Security and Privacy

Crypto basics

5.03.2019

slide credits: K. Nikitin, D. Kostic, B. Ford, Ph. Oechslin

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Outline

- Crypto
  - Symmetric-key cryptography
  - Data integrity
  - Authenticated Encryption
  - Public Key Cryptography
  - Public Key Inrastructure (PKI)
- TLS and HTTPS
  - TLS
  - Depolying HTTPS in the Internet
  - Attacks and defenses for HTTPS
- Conclusions
  - Summary and questions

Introduction

Goals of cryptography

cryptography: ‘secret writing’

- Confidentiality
  - data is only accessible with the correct key
- Integrity
  - any modification of data can be detected
- Authentication
  - the author of a message can be identified
- Non Repudiation
  - the author of a message cannot deny it (signatures)

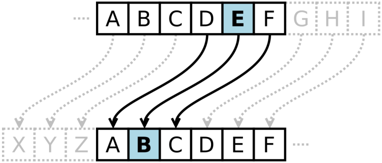
Brief history

- Some data points:
  - 50BC Ceaser's Cipher (substitution)
  - 1883 Kerchoff's principle
  - 70s Data Encryption Standard (DES)
  - 70s Diffie-Hellman public key crypto
  - 70s Rivest, Shamir, Adleman RSA
  - 1985 Elliptic curve crypto (ECC)
  - 90s MD5, SHA-1 hashes
  - 2001 SHA-2
  - 2001 Avanced Encryption Standard AES
  - 2015 SHA-3

- If quantum computer ever come to exist, they will break many crypto algorithms (mainly asymmetric)
  - new resaarch: post-quantum-crypto
    - more complicated but unbreakable by quantum computers

Naive Aproach

- Two parties agree on an algorithm (e.g. rot13) and keep it secret
  - Cesar's cipher was a simple shift of letters



source: **wikipedia**

- If you know that it is a shift, it only takes 26 trials to break it

- 1883: Kerckhoffs' principle: A cryptosystem should be secure even if everything about the system, except the key, is public knowledge

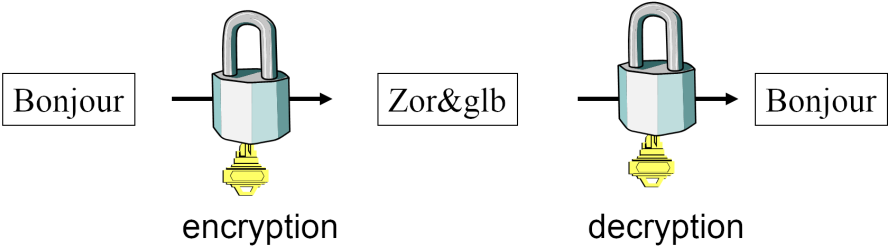
One-Time Pad

- Key is a random string at least as long as the plaintext
  - we use the xor operation to encrypt and decrypt
  - $a \oplus 1 = \bar{a}, \;\; a \oplus 0 = a$
- Encryption: $c = Enc_{key}(m) = m \oplus key$
- Decryption: $m = Dec_{key}(m) = c \oplus key$
- One-time pad is perfect in theory
- Practical disadvantages
  - requires truly uniform random one-time pad values
  - key must not be used more than once
  - key length depends on the message length
    - needs a 1TB key to encrypt a 1TB disk

Symmetric Crypto

Symmetric Crypto

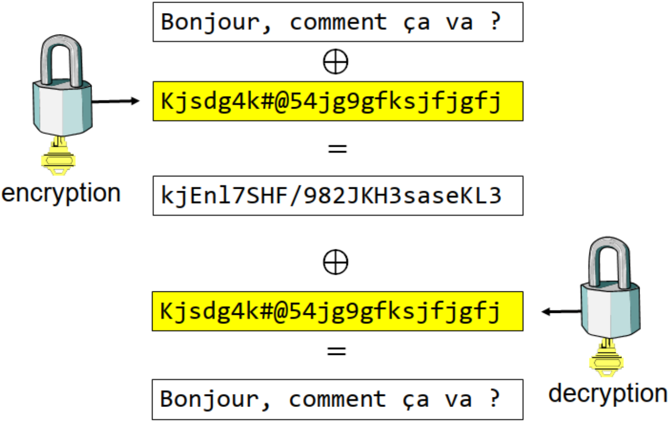
- Encryption and decryption is done with the same key, hence *symmetric*



- Solves the problem of transferring large amounts of confidential data
- Creates the problem of transferring a symmetric key (key exchange)

Stream Cipher

- Use the key and a Pseudo Random Generator to generate a stream of random bits
  - can encrypt data of arbitrary length



Bonjour, comment ça va ?

⊕

Kjsdg4k#@54jg9gfksjfjgfj

=

kjEnl7SHF/982JKH3saseKL3

encryption

⊕

Kjsdg4k#@54jg9gfksjfjgfj

=

Bonjour, comment ça va ?

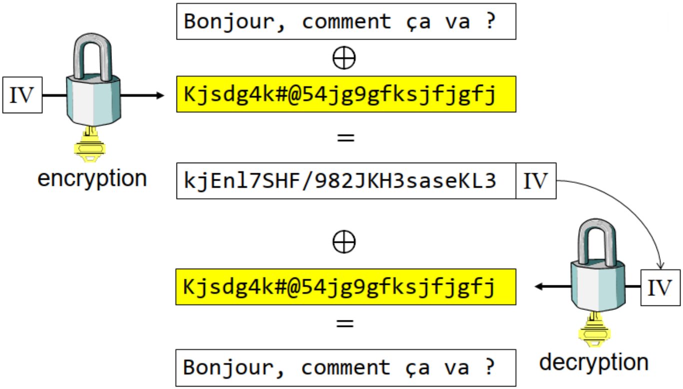
decryption

Stream Ciphers limitations

- Some possible weaknesses in stream ciphers:
- Malleability: Flipping one bit of the cipher text will flip the same bit in the cleartext.
  - e.g. if you know which bit encodes the sign of a value you can change a payment of \$100 to -\$100
  - ➡ you should always add an integrity check when encrypting data
- If two cleartexts are encrypted with the same cipherstream, then: $M_1 \oplus M_2 = C_1 \oplus C_2$
  - if you know some bits of one message, you can deduce the corresponding bits of the other message: $M_2[k] = M_1[k] \oplus C_1[k] \oplus C_2[k]$
  - ➡ you should always add unique initialization vector to create a random seed (a number used only once, a nonce)

Stream Cipher

- Stream cipher with an initialization vector (IV)
- The IV has to be sent with each message

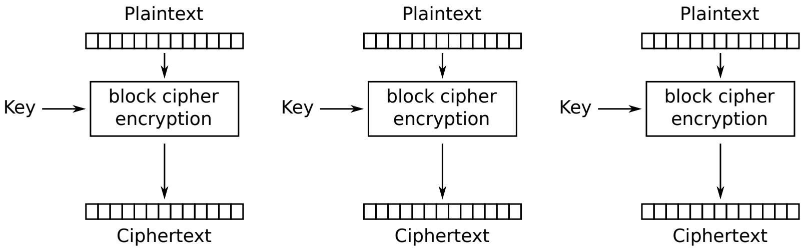


Block cipher

- Encrypts fixed size blocks of data
  - a padding scheme is used to fill the last block
  - a mode of operation is used to combine multiple blocks
- Data Encryption Standard (DES):
  - block size 64 bits (too short, collisions)
  - key size 56 bits (too short, can be brute forced)
  - deprecated (as is tripple DES)
- Advanced Encryption Standard (AES)
  - block size 128 bits
  - key size 128/192/256 bits
  - hardware support in modern processors (e.g. Intel, AMD)

Modes of operation: ECB

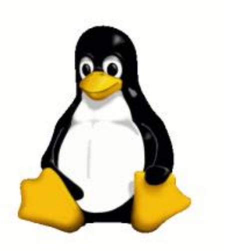
- Encrypt each block separately, with the same key



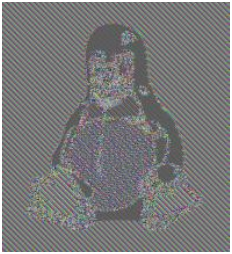
Electronic Codebook (ECB) mode encryption

Modes of operation: ECB

- The problem ? You can see the penguin!



original



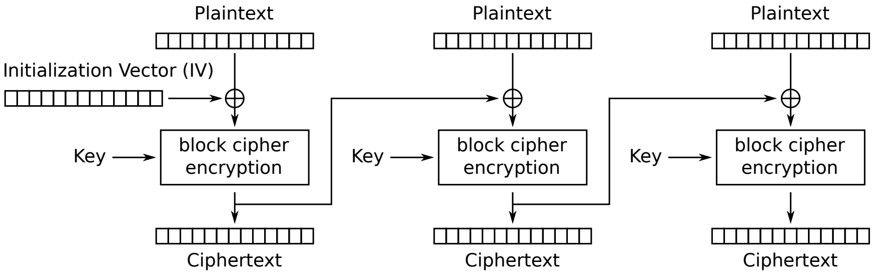
encrypted with ECB

source: Larry Ewing, The GIMP

- Same clear text block results in same ciphertext block

Mode of operation: CBC

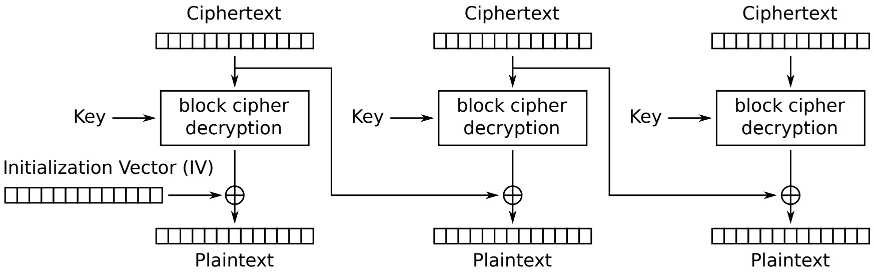
- Introduces the use of an initialisation vector



Cipher Block Chaining (CBC) mode encryption

Mode of operation: CBC

- Decryption is the opposite of encryption:



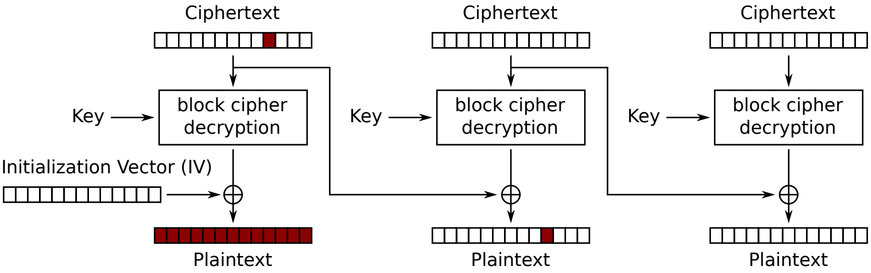
Cipher Block Chaining (CBC) mode decryption

Quiz!

- What happens if a bit is flipped due to a transmission error ?
  - does the error propagate to all following blocks ?
  - does it only affect the current block ?
  - none of the above ?

CBC: Flipping one bit

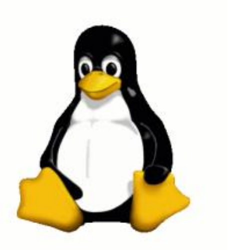
- A whole block is garbled
- One bit of the next block is inverted



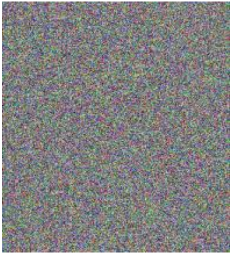
Cipher Block Chaining (CBC) mode decryption

Mode of operation: CBC

- CBC encryption does not reveal any structure



original



encrypted with CBC

source: Larry Ewing, The GIMP

CBC limitations

- Some possible weaknesses of CBC
- Malleability:
  - Flipping one bit of the IV will flip the same bit in the cleartext
  - flipping on bit in a ciphertext block flips the same bit in the next cleartext block and mangles the current block
  - ➡ always add an integrity check when encrypting data
- Padding oracle attacks:
  - The last block must be padded to obtain the correct block size
  - If not carefully implemented, validation of padding can lead to leakage of the cleartext (e.g. in TLS)
    - Discovered by Serge Vaudenay

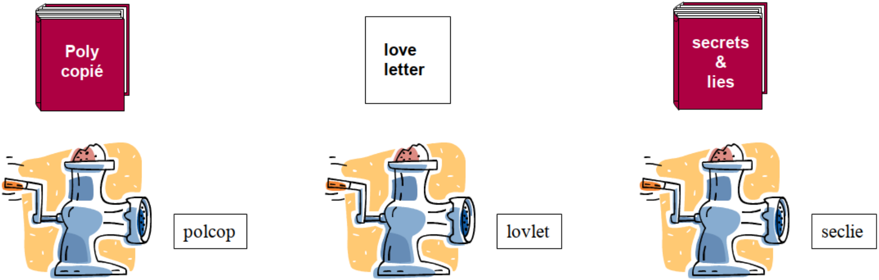
Data integrity

Data Integrity

- Integrity: ensure that data is not modified by unauthorized parties
- Integrity $\neq$ confidentiality
- Remember malleability of stream and block ciphers: easy to flip bits of the cleartext
- We need a primitive that
  - detects any modification of the message
  - can not be fooled by an attacker

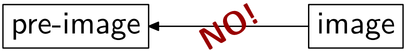
Cryptographic hash functions

- Hash functions take an arbitrary length input and generate a fixed length output
- Cryptographers says they create an ‘image’ from a ‘preimage’



Properties of crypto hash functions

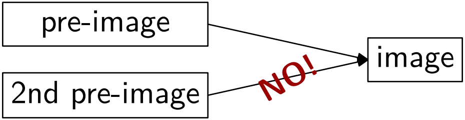
- Pre-image resistance:
  - Given an hash h, it is difficult to find a message m for which $h = \mathrm{hash}(m)$
  - This implies that the function is a *one-way* function
  - Very useful for passowrd hashing



  - Possible attack: Given a password hash, an attacker should not be able to calculate the password

Properties of crypto hash functions

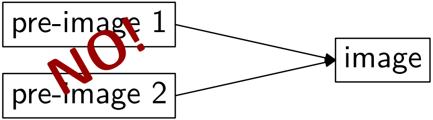
- Second pre-image resistance:
  - Given a message m_1, it is difficult to find a second message m_2 such that $\mathrm{hash}(m_1) = \mathrm{hash}(m_2)$



  - Possible attack: Hashes protect integrity of messages: an attacker should not be able to to forge a message with the same hash as an existing message.

Properties of crypto hash functions

- Collision resistance:
  - It is difficult to find two arbitrary messages that have the same hash.
  - Such a pair of messages is called a collision



  - Possible attack: hashes are used in signatures. An attacker could show one pre-image and have it signed by a third party. The signature would also be valid for the other pre-image.

Quiz!

- What is the complexity of randomly finding a 2nd pre-image in a hash that generates 160 bit hashes (e.g. SHA-1) ?
- What is the complexity of randomly finding a collision SHA-1 ?
- note: if the hash function has some weaknesses the complexity might be lower.

2nd pre-image or collisinon ?

James Brown

Barry White

```
com402/slides/figures$ md5sum james_brown.jpg
e06723d4961a0a3f950e7786f3766338  james_brown.jpg
com402/slides/figures$ md5sum barry_white.jpg
e06723d4961a0a3f950e7786f3766338  barry_white.jpg
```

2nd pre-image or collision ?

- It is a chosen-prefix collision
- The MD5 hash function is broken
- It is easy to find collisions: $\mathrm{md5}(something) = \mathrm{md5}(something\_else)$
- It is even possible to find collision with given prefixes:

$$\mathrm{md5}(prefix_1 \| something) = \mathrm{md5}(prefix_2 \| something\_else)$$

- In the jpeg file format, you can add any garbage after the End Of Image marker.
- ➡ You can choose the orginal `James_Brown.jpg` as $prefix_1$ and `Barry_White.jpg` as $prefix_2$ and calculate the chosen prefix collision.
- See the details on Nat McHuhg's **Blog**

Cryptographic hash functions:

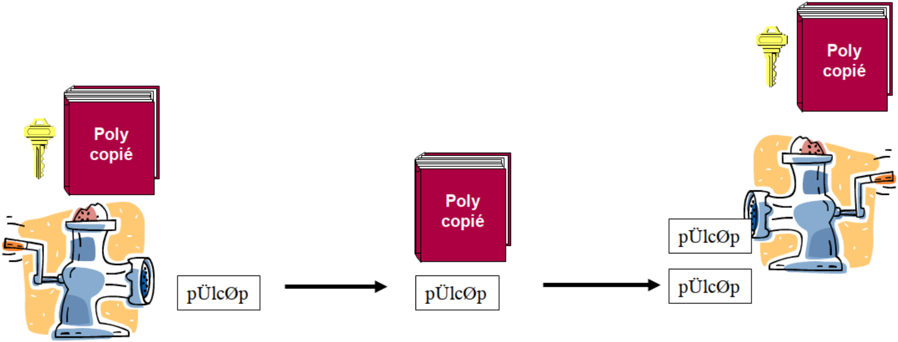
- MD5: 128 bits, broken, chosen prefix collisions are easy
- SHA-1: 160 bits, broken, collisions are difficult but possible
- SHA-2: 224/256/384/512 bits, not broken, but related to SHA-1
- **SHA-3**: 224/256/384/512 bits, no weaknesses known

Message authentication codes (MACs)

- Like a hash function, but involves a symmetric key
- The same key is used to generate the MAC and to validate it
- If the key is known only to the two parties of an exchange, a correct MAC proves:
  - that the message was not created by a third party (authentication)
  - that the message has not been modified (integrity)

Message authentication codes

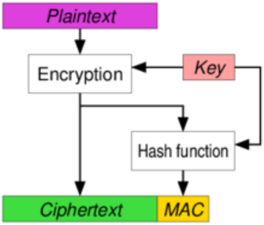
- Often based on a hash functions (e.g.HMAC SHA-2) or on block ciphers (e.g CBC-MAC-AES)



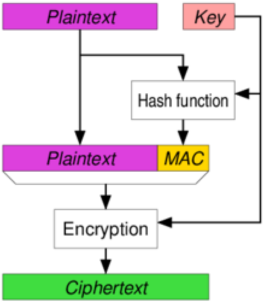
Authenticated Encryption

Encryption and Integrity

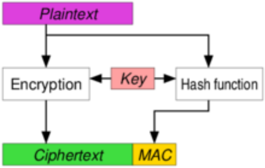
- Usually we want encryption and integrity (e.g AES-CBC & HMAC-SHA2)
- Different approaches:
  - encrypt then mac (e.g. in IPSec)
  - mac then encrypt (e.g. in TLS)
  - encrypt and mac (e.g. in SSH)



enc. then mac



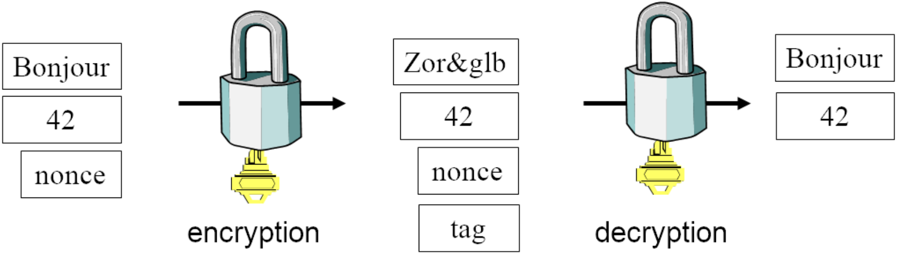
mac then enc.



enc. and mac

Authenticated Encryption

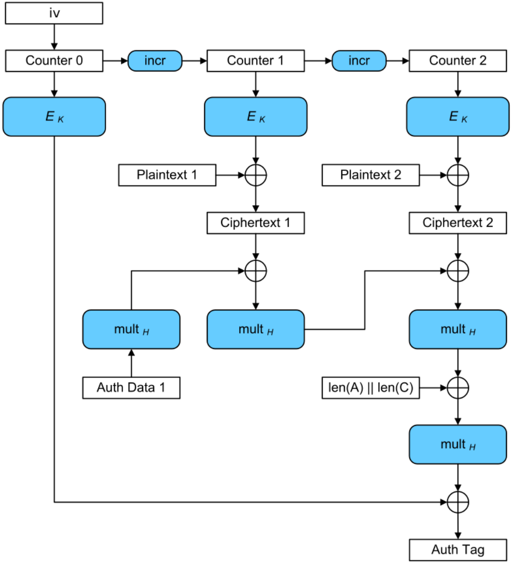
- Modern encrpytion modes combine encryption and authentication
- They include additionnal data that is authenticated but not encrypted
  - used for sequence numbers or other meta data
- AEAD (Authenticated Encryption with Authenticated Data)



- ‘Bonjour’:Message, ‘42’:authenticated data, nonce: for initialization (IV), tag: for integrity

AEAD: Gallois/Counter Mode GCM

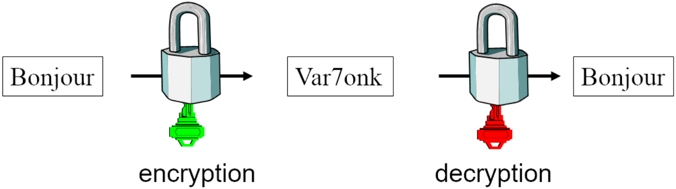
- used with any block cipher, typically AES: AES-GCM
- This is the current best-practice

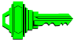
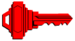


Public-key Cryptography

Public-key Cryptography

- Solves the problem of having to agree on a pre-shared symmetric key
- Uses a pair of public and private key instead



- Encryption with the public key  guarantees that only the private key  can decrypt
- The public key can be transferred freely, no need to keep it secret.
- Much better than symmetric keys!

Origin of public-key crypto

- 1975 Diffie and Hellman describe the idea of asymmetric crypto in 'New directions in Cryptography'

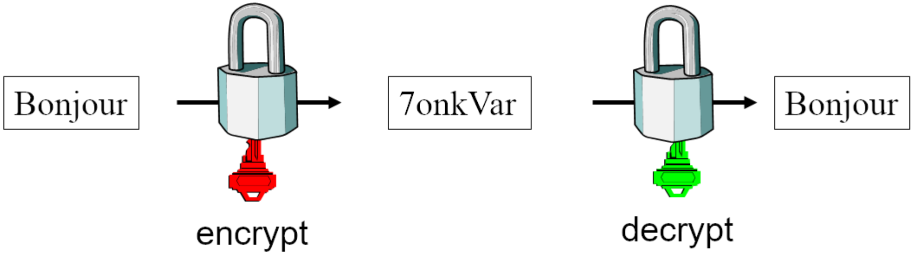
> "We stand today on the brink of a revolution in cryptography. The development of cheap digital hardware has freed it from the design limitations of mechanical computing and brought the cost of high grade cryptographic devices down to where they can be used in such commercial applications as remote cash dispensers and computer terminals.
>
> In turn, such applications create a need for new types of cryptographic systems which minimize the necessity of secure key distribution channels and supply the equivalent of a written signature. At the same time, theoretical developments in information theory and computer science show promise of providing provably secure cryptosystems, changing this ancient art into a science."

Original Primitives

- Public and private key
  - Two keys, public is widely distributed, private is kept secret
  - Must be hard to derive private from public
  - May be easy to derive public from private
- Encryption and decryption
  - Encrypt with public key, decrypt with private key
  - Hard to decrypt without private key
- Digital signature
  - Sign with private key, verify with public key
  - Hard to create signature without private key
- Interactive key exchange
  - Create a shared private key over an insecure communication channel

Digital signature

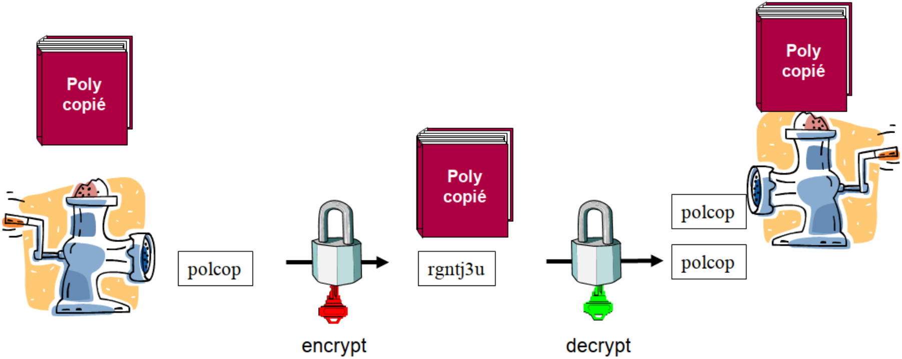
- For signing we just encrypt with the private key
- Anybody can verify the signature by decrypting with the public key



- If the public key correctly decrypts the signature, this is proof that it has been created with the private key of the signer.

Digital signature

- Signing long messages with asymmetric crypto is expensive
- We can use a hash function to make it more efficient:



Digital signatures: usage examples

- Software-update distribution
  - Your computer knows the public key of the software developer
    - type `apt-key list` on linux to see them
  - Updates are signed
  - Your computer can verify that the update indeed comes from the developer and that it has not been altered on its way
- Authenticated e-mail
  - If Alice knows Bob's public key, she can verify Bob's signature on the messages.
- Contracts !
  - In Switzerland you can sign contracts with digital signatures
    - provided you get your public key certificate from a certified provider

Diffie Hellman key exchange

Alice		**Bob**
choose a, $A = g^a \bmod n$ send A receive B $K = B^a \bmod n$ $= \boldsymbol{g^{ab} \bmod n}$	agree on g and n $\rightarrow A$ $B \leftarrow$	choose b $B = g^b \bmod n$ send B receive A $K = A^b \bmod n$ $= \boldsymbol{g^{ab} \bmod n}$

- By exchanging the public values A, B, and combining them with their private values b, a, Alice & Bob can create a shared secret K

- A attacker who observes A and B cannot find K

Public-key Algorithms: RSA

by Rivest, Shamir & Adleman

- n is product of two large primes $p * q$,
- e is coprime with $(p-1)(q-1)$, $(ed-1)$ is multiple of $(p-1)(q-1)$

Alice		**Bob**
$K_{pub} = (e, n)$, $K_{priv} = (d, n)$	$\rightarrow K_{pub}$	chose message M
$M = C^d \bmod n$	$C \leftarrow$	$C = M^e \bmod n$

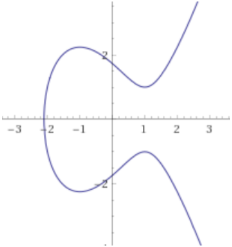
- RSA can be broken by factoring n. Use a number of 2048 bits or more

Public-key Algorithms: ECC

Elliptic curve cryptography

- Based on the algebraic structure of eliptic curves over finite fields.

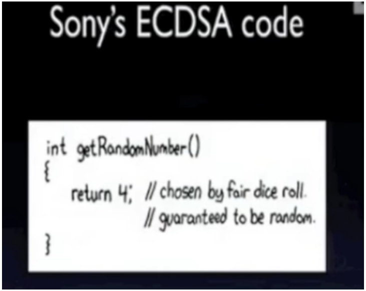
- Smaller keys for equivalent security (e.g. 256 bit ECC comparable to 2048 bit RSA)
  - faster operations
  - less data



- Popular curves offer fast performance and additional security guarantees
  - Curve25519
  - Curve448

Public-key Algorithms: ECC

- ECC is used for key exchange, e.g. in TLS: ECC Diffie-Hellman, ECDHE
- Used for signature: ECC Digital Signature Alg. ECDSA
- For example, Sony used ECDSA to sign the software of the Playstation 3
  - ECDSA uses the private key and a random number to generate the signature
  - The private key can be recovered if the same random number is used in two signatures



Sony's ECDSA code

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

source: fail0ver crew at 27c3

- Bitcoin addresses are a hash of an ECDSA public key
- iMessages and iCloud keychain sync rely on ECDSA

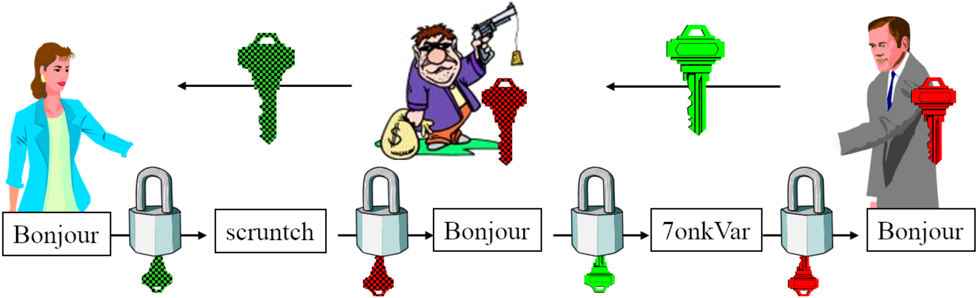
Crypto comparison

- Asymmetric is powerful but orders of magnitude slower than symmetric crypto
- Asymmetric is typically used to exchange a symmetric key, then symmetric takes over
- All these algorithms are only safe if you use keys that are long enough:
  - symmetric 128 to 256 bits
  - asymmetric: RSA 2049 bits, ECC 256 bits
  - hash functions: 256 bits

Public Key Infrastructure (PKI)

Key Distribution

- With Public-key crypto, the public key doesn't have to be secret
- It still has to be authentic, however
- If a man in the middle can replace a public key he can decrypt the traffic

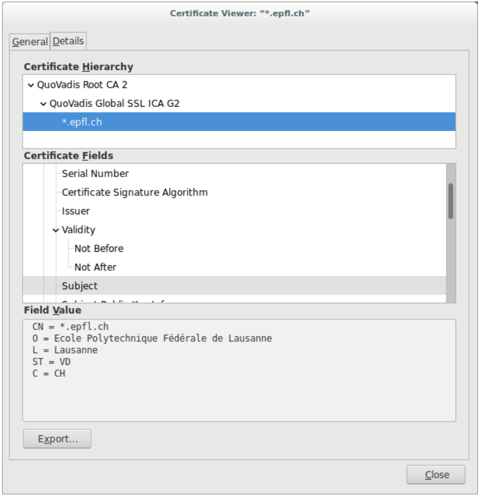


Key Distribution

- We need a trusted third party to distribute the public keys
- The Certification Authority (CA) certifies the keys by signing them
- It does some validation before signing the keys
  - may ask for an e-mail
  - may ask for a copy of your passport
  - this is documented in the Certificate Practice Policy (CSP) of the CA
- If we trust the key of the CA, we can trust all keys signed by the CA
- A ‘signed key’ is a certificate. It contains at least
  - the identity of the holder (subject)
  - the validity date of the certificate
  - the public key of the subject
  - the signature by the CA

Example certificate

- The certificate of of the EPFL website is signed by QuoVadis
- Quovadis Global SSL ICA G2 is signed by QuoVadis Root CA



Certificate Viewer: "*.epfl.ch"

General | Details

Certificate Hierarchy

```
⌄ QuoVadis Root CA 2
  ⌄ QuoVadis Global SSL ICA G2
      *.epfl.ch
```

Certificate Fields

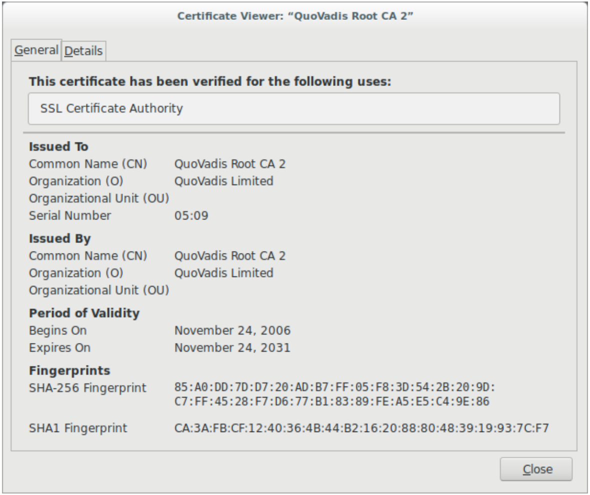
```
    Serial Number
    Certificate Signature Algorithm
    Issuer
  ⌄ Validity
      Not Before
      Not After
    Subject
```

Field Value

```
CN = *.epfl.ch
O = Ecole Polytechnique Fédérale de Lausanne
L = Lausanne
ST = VD
C = CH
```

Example CA root certificate

- The root certificate of a CA is signed by itself



Certificate Viewer: “QuoVadis Root CA 2”

This certificate has been verified for the following uses:

SSL Certificate Authority

Issued To

Common Name (CN)	QuoVadis Root CA 2
Organization (O)	QuoVadis Limited
Organizational Unit (OU)	
Serial Number	05:09

Issued By

Common Name (CN)	QuoVadis Root CA 2
Organization (O)	QuoVadis Limited
Organizational Unit (OU)	

Period of Validity

Begins On	November 24, 2006
Expires On	November 24, 2031

Fingerprints

SHA-256 Fingerprint	85:A0:DD:7D:D7:20:AD:B7:FF:05:F8:3D:54:2B:20:9D: C7:FF:45:28:F7:D6:77:B1:83:89:FE:A5:E5:C4:9E:86
SHA1 Fingerprint	CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7

Remember James Brown ?

- In 2005 Arjen Lenstra *et al* found a way to make colliding certificates, when the MD5 hash function is used for the signature

- You can create a web site certificate and an intermediate CA certificate that have the same hash
  - ➡ thus they have the same signature

- You present the web server certificate for signature to a CA and then append the signature to the other certificate:
  - you can now generate as many certificates as you want.

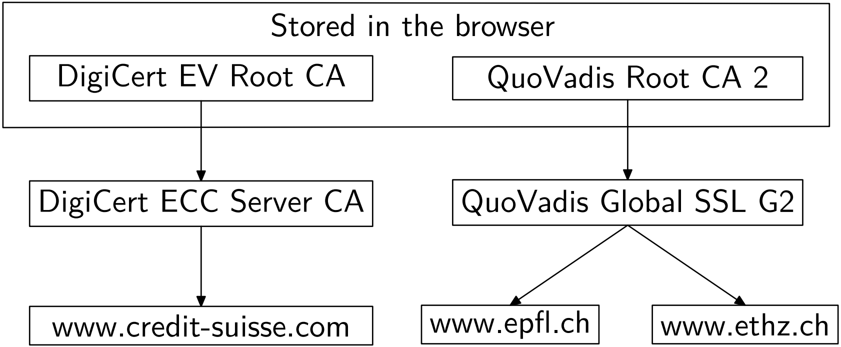
- In december 2008, with the help a few other researchers, they were able to carry out the attack against the RapidSSL CA.
  - see **http://www.win.tue.nl/hashclash/rogue-ca/**

- Since then MD5 is deprecated in x509 certificates

Hierarchy of Trust

- Current browsers know a set of root CAs
  - 151 root CAs in Firefox as of Feb 2019

- If there is a chain of signatures up to a trusted root, the browser trusts the certificate



Conclusions & Questions

Conclusions

- Crypto can be symmetric (fast), asymmetric (public keys) and provides confidentiality, integrity, authentication and non-repudiation

- Asymmetric is only used to encrypt short data. e.g.
  - The hash of a document, when signing it
  - A random symmetric key, for encryption if a document

- ECC is replacing RSA as asymmetric algorithm, as it is faster and needs shorter keys

- Trust is not possible without a trust root

- Web browsers trust 150-ish Certification Authorities from almost as many countries.
  - this is not perfect

Questions

- A buyer and a seller sign a contract using a symmetric key and a HMAC.
  - why doesn't this work ?
- If asymmetric crypto is so very powerful, why do we still use symmetric ciphers ?
- Using the same IV twice in a stream cipher is more dangerous that using it twice in a block cipher. Why ?
- Describe an attack that is possible if a hash function is not resistant to collisions.
- How comes AEAD in not malleable ?