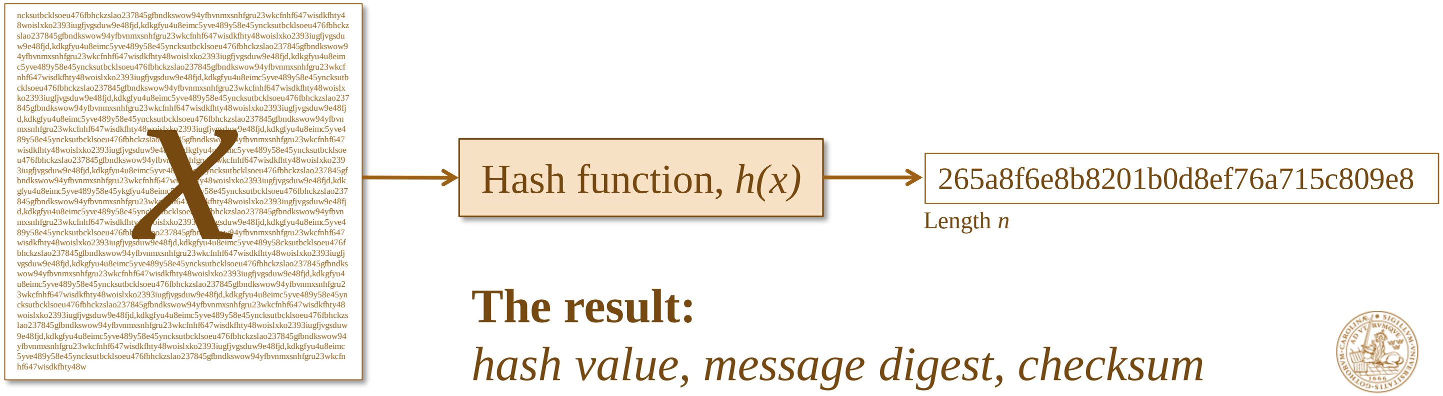
Stream cipher model



Hash Functions


Security primitives
Unkeyed
Arbitrary length hash functions
Symmetric key (secret key)
Ciphers
Block cipher
Stream cipher
Message Authentication Codes (MAC)
Asymmetric key (public key)
Ciphers
Digital Signatures

- Defining properties
  - *Ease of computation:* Easy to compute *h(x)*
  - *Compression:* *x* of arbitrary bit length maps to fixed length *n* output.



The result:

hash value, message digest, checksum

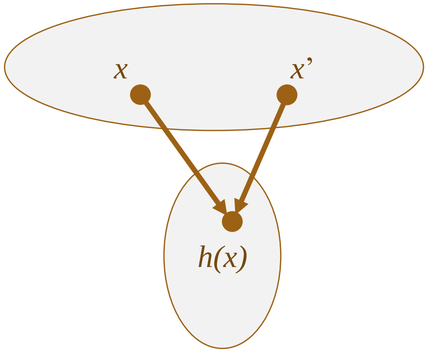
Hash Functions, properties

- Additional properties
  - *Preimage resistance:* given *y* it is in general infeasible to find *x* such that *h(x)* = *y*.
    - Also called one-way
  - *Second preimage resistance:* given *x, h(x)* it is infeasible to find *x'* such that *h(x)* = *h(x')*.
    - Also called weak collision resistance
  - *Collision resistance:* it is infeasible to find *x, x'* such that *h(x)* = *h(x').*
    - Also called strong collision resistance

Birthday Paradox

How many people do you need to be in a room such that the probability that two have the same birthday (month and day) is > 0.5?

Collision



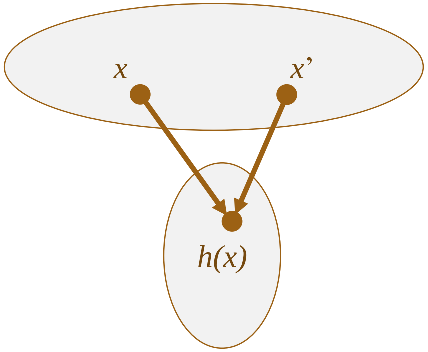
Possible outcomes: 2^n

Expected number of trials before collision with *one given* $y = h(x)$ is 2^n. *(Not Birthday paradox)*

Expected number of trials before collision with *any* previously observed $y = h(x)$ is approximately $2^{n/2}$. *(Birthday paradox)*

Birthday Paradox – Consider Implementing

Collision



Possible outcomes: 2^n

Expected number of trials before collision with *one* given $y = h(x)$ is 2^n. *(Not Birthday paradox)*

Expected number of trials before collision with *any* previously observed $y = h(x)$ is approximately $2^{n/2}$. *(Birthday paradox)*

Common Hash Functions

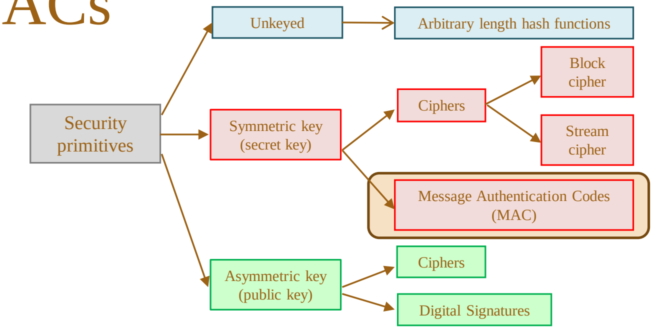
- MD5
  - 128-bit output
  - Very common when checking downloaded files
  - Often used to save passwords on www
  - Broken – should not be used
  - In theory, about 2^{64} messages before we have a collision
  - Collisions can be found within a minute

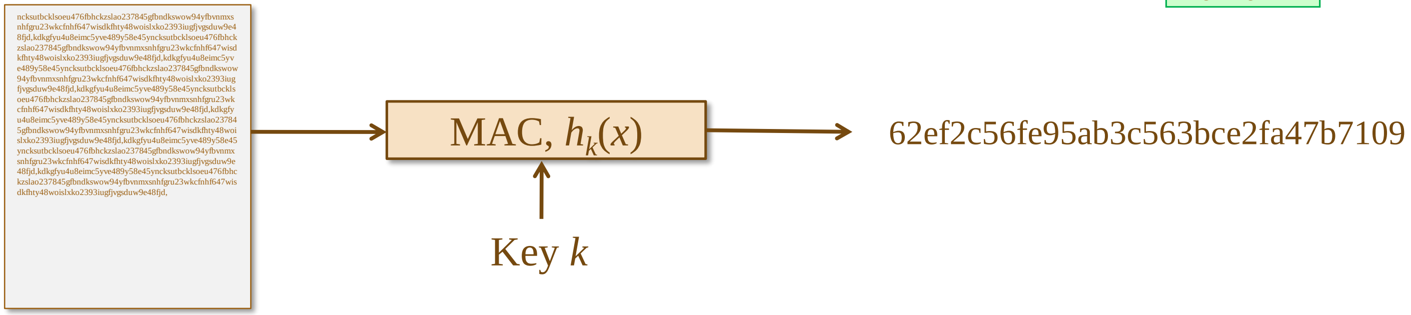
- SHA-256, SHA-3
  - Not broken
  - These should be used

- SHA-1
  - 160-bit output
  - (Previously) common in many applications (TLS, certificates, checksums)
  - Broken - Theoretically in 2005, practically in 2017.
  - In theory, about 2^{80} messages before we have a collision
  - Weakness shows that we need only about $2^{63.1}$ (6500 CPU years in 2017 attack)
  - Best attack (2020-01-05) by G. Leurent and T. Peyrin, chosen prefix attack with complexity $2^{63.4}$, estimated cost $45,000 per collision

Message Authentication Codes, MACs

- *Keyed hash functions*
- Computed from two inputs, message and a key
- *Message authentication codes* prove the integrity of a message (source)


Security primitives
Unkeyed
Arbitrary length hash functions
Symmetric key (secret key)
Ciphers
Block cipher
Stream cipher
Message Authentication Codes (MAC)
Asymmetric key (public key)
Ciphers
Digital Signatures



ncksutbcklsoeu476fbhckzslao237845gfbndkswow94yfbvnmxs
nhfgru23wkcfnhf647wisdkfhty48woislxko2393iugfjvgsduw9e4
8fjd,kdkgfyu4u8eimc5yve489y58e45yncksutbcklsoeu476fbhck
zslao237845gfbndkswow94yfbvnmxsnhfgru23wkcfnhf647wisd
kfhty48woislxko2393iugfjvgsduw9e48fjd,kdkgfyu4u8eimc5yv
e489y58e45yncksutbcklsoeu476fbhckzslao237845gfbndkswow
94yfbvnmxsnhfgru23wkcfnhf647wisdkfhty48woislxko2393iug
fjvgsduw9e48fjd,kdkgfyu4u8eimc5yve489y58e45yncksutbckls
oeu476fbhckzslao237845gfbndkswow94yfbvnmxsnhfgru23wk
cfnhf647wisdkfhty48woislxko2393iugfjvgsduw9e48fjd,kdkgfy
u4u8eimc5yve489y58e45yncksutbcklsoeu476fbhckzslao23784
5gfbndkswow94yfbvnmxsnhfgru23wkcfnhf647wisdkfhty48woi
slxko2393iugfjvgsduw9e48fjd,kdkgfyu4u8eimc5yve489y58e45
yncksutbcklsoeu476fbhckzslao237845gfbndkswow94yfbvnmx
snhfgru23wkcfnhf647wisdkfhty48woislxko2393iugfjvgsduw9e
48fjd,kdkgfyu4u8eimc5yve489y58e45yncksutbcklsoeu476fbhc
kzslao237845gfbndkswow94yfbvnmxsnhfgru23wkcfnhf647wis
dkfhty48woislxko2393iugfjvgsduw9e48fjd,

Message	MAC

MAC Properties

- Defining properties
  - *Ease of computation* – Given k and x, $h_k(x)$ is easy to compute.
  - *Compression* – $h_k(x)$ maps x of arbitrary bit length to fixed length n output.
  - *Computation resistance* – given zero or more pairs $(x_i, h_k(x_i))$, it is infeasible to compute another pair $(x, h_k(x))$ with a new message x (without knowing the key).
- Does NOT provide encryption. That has to be added separately!

MAC Example

- HMAC makes a MAC from a hash function.

$$\mathrm{HMAC}(m) = h(\ k \oplus p_1 \ \| \ h((\ k \oplus p_2\ ) \ \| \ m\ ))$$

- A simpler construction like $h(k \| x)$ is insufficient for many hash functions.
- A MAC can also be constructed from a block cipher.

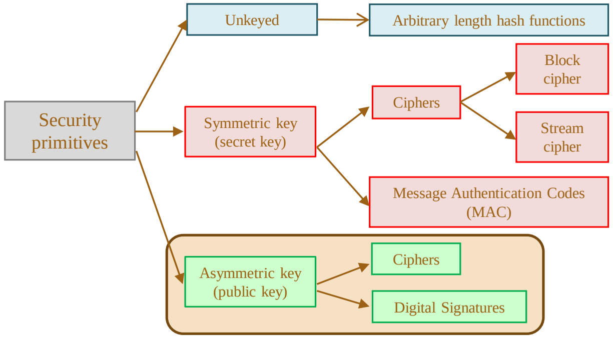
- *Limitation of MACs*:

  Transmitter and receiver share the same key *k*.

  Not possible to resolve internal disputes.

  Does not provide nonrepudiation.

Public Key Cryptography


Security primitives
Unkeyed
Arbitrary length hash functions
Symmetric key (secret key)
Ciphers
Block cipher
Stream cipher
Message Authentication Codes (MAC)
Asymmetric key (public key)
Ciphers
Digital Signatures

- Also called asymmetric cryptography
- Encryption
  - Public key used to encrypt
  - Private key used to decrypt
- Digital Signatures
  - Public key used for verification
  - Private key used for signing
- Note the terminology!
  - *Secret key* used in symmetric algorithms
  - *Public key* and *private key* used in asymmetric algorithms
    - Private key is sometimes also called secret key

Some Mathematics Before We Move On

Modular arithmetic:

- $a \equiv b \bmod n$ if and only if $a - b = k \cdot n$ for some integer k
- $a \equiv b \bmod n$ if and only if $a = k \cdot n + b$ for some integer k

Properties:

- $(a \bmod n) + (b \bmod n) \equiv (a + b \bmod n)$
- $(a \bmod n) \cdot (b \bmod n) \equiv (a \cdot b \bmod n)$
- for every a $\neq 0 \bmod p$, p prime, there exists an integer such that a^{-1} such that

$$a \cdot a^{-1} \equiv 1 \bmod \mathrm{p}$$

gcd(a,b) is the greatest common divisor of *a* and *b*

- More generally: There exists an integer a^{-1} such that $a \cdot a^{-1} \equiv 1 \bmod \mathrm{p}$, if and only if $gcd(a, n) = 1$.

Examples

a) $32 \equiv 6 \bmod 13 \text{ since } 32 - 6 = 2 \cdot 13$

b) $60 \bmod 13 \equiv (20 \bmod 13) + (40 \bmod 13) \equiv 7 + 1 \bmod 13 \equiv 8 \bmod 13$

c) $2^{10} \bmod 13 \equiv (2^5 \bmod 13) \cdot (2^5 \bmod 13) \equiv 6 \cdot 6 \bmod 13 \equiv 10 \bmod 13$

d) $8^{-1} \bmod 13 \equiv 5 \bmod 13 \text{ since } 8 \cdot 5 \equiv 1 \bmod 13$

e) $8^{-1} \bmod 12 \text{ does not exist since } \gcd(8,12) = 4 \neq 1$

Computing Modular Inverses

There exists an integer a^{-1} such that $a \cdot a^{-1} \equiv 1 \bmod p$, if and only if $gcd(a, n) = 1$.

Example:

What is $1337^{-1} \bmod 31337$?

$gcd(1337, 31337) = 1$, so 1337 has an inverse modulo 31337.

```
31337 = 23 · 1337 + 586   | 31337 - 23 · 1337 = 586
 1337 =  2 ·  586 + 165   |  1337 -  2 ·  586 = 165
  586 =  3 ·  165 +  91   |   586 -  3 ·  165 =  91
  165 =  1 ·   91 +  74   |   165 -  1 ·   91 =  74
   91 =  1 ·   74 +  17   |    91 -  1 ·   74 =  17
   74 =  4 ·   17 +   6   |    74 -  4 ·   17 =   6
   17 =  2 ·    6 +   5   |    17 -  2 ·    6 =   5
    6 =  1 ·    5 +   1   |     6 -  1 ·    5 =   1
    5 =  5 ·    1
```

Computing Modular Inverses

There exists an integer a^{-1} such that $a \cdot a^{-1} \equiv 1 \bmod p$, if and only if $gcd(a, n) = 1$.

Example:

What is $1337^{-1} \bmod 31337$?

$gcd(1337, 31337) = 1$, so 1337 has an inverse modulo 31337.

$$5508 \cdot 1337 - 235 \cdot 31337 = 1$$

$$5508 \cdot 1337 = 1 + 235 \cdot 31337$$

$$1337^{-1} = 5508$$

$$\begin{array}{rcrcrcr}
31337 & - & 23 & \cdot & 1337 & = & 586 \\
1337 & - & 2 & \cdot & 586 & = & 165 \\
586 & - & 3 & \cdot & 165 & = & 91 \\
165 & - & 1 & \cdot & 91 & = & 74 \\
91 & - & 1 & \cdot & 74 & = & 17 \\
74 & - & 4 & \cdot & 17 & = & 6 \\
17 & - & 2 & \cdot & 6 & = & 5 \\
6 & - & 1 & \cdot & 5 & = & 1
\end{array}$$

More Mathematics

- Euler phi function: $\varphi(n)$ is the number of integers < n that are coprime to n

$\varphi(p^k) = p^k - p^{k-1}$ if p is prime,
$\varphi(mn) = \varphi(m)\varphi(n)$ if m and n are are coprime

- Euler's Theorem:

If a and n are coprime, then
$$a^{\varphi(n)} \equiv 1 \bmod n$$

More Examples

a) $\phi(13) = 12$

b) $\phi(17) = 16$

c) $\phi(221) = \phi(13 \cdot 17) = \phi(13) \cdot \phi(17) = 12 \cdot 16 = 192$

d) $\phi(12) = \phi(4) \cdot \phi(3) = (2^2 - 2)(3 - 1) = 4$

e) $a^{12} \equiv 1 \bmod 13$ for all *a* that are not multiples of 13

f) $a^{192} \equiv 1 \bmod 221$ for all *a* such that gcd(*a,221*)=*1*

More Mathematics

Let p be a prime and a an arbitrary (nonzero) integer.

- The *multiplicative order* of the element a modulo p is defined to be the smallest integer j such that $a^j = 1 \bmod p$.

- Fermat's little theorem:

For all a $\neq 0$, p prime,
$a^{p\text{-}1} \equiv 1 \bmod p$

- The order of an element divides $p - 1$.

(Classical) Public Key Cryptography

- Usually based on one of two mathematical problems
  - Factoring – Given an integer *n*, find the prime factors.
  - Discrete Logarithm Problem (DLP) – Given a prime *p* and integers *a* and *y*, find *x* such that $y \equiv a^x \bmod p$.

- This gives provable security
- Other mathematical problems can be used
  - Modern Public Key Cryptography
  - Post-Quantum Cryptography

RSA Encryption, Parameters

Provably secure, based on the problem of factoring

- Pick primes p, q. Let $n = p \cdot q$ and compute

$$\varphi(n) = (p - 1)(q - 1)$$

- Pick an integer e such that

$$gcd(e, \varphi(n)) = 1$$

- Find d such that

$$e \cdot d \equiv 1 \bmod \varphi(n)$$

- Public key: e, n
- Private key: d, $\varphi(n)$, p, q

RSA Encryption

Encrypt: $c = m^e \bmod n$

Decrypt: $m = c^d \bmod n$

Proof that it works:

$$c^d = m^{ed} = m^{k\varphi(n)+1} = (m^{\varphi(n)})^k m = 1^k m \equiv m \bmod n$$

Note that only *d* and *n* is needed in decryption. However, in practice *p* and *q* are used to speed up decryption using the chinese remainder theorem. (Not included in course)

Security of RSA (factoring)

If we can factor the public value *n*, we will get *p* and *q* and can easily find *d* → RSA would be broken

How easy is it to factor large numbers?

- Aug 1999: 512-bits number was factored
- May 2005: 663-bit number was factored
- December 2009: A 768-bit number was factored (1500 core years)
  - Single core 2.2GHz AMD Opteron, 2GB RAM would need 1500 years
  - Of course hundreds of computers were used instead, so it took about two years
- December 2019: A 795-bit number was factored
  - 900 core years
  - 2.25 times harder but 3 times faster than in 2009

Estimated that factoring 1024-bit numbers are 1000 times harder – will be possible within 10 years with similar computing effort

Note: Finding *d* is equivalent to factoring, but breaking RSA (decrypting) might be easier than factoring

- With quantum computers, factoring is easy → Post-quantum cryptography

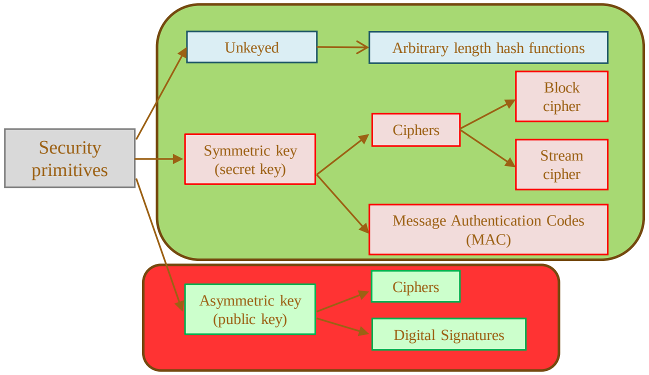
Post-Quantum Cryptography

Cryptography that is difficult to break even if an adversary has a (large) quantum computer.
But still efficiently computed on classical computers.

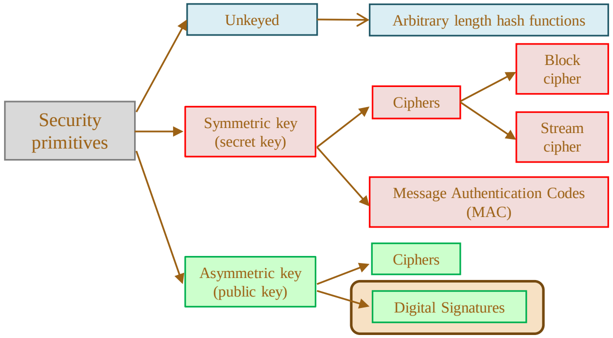
Ongoing NIST competition.
Primitives built from HARD problems.

Two algorithms:

- Shor’s algorithm
- Grover’s algorithm


Security primitives
Unkeyed
Arbitrary length hash functions
Symmetric key (secret key)
Ciphers
Block cipher
Stream cipher
Message Authentication Codes (MAC)
Asymmetric key (public key)
Ciphers
Digital Signatures

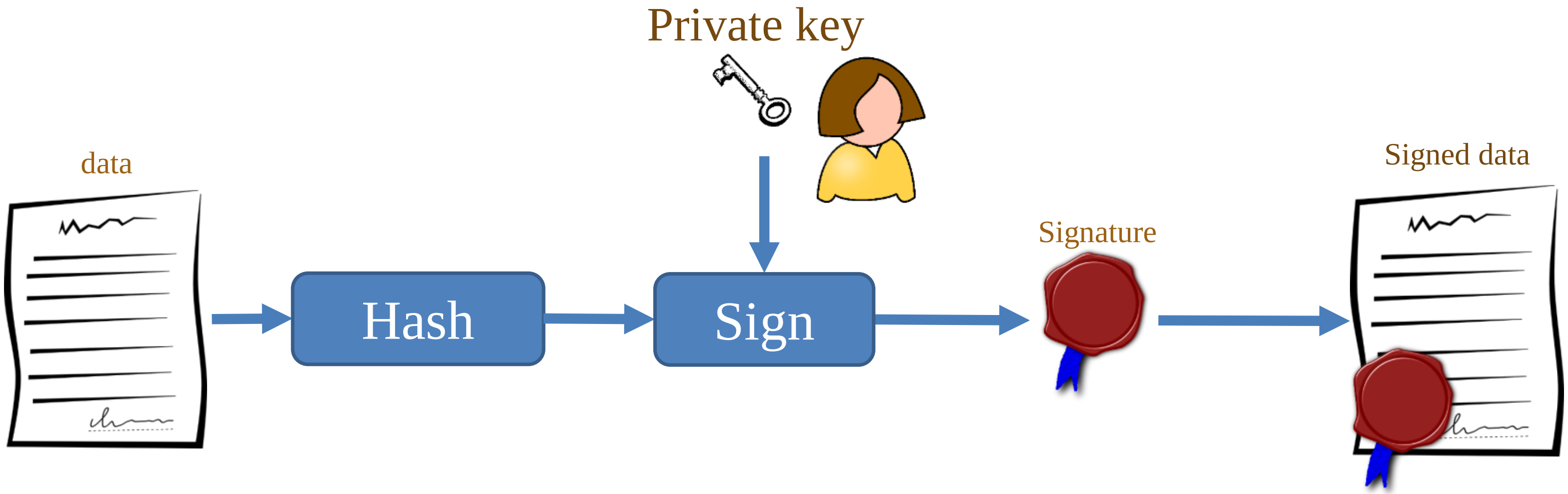
Digital Signatures


Security primitives
Unkeyed
Arbitrary length hash functions
Symmetric key (secret key)
Ciphers
Block cipher
Stream cipher
Message Authentication Codes (MAC)
Asymmetric key (public key)
Ciphers
Digital Signatures

- Scheme consists of
  - Key generation algorithm
  - Signature algorithm
  - Verification algorithm
- Private signature key, Public verification key
- Does NOT provide encryption. That has to be added separately!
- Provides nonrepudiation. A MAC does not!

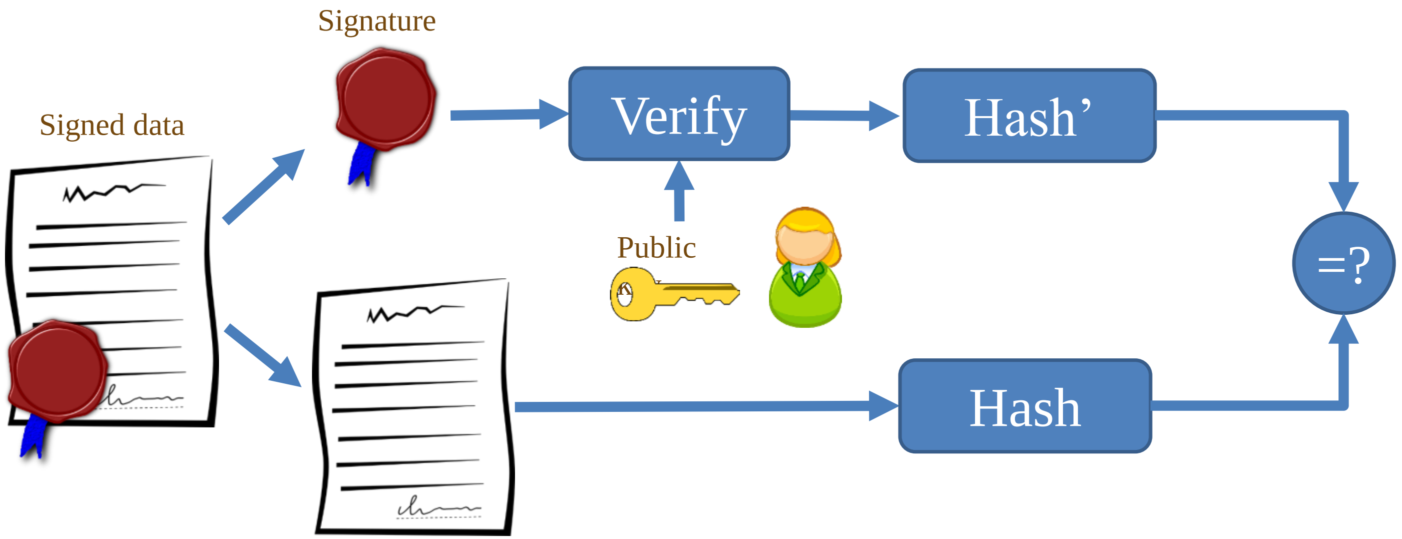
A third party can resolve disputes about the validity of a signature without the signer's private key

Signing a Document



With data and the private key, a signature can be computed

Verifying a Signature



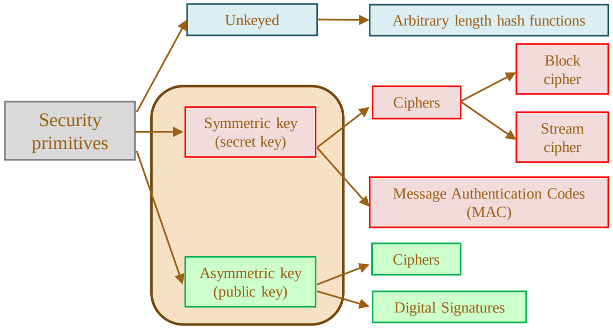
With data, signature and public key, a signature can be verified

RSA Signatures

- Key generation same as in RSA encryption
- Public verification key: n, e
- Private signing key: d, p, q,
- Signing: Hash message M: $m=h(M)$ and then sign by
  $$s = m^d \bmod n.$$
- Verification: Check if
  $$s^e = m \bmod n$$
- Property: We can select public e to be small (e.g. $e=3$ or $e=2^{16}+1$). This allows fast verification, but signing will be slow.

Comparing Symmetric and Asymmetric Algorithms

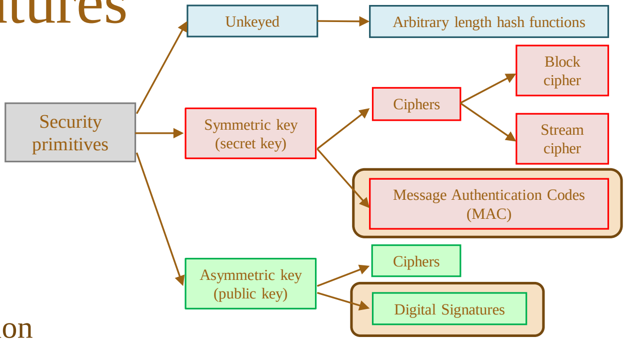
- Symmetric algorithms are generally *much* faster than asymmetric algorithms. About a factor of 1000.


Unkeyed
Arbitrary length hash functions
Security primitives
Symmetric key (secret key)
Ciphers
Block cipher
Stream cipher
Message Authentication Codes (MAC)
Asymmetric key (public key)
Ciphers
Digital Signatures

- Symmetric algorithms can use shorter key with same security. 1024-bit RSA modulus corresponds to about 80-bit symmetric key.

- Elliptic curves are often used to make public key cryptography more efficient. Both shorter keys and faster algorithms are possible.

Comparing MAC and Digital Signatures


Security primitives
Unkeyed
Arbitrary length hash functions
Symmetric key (secret key)
Ciphers
Block cipher
Stream cipher
Message Authentication Codes (MAC)
Asymmetric key (public key)
Ciphers
Digital Signatures

Message Authentication Codes

- Message authentication
- Integrity

- Symmetric cryptography
- Fast
- Need pre-shared key
- Holders of secret key can sign and verify

Digital Signatures

- Message authentication
- Integrity
- Nonrepudiation

- Asymmetric cryptography
- Slow
- Need digital certificates
- One can sign, all can verify

Digital Certificates

Public key cryptography:

- Alice has a key pair, one private key and one public key.
- Alice can *sign messages using her private key* and some redundancy in the message (hash value). Anyone can verify the signature using her public key.
- Anyone can *send encrypted messages to Alice using Alice's public key*. Only Alice can decrypt using her private key.
- **Problem:** We need to make sure that the public key we are using really belongs to Alice. Otherwise
  - We may verify a forged signature, thinking it is genuine
  - We may encrypt sensitive data allowing an adversary to decrypt it
- **Solution:** Certificates

Not much different from a driver's license

Certificates

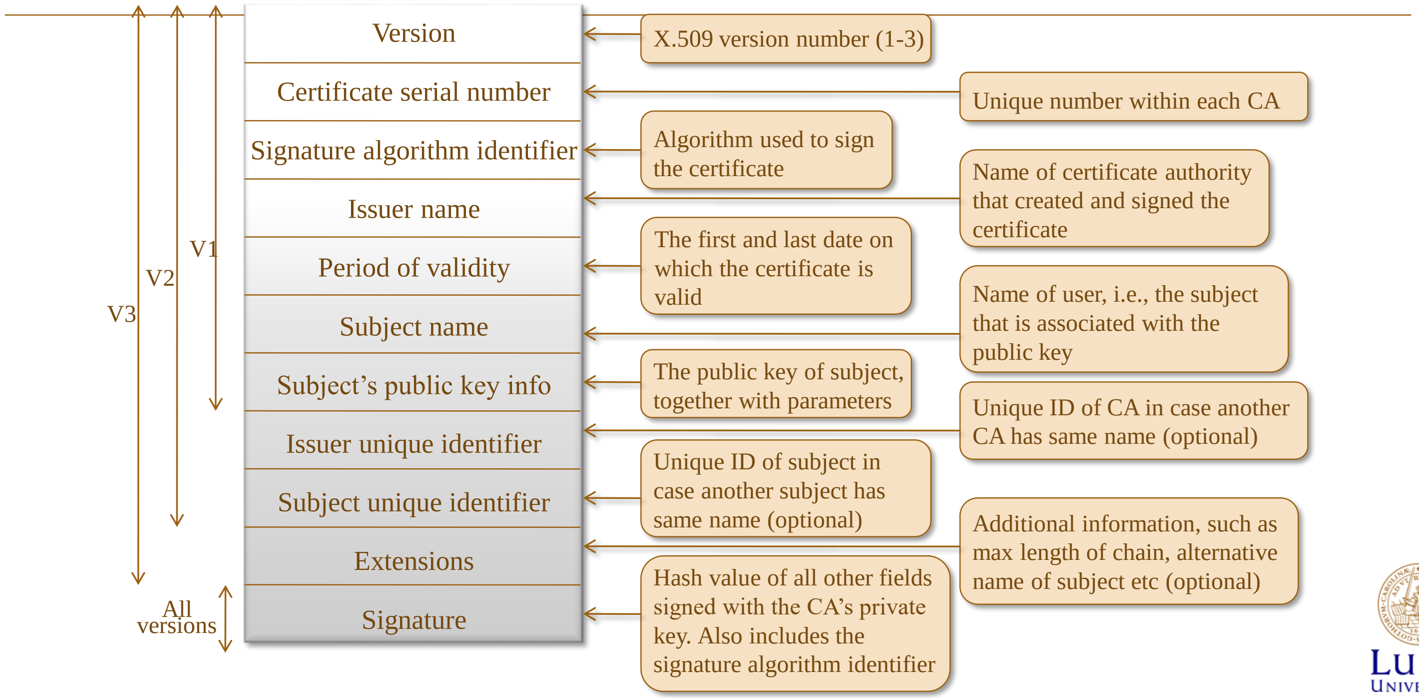
- Primarily binds a subject name to a public key, but can also contain other information such as authorization
- Information is signed by a Certification Authority (CA)
- If CA is trusted, then we trust the binding between user and public key

Public Key Infrastructure

> The set of hardware, software, people, policies and procedures needed to create, manage, store, distribute and revoke digital certificates based on asymmetric cryptography

RFC 4949, Internet Security Glossary

X.509 Certificates



Version ← X.509 version number (1-3)

Certificate serial number ← Unique number within each CA

Signature algorithm identifier ← Algorithm used to sign the certificate

Issuer name ← Name of certificate authority that created and signed the certificate

Period of validity ← The first and last date on which the certificate is valid

Subject name ← Name of user, i.e., the subject that is associated with the public key

Subject's public key info ← The public key of subject, together with parameters

Issuer unique identifier ← Unique ID of CA in case another CA has same name (optional)

Subject unique identifier ← Unique ID of subject in case another subject has same name (optional)

Extensions ← Additional information, such as max length of chain, alternative name of subject etc (optional)

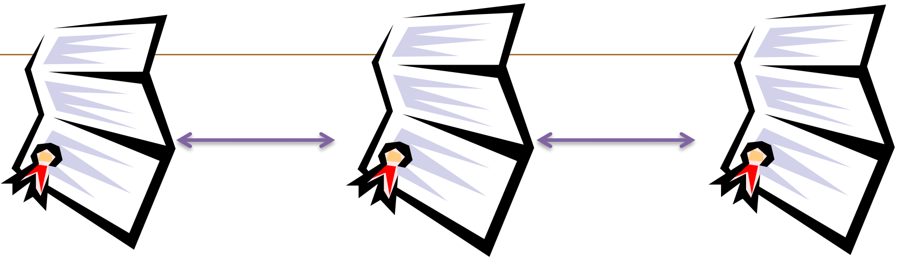
Signature ← Hash value of all other fields signed with the CA's private key. Also includes the signature algorithm identifier

Certificate Chains

Verify Alice’s public key!

1. Receive Alice’s certificate containing her name and her public key

2. We see that it is signed by Bob so we obtain his certificate and verify the signature



Subject: CA
Issuer: CA

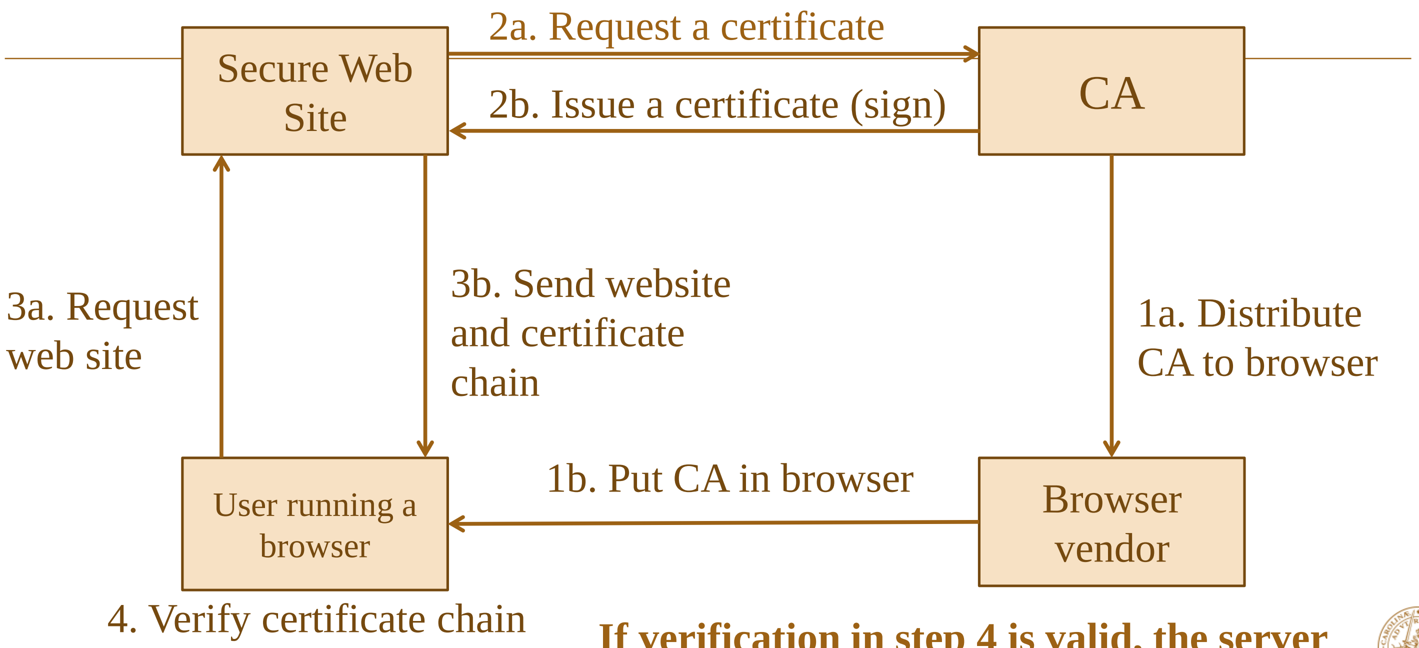
Subject: Bob
Issuer: CA

Subject: Alice
Issuer: Bob

3. Bob’s certificate is signed with CA’s private key so we obtain this certificate and verify the signature

4. The CA certificate is self-signed but if this certificate is among the ones we trust, we decide that the public key of the CA is genuine. We trust Alice’s certificate.

Certificates in TLS



Secure Web Site

2a. Request a certificate

2b. Issue a certificate (sign)

CA

3a. Request web site

3b. Send website and certificate chain

1a. Distribute CA to browser

User running a browser

1b. Put CA in browser

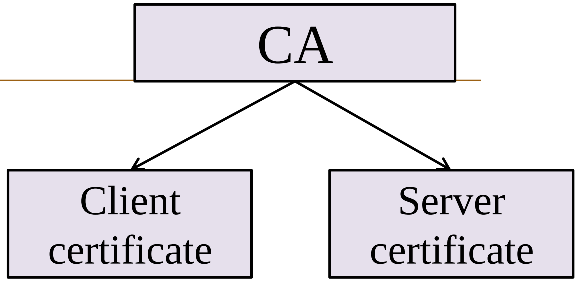
Browser vendor

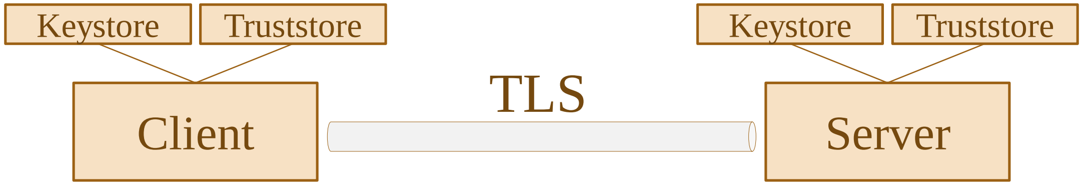
4. Verify certificate chain

If verification in step 4 is valid, the server and client can set up a secure connection

Certificates in Project 1

- Keystore should contain certificate chain
- Truststore should contain the root certificate (CA)
- Connection is established by each party sending its own certificate chain
  - Chain is verified by receiver
    → Public key is trusted
  - Don’t worry about how connection is actually established, we will get there





LUND
UNIVERSITY

©Martin Hell