Basics of ICT security

Antonio Lioy
< lioy @ polito.it >

Politecnico di Torino
Dip. Automatica e Informatica

Cryptography

sender
key-1
message (in clear)
encryption
message (encrypted)
receiver
key-2
message (in clear)
decryption

Terminology

- message in clear:
  - plaintext or cleartext
  - we will refer to it as P
- encrypted message:
  - ciphertext
  - we will refer to it as C
  - note that in some countries "encrypted" sounds offensive for religious reasons (cult of dead); in those cases "enciphered" is preferred
Cryptography’s strength (Kerchoffs’ principle)

- if the keys:
  - are kept secret
  - are managed only by trusted systems
  - are of adequate length
- then ...
- ... it has no importance that the encryption and decryption algorithms are kept secret
- ... on the contrary it is better to make the algorithms public so that they can be widely analysed and their possible weaknesses identified


Security through obscurity (STO)

Security trough obscurity is a thing as bad with computer systems as it is with women

Secret key cryptography

- key-1 = key-2
- symmetric algorithms
- low computational load
- used for data encryption
- main algorithms:
  - DES, triple-DES
  - IDEA
  - RC2, RC5
  - RC4
  - AES
Symmetric cryptography

- single key
- key shared between sender and receiver (only!)
- \( C = \text{enc (} K , P \text{)} \) or \( C = \{ P \} K \)
- \( P = \text{dec (} K , C \text{)} = \text{enc}^{-1} ( K , C ) \)
- problem: how to share (securely) the secret key among sender and receiver?

![Symmetric cryptography diagram]

Symmetric algorithms

<table>
<thead>
<tr>
<th>name</th>
<th>key</th>
<th>block</th>
<th>note</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES</td>
<td>56 bit</td>
<td>64 bit</td>
<td>obsolete</td>
</tr>
<tr>
<td>3-DES</td>
<td>112 bit</td>
<td>64 bit</td>
<td>56-112 bit strength</td>
</tr>
<tr>
<td>3-DES</td>
<td>168 bit</td>
<td>64 bit</td>
<td>112 bit strength</td>
</tr>
<tr>
<td>IDEA</td>
<td>128 bit</td>
<td>64 bit</td>
<td></td>
</tr>
<tr>
<td>RC2</td>
<td>8-1024 bit</td>
<td>64 bit</td>
<td>usually K=64 bit</td>
</tr>
<tr>
<td>RC4</td>
<td>variable</td>
<td>stream</td>
<td>secret</td>
</tr>
<tr>
<td>RC5</td>
<td>0-2048 bit</td>
<td>1-256 bit</td>
<td>optimal when B=2W</td>
</tr>
<tr>
<td>AES</td>
<td>128-256 bit</td>
<td>128 bit</td>
<td>alias Rjindael</td>
</tr>
</tbody>
</table>

The EX-OR (XOR) function

- ideal "confusion" operator
- if the input is random (probability 0 : 1 = 50 : 50%) then also the output will be equally random
- primitive operation available on all CPU
- truth table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
DES

- Data Encryption Standard
- standard FIPS 46/2
- mode of application standard FIPS 81
- 56 bits key (+ 8 parity bits) = 64 bits
- 64 bits data block
- designed to be efficient in hardware because it requires:
  - XOR
  - shift
  - permutation (!)

Triple DES (3DES, TDES)

- repeated application of DES
- uses two of three 56 bits keys
- usually applied in the EDE mode
  (for compatibility with DES when K1 = K2 = K3)
- 3DES with 2 keys (Keq=56 bit if 2^{56}B of memory is available, otherwise Keq=112 bit)
  \[ C' = \text{enc}(K_1, P) \quad C'' = \text{dec}(K_2, C') \quad C = \text{enc}(K_3, C'') \]
- 3DES with 3 keys (Keq=112 bit)
  \[ C' = \text{enc}(K_1, P) \quad C'' = \text{dec}(K_2, C') \quad C = \text{enc}(K_3, C'') \]
- standard FIPS 46/3 and ANSI X9.52

Double DES?

- double application of encryption algorithms is subject to a known-plaintext attack named meet-in-the-middle which allows to decrypt data with at most \(2^{N+1}\) attempts (if the keys are N-bit long)
- thus usually the double version of encryption algorithms is never used
- the computation time doubles but the effective key length increases of just one bit!
- note: if the base symmetric algorithm would be a group then it would exist \(K_3\) so that
  \[ \text{enc} (K_2, \text{enc} (K_1, P)) = \text{enc} (K_3, P) \]
Meet-in-the-middle attack

- **hypothesis:**
  - N bit keys
  - known P and C such that C = enc (K₂, enc (K₁, P))

- **note:**
  - ∃ M such that M = enc(K₁, P) and C = enc(K₂, M)

- **actions:**
  - compute 2ᴺ values Xᵢ = enc (Kᵢ, P)
  - compute 2ᴺ values Yⱼ = dec (Kⱼ, C)
  - search those values Kᵢ and Kⱼ such that Xᵢ = Yⱼ
  - “false positives” can be easily discarded if more than one (P,C) couple is available

IDEA

- International Data Encryption Algorithm
- patented but with low royalty (only for commercial use, ASCOM AG)
- 128 bits key
- 64 bits data block
- famous because used in PGP
- operations used:
  - XOR
  - addition modulo 16
  - multiplication modulo 2¹⁶+1

An application of IDEA

courtesy of Ascom AG
RC2, RC4

- developed by Ron Rivest
- RC = Ron’s Code
- algorithms proprietary of RSA but not patented
- 3 or 10 times faster than DES
- RC2 is a block algorithm, RC4 is a stream one
- variable length key
  - RC2:
    - published as RFC-2268 (mar 1998)
    - 8 to 1024 bits keys (usually 64 bits)
    - 64 bits data block
  - RC4 reverse engineered (ARCFOUR)

RC5

- RFC-2040
- B bits data block (0 < B < 257)
- b bytes fix/variable key (0 ≤ b < 256) that is between 0 and 2048 bits
- works best when B = 2 W
- operations used:
  - shift
  - rotate (!)
  - modular addition
- used in WAP

Application of block algorithms

How a block algorithm is applied to a data quantity different from the algorithm’s block size?

- to encrypt data of size > algorithm’s block size:
  - ECB (Electronic Code Book)
  - CBC (Cipher Block Chaining)
- to encrypt data of size < algorithm’s block size:
  - padding
  - CFB (Cipher FeedBack), OFB (Output FeedBack)
  - CTR (Counter mode)
ECB (Electronic Code Book)

- formula for the i-th block:
  \[ C_i = \text{enc} (K, P_i) \]
- NOT to be used on long messages because
  - swapping of two blocks goes undetected
  - identical blocks generate identical ciphertexts
  hence it is vulnerable to known-plaintext attacks

\[ P_1 \quad P_2 \quad P_3 \quad P_4 \]
\[ K \quad K \quad K \quad K \]
\[ C_1 \quad C_2 \quad C_3 \quad C_4 \]

ECB - decrypt

- formula for the i-th block:
  \[ P_i = \text{enc}^{-1} (K, C_i) \]
- an error in transmission generates an error at the decryption of one block

\[ P_1 \quad P_2 \quad P_3 \quad P_4 \]
\[ K \quad K \quad K \quad K \]
\[ C_1 \quad C_2 \quad C_3 \quad C_4 \]

CBC (Cipher Block Chaining)

- formula for the i-th block:
  \[ C_i = \text{enc} (K, P_i \oplus C_{i-1}) \]
- requires \( C_0 = \text{IV} \) (Initialization Vector)
CBC - decryption

- formula for the i-th block:
  \[ P_i = \text{enc}^{-1}(K, C_i) \oplus C_{i-1} \]
- requires \( C_0 \) (i.e. IV) to be known by the receiver
- an error in transmission generates an error at the decryption of two blocks

Padding (aligning, filling)

- size of algorithm's block \( B \)
- size of data to process \( D \) (not a multiple of \( B \))
- add bits until a multiple of \( B \) is reached

Padding techniques

- (if length is known or it can be obtained – e.g. a C string) add null bytes
  - \( \ldots 0x00 \ 0x00 \ 0x00 \)
- (original DES) one 1 bit followed by many 0
  - \( \ldots 1000000 \)
- one byte with value 128 followed by null bytes
  - \( \ldots 0x80 \ 0x00 \ 0x00 \)
- last byte's value equal to the length of padding
  - \( \ldots 0x?? \ 0x?? \ 0x03 \)
- what about the value of the other bytes?
Padding with explicit length (L)

- (Schneier) null bytes:
  - e.g. … 0x00 0x00 0x03
- (SSL/TLS) bytes with value L:
  - e.g. … 0x03 0x03 0x03
- (SSH2) random bytes:
  - e.g. … 0x05 0xF2 0x03
- (IPsec/ESP) progressive number:
  - e.g. … 0x01 0x02 0x03
- byte with value L-1:
  - e.g. … 0x02 0x02 0x02
- some offer (minimal) integrity control

Padding – some notes

- typically applied to large data, on the last fragment resulting from the division in blocks (e.g. for ECB or CBC)
- for |D| < |B| we prefer ad-hoc techniques (CFB, OFB, CTR, …)
- even if the plaintext is an exact multiple of the block, padding must be added anyhow to avoid errors in the interpretation of the last block
  \[ 1 \leq L \leq |B| \]
- the SSH2 padding implies that equal data are encrypted to different ciphertexts
- the padding type for a certain algorithm determines the type of (some) possible attacks

Ciphertext stealing (CTS)

- CTS permits to use block algorithms without padding
  - last (partial) block filled with bytes from the second-to-last block
  - these bytes are removed from the second-to-last block (which becomes a partial one)
  - after encryption, exchange the position of the last and second-to-last blocks
  - useful when we cannot increase the size of the data after encryption
  - the computation time slightly increases
**CTS – example with ECB (encryption)**

\[ P_{n-1} \oplus K \rightarrow C_{n-1} \]

\[ P_n \rightarrow C_n \]

**CFB (Cipher Feedback)**

- allows to encrypt N bits at a time (a group)
- requires an IV (to initialize the shift register)
- a transmission error causes an error in the decryption of a block plus all the subsequent data

**OFB (Output Feedback)**

- allows to encrypt N bits at a time (a group)
- requires an IV (to initialize the shift register)
- a transmission error causes an error only in one group
CTR (Counter mode)

- allows to encrypt N bits at a time (a group)
- random direct access to any ciphertext group
- requires a nonce and a counter (concatenated, summed, XOR, …)
- a transmission error ~ causes error only in one group

nonce & counter (1 byte)
\[ K \]
\[
\text{leftmost byte}
\]

Symmetric stream algorithms

- operate on a stream of data without requiring the division on blocks, typically on one bit or one byte at a time
- ideal algorithm:
  - one-time pad (requires a key which is as long as the message to protect!)
- real algorithms:
  - use pseudo-random key generators, synchronized between the sender and the receiver
  - examples: RC4 and SEAL

Algorithms of type stream
Competition for stream ciphers

- **ESTREAM**, http://www.ecrypt.eu.org/stream/
- completed on April 2008 but revised periodically (last revision on January 2012)
- selected an algorithm portfolio:
  - (software, 128-bit key)
    - HC-128, Rabbit, Salsa20/12, SOSEMANUK
  - (hardware, 80-bit key)
    - Grain v1, MICKEY 2.0, Trivium
- **extended versions:**
  - (sw, 256-bit) HC-256, Salsa20/12
  - (hw, 128-bit) MICKEY-128 2.0

Symmetric cryptography

- single and secret key
- one key for each couple / group of users

Key distribution for symmetric cryptography

- for a complete private communication between N parties \( N \times (N-1) / 2 \) keys are necessary:
  - distribution OOB (Out-Of-Band)
  - distribution by means of key exchange algorithms
Length of secret keys

- If:
  - the encryption algorithm was well designed
  - the keys – Nbit in length – are kept secret
- ... then the only possible attack is the brute force (exhaustive) attack which requires a number of trials equal to
  \[ T = 2^{N_{\text{bit}}} \]

Length of cryptographic keys

<table>
<thead>
<tr>
<th>Symmetric</th>
<th>40</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asymmetric</td>
<td>256</td>
<td>512</td>
<td>1024</td>
<td>2048</td>
<td>...</td>
</tr>
</tbody>
</table>

Low security

High security

DES challenges

- \(2^{56} = 72,057,594\) billions of possible keys
- DES challenge I
  - start=18-feb-1997, fine=17-june-1997
  - 17,731,000 billions of tried keys (25%)
  - about 15,000 computer in network
- DES challenge II
  - start=13-jan-98, end=23-feb-98
  - 63,686,000 billion of tried keys (87%)
  - about 20,000 computer in network
The end (?) of DES

- **DES challenge III**
  - start=13-jul-98, end=15-jul-98
  - 17,903,000 billions of tried keys (25%)
  - 1 special-purpose system (DEEP CRACK) developed by the EFF at a cost of 250,000 $
  - it is thus possible to construct a computer system that can decrypt a generic DES message, but:
    - it is necessary to know the type of data (e.g. ASCII)
    - the machine cannot decrypt 3DES messages
    - DES is not intrinsically weak, it uses only a short key!

Faster and faster

- **DES challenge IV**
  - start=18-jan-1999, end=after 22h 15m
  - 16,017,000 billions of tried keys (22%)
  - DEEP CRACK plus a few thousands of workstations
  - peak power: 250 Gkey/s
  - average power: 199 Gkey/s

What after DES?

- IETF changes all RFC advising not to use DES and suggesting the use of triple DES
- RFC-4772 (security implications of using DES)
- a German bank sentenced for a fraud made by means of a system based on DES
- on 15-jan-1999 FIPS withdrew DES (46/2) and replaced it with 3DES (46/3)
- competitive call of the US government for selecting a new symmetrical algorithm:
  - AES (Advanced Encryption Standard)
  - key length at least 256 bits
  - block size at least 128 bits
AES (Advanced Encryption Standard)

- 15 candidates
- 5 finalists (9 August 1999):
  - MARS (IBM)
  - RC6 (RSA, i.e. Ron Rivest)
  - Rijndael (Joan Daemen, Vincent Rijmen)
  - Serpent (Ross Anderson, Eli Biham, Lars Knudsen)
  - Twofish (Bruce Schneier and others)
- Information about the selection process:
  http://www.nist.gov/aes

AES = RIJNDAEL

- 2 October 2000
- RIJNDAEL chosen as winner
- Published in November 2001 as FIPS-197

Gradually being adopted (it takes so long because crypto algorithms are like wine: the best ones are those aged for several years ...)

Public key cryptography

- key-1 ≠ key-2
- Asymmetric algorithms
- Pair of keys (public and private)
- If one of key is used for encryption then the other one must be used for decryption
- Processing load is high
- Used to distribute secret keys and for the electronic signature (with hashing)
- Principal algorithms:
  - Diffie-Hellman, RSA, DSA, El Gamal, ...
Asymmetric cryptography

- keys generated in pairs: 
  - private key (Kpri) + public key (Kpub)
- keys with inverse functionality: data encrypted with one key can be decrypted only by the other key

```
  SENDER: Kpri -> E -> q #%3& -> D -> hello!
  RECEIVER: Kpub -> E -> Y?2gs3 -> D -> hello!
```

Digital signature

- digital signature = asymmetric encryption of data made with the private key of the author
- usually data is not directly encrypted but only its summary (digest)
- provides data authentication (and integrity)

```
  anybody: KXpub -> E -> public message signed by X
  X: KXpri -> D -> private message for X
```

Confidentiality without shared secrets

- it is possible to generate a secret message for a particular receiver given only its public key

```
  anybody: KXpub -> E -> private message for X
  X: KXpri -> D ->
```
### Public key algorithms

- **RSA (Rivest - Shamir - Adleman)**
  - Product of prime numbers, factoring of result
  - Secrecy and digital signature
  - Patented - only in USA - by RSA; patent expired on 20-set-2000
- **DSA (Digital Signature Algorithm)**
  - Taking the power, logarithm of the result
  - Digital signature only
  - For encryption use El-Gamal
  - Standard NIST for DSS (FIPS-186)

### The DSA algorithm

#### Sign

- \( r = f_2(kBM, k, x, y) = g^k (\bmod \ q) \)
- \( s = f_1(r, x, y) = (y^k \cdot x \bmod q) \)

#### Verify

- \( w = f_3(r) = (s^r \cdot y) \mod q \)
- \( v = f_4(w, q, r) \)

### RSA – the algorithm (I)

- Public module \( N = P \times Q \) known to anybody
- \( P \) and \( Q \) are prime, large and secret
- \( PHI = (P-1) \) (Q-1)
- Public exponent \( E \) such that arbitrarily \( 1 < E < PHI \) and it is relatively prime with respect to \( PHI \)
- Private exponent \( D = E^{-1} \bmod PHI \)
- Public key = \( (N, E) \)
- Private key = \( (N, D) \)
- \( P \) and \( Q \) are deleted, discarded, killed, ...
RSA – the algorithm (II)
- RSA may cipher/decipher only data whose value is less than the value of the module N (it’s a sort of block algorithm)
- plaintext:  \( p < N \)
- encrypt:  \( c = p^E \mod N \)
- decrypt:  \( p = c^D \mod N \)
- the roles of E and D are interchangeable because  \((x^E)^D \mod N = (x^D)^E \mod N\)
- note that the complexity of the operations depends upon the number of bits with value 1 in the exponents E and D

Modular arithmetic
- \( X = A \mod N \)
  - if \( X \) is the remainder of the integer division of \( A \) by \( N \)
- examples:
  - \( 7 \mod 5 = 2 \)
  - \( 13 \mod 5 = 3 \)
- optimal for computers because they work with finite integer numbers (e.g. 32 bit registers)

Inversion in modular arithmetic
- The inverse of a number \( X \) is that number \( X^{-1} \) that multiplied by \( X \) gives as result 1
- in normal arithmetic:
  - \( X = 5 \) implies \( X^{-1} = 1/5 \)
  - because \( 5 \times 1/5 = 1 \)
- in modular arithmetic (e.g. modulo 4):
  - \( X = 5 \) implies \( X^{-1} \mod 4 = \{ 5, 9, 13, ... \} \)
  - because \( 5 \times 5 \mod 4 = 25 \mod 4 = 1 \)
  - \( 5 \times 9 \mod 4 = 45 \mod 4 = 1 \)
  - ...
RSA - an example

- chosen $P=11$, $Q=19$ we have $N=209$ and $\Phi=180$
- $E$ (relative prime to $10$ and $18$ and $<180$) $= 31$
- $D = 31^{-1} \mod 180 = 151$
- $K_{pub} = (209, 31)$; $K_{pri} = (209, 151)$
- text to encrypt: $101323$ (note: $p_i < 209$)
  - $c_1 = 10^{31} \mod 209 = 32$
  - $c_2 = 13^{31} \mod 209 = 167$
  - $c_3 = 23^{31} \mod 209 = 199$
  - $p_1 = 32^{150} \mod 209 = 7$
  - $p_2 = 167^{150} \mod 209 = 2$
  - $p_3 = 199^{150} \mod 209 = 3$

RSA computational optimization (I)

- usually all public keys have $E=3$, 17 or 65537 ($0x10001$, the Fermat number)
  - the power operation is very easy because these numbers have only two bits set to one
    - (high) speed of the encryption operation
    - (high) speed in the operation of signature verification
  - optimized algorithms for this special case
- attack: provide a signature made with a key whose exponent has many bits set to one, to generate a high computational load

RSA computational optimization (II)

- in RSA the operations involving the private key (signing and decrypting) are slow
- the CRT (Chinese Remainder Theorem) makes them faster (4x) thanks to the equivalence $f(x) \mod N \sim f(x) \mod P \& f(x) \mod Q$
- it is a different representation of the private key (beware! we don't store $P$ and $Q$ but some derivatives useful in CRT computations)
- standardized in PKCS#1
- attack: makes RSA more susceptible to fault injection attacks
RSA weaknesses
- small encryption exponent
  - if \( E = 3 \) and send the same message to different recipients with plain RSA algorithm then an eavesdropper could recover the plaintext
- using the same key for encryption and signing
  - same underlying math for encryption and signing, only in reverse; if an attacker can convince a key holder to sign an unformatted encrypted message using the same key then she gets the original
- acting as an oracle
  - techniques to recover the plaintext if a user just blindly returns the RSA transformation of the input

Solutions to RSA weaknesses
- if using PKCS#1 v1.5 encoding, use \( E = 0x10001 \)
- always add fresh random padding (salt, at least 8 bytes) to the message before encrypting
- don't use the same RSA key for encryption and signing
- format the input before encrypting or signing (i.e. don't accept to encrypt or sign raw data)
- when decrypting, check the format of the decrypted block; if it is not as expected, return an error, not the decrypted string
- when verifying a signature, if there is any error whatsoever, just respond with "Invalid Signature"

Length of the public keys
- 512 bits can be attacked in some weeks
- 1024 bits can be attacked in some months
- 2048 bits offer an appropriate security level for several years
  - proved by means of RSA challenges:
Note1: with a patch on 9/10/2012 MS systems do not accept any more RSA keys < 1024 (KB2661254)
Note2: since 31/12/2013 Mozilla does not accept RSA < 2048 & MD5 (https://wiki.mozilla.org/CA:MD5and1024)
RSA challenges
- solved challenges (old style, size in decimal digits):
  - 10-apr-1996, RSA-130, 1000 MIPS-years
  - 2-feb-1999, RSA-140 (465 bits), 2000 MIPS-years
  - 22-aug-1999, RSA-155 (512 bits), 8000 MIPS-years
  - 9-may-2005, RSA-200 (663 bits), ~75 years Opteron 2.2 GHz
- solved challenges (new style, size in bits):
  - 3-dec-2003, RSA-576 (174 decimal digits)
  - 2-nov-2005, RSA-640 (193 decimal digits)
  - 12-dec-2009, RSA-768 (232 decimal digits)
    ~1500 anni Opteron 2.2 GHz; record on 30/9/2014

Twinkle (!?)

An Analysis of Shamir's Factoring Device
Robert D. Silverman
RSA Laboratories
May 3, 1999
At a Eurocrypt rump session, Professor Adi Shamir of the Weizmann Institute announced the design for an unusual piece of hardware. This hardware, called “TWINKLE” (which stands for The Weizmann Institute Key Locating Engine), is an electro-optical sieving device which will execute sieve-based factoring algorithms approximately two to three orders of magnitude as fast as a conventional fast PC. The announcement only presented a rough design, and there are a number of practical difficulties involved with fabricating the device. It runs at a very high clock rate (10 GHz), must trigger LEDs at precise intervals of time, and uses wafer-scale technology. However, it is my opinion that the device is practical and could be built after some engineering effort is applied to it. Shamir estimates that the device can be fabricated (after the design process is complete) for about $5,000.

Firmware signature for TI calculators
- TI83+ graphing calculator
- firmware protected by 512 bit RSA signature
- signature key factored on July 2009
  - 73 days of computation on a 1.9 GHz dual-core PC
  - on 1999 same computation required 8000 MIPS-years + Cray C916
- now all users may autonomously modify the firmware by themselves
RSA attack: a different approach

http://xkcd.com/538/

Key distribution for asymmetric cryptography

- private key never disclosed!
- public key distributed as widely as possible
- problem: who guarantees the binding (correspondence) between the public key and the identity of the person?
- solution #1: exchange of keys OOB (e.g. key party!)
- solution #2: distribution of the public key by means of a specific data structure named public key certificate (= digital certificate)
  - format of the certificate?
  - trust in the certificate issuer?

Secret key exchange by asymmetric algorithms

confidentiality without shared secrets is often used to send the secret key chosen for a symmetric algorithm
**Diffie-Hellman**

- A and B choose / agree two public integers \( p \) (prime, large) and \( g \) (generator) such that:
  - \( 1 < g < p \) (typically \( g = 2, 3, \) or \( 5 \))
- A arbitrarily chooses an integer \( x > 0 \) and computes:
  \( X = g^x \mod p \)
- B arbitrarily chooses an integer \( y > 0 \) and computes:
  \( Y = g^y \mod p \)
- A and B exchange (publish) \( X \) and \( Y \)
- A computes \( K_A = Y^x \mod p \)
- B computes \( K_B = X^y \mod p \)
  - but \( K_A = K_B = g^{xy} \mod p \)

**Diffie-Hellman (DH)**

\[
\begin{align*}
A &= g^x \mod p \\
B &= g^y \mod p \\
K_A &= B^x \mod p \\
K_B &= A^y \mod p \\
K_A = K_B &= g^{xy} \mod p = K_{AB}
\end{align*}
\]
DH: man-in-the-middle attack

\[ A = g^X \mod p \]
\[ B = g^Y \mod p \]
\[ K_A = M^X \mod p \]
\[ K_B = M^Y \mod p \]
\[ K_{AM} = g^{XZ} \mod p \]
\[ K_{BM} = g^{YZ} \mod p \]

DH, RSA and quantum computing

- **DH ~ complexity as discrete logarithm**
  - if \( A = g^x \mod p \) then \( x = \log_g A \mod p \)
  - successive products ~ time linear in \( p \) and hence exponential in the number of bits of \( p \)
  - other algorithms (e.g. Pohlig–Hellman, number field sieve) are better but never polynomial
  - algorithms to compute the discrete logarithm often can be adapted for factorization
  - Shor's algorithm is polynomial ~ \( O((\log N)^3) \), solves RSA and DH ... but requires a quantum computer!
  - (2012) computer with 7 qubit computed \( 2^{1} = 3 \cdot 7 \)

Elliptic curve cryptography

- **ECC (Elliptic Curve Cryptosystem)**
- instead of using modular arithmetic, the operations are executed on the surface of a 2D (elliptic) curve
- problem of discrete logarithm on such a curve
  - more complex than problem in modular arithmetic
  - possible to use shorter keys (about 1/10)
- digital signature = ECDSA
- key agreement = ECDH
- authenticated key agreement = ECMQV (patented)
- key distribution = ECIES (EC Integrated Encryption Scheme)
Elliptic curve arithmetics

- Elliptic curve: \( y^2 = x^3 + ax + b \pmod{p} \) with \( 4a^3 + 27b^2 \neq 0 \)
- Compute \( R = (x, y) = P + Q \) given:
  - \( P = (x_P, y_P) \)
  - \( Q = (x_Q, y_Q) \)
  - \( x = \lambda^2 - x_P - x_Q \)
  - \( y = \lambda(x_P - x) - y_P \)
  - \( \lambda = (y_P - y_Q) / (x_P - x_Q) \) if \( P \neq Q \) (i.e. \( P+Q \))
  - \( \lambda = (3x_P + a) / 2y_P \) if \( P = Q \) (i.e. \( 2P \))
- So we can compute:
  - Addition of two points
  - Multiplication of a point by a scalar

EC-Diffie-Hellman

- A and B select the same elliptic curve and a point \( G \) of its
- A chooses a random value \( x \) and computes:
  \( X = x G \)
- B chooses a random value \( y \) and computes:
  \( Y = y G \)
- A and B exchange (publish) \( X \) and \( Y \)
- A computes \( K = x Y \)
- B computes \( K' = y X \)
- But \( \text{if } \text{and only if } x = y \) then \( K = K' = x y G \)
- Note: uses only scalar multiplication of a point
**ECDSA e ECIES**

- **ECDSA**:
  - message digest computed with a normal hash function (e.g. SHA-256)
  - signature = pair of scalars derived from the digest plus some operations on the curve

- **ECIES**
  - generates a symmetric encryption key (e.g. an AES-128 one) with operations on the curve
  - gives to the receiver the information (based on his public key) needed to recompute the encryption key

---

**Sony PS3 hacking**

- PS3 has embedded Linux with loader verifying the binaries’ ECDSA signature before execution
- generation of an ECDSA signature requires a random nonce, otherwise from the signature the private key can be computed (!!!)
- … but Sony uses a fixed random (!!!)
- consequence: private key computed and distributed world-wide, so that anybody can run his own binaries on his PS3

"Console Hacking 2010 - PS3 Epic Fail" by fail0verflow
http://events.ccc.de/congress/2010/Fahrplan/events/4087.en.html

---

**Who's who in crypto**

<table>
<thead>
<tr>
<th>Adi Shamir</th>
<th>Ron Rivest</th>
<th>Len Adleman</th>
<th>Ralph Merkle</th>
<th>Martin Hellman</th>
<th>Whit Diffie</th>
</tr>
</thead>
</table>

---
Message integrity

- A person that intercepts an encrypted communication cannot read it...
- ... but can modify it in an unpredictable way!

Digest

**Digest**: sent data → ??? → received data

- Message digest
- Received message digest

- Digest OK?

Message digest and hash functions

- The message digest is a **fixed-length** “summary” of the message to be protected (of any length)
- It must be:
  - Fast to compute
  - Impossible or very difficult to invert
  - Difficult to create “collisions”
- Digest often used to avoid performing operations on the whole message, especially when the message is very large (e.g., because public-key cryptography is very slow)
- Digest can be calculated in many ways, but usually via a (cryptographic) hash function
Hash functions (dedicated)

usually:
- split the message M in N blocks M₁ ... Mₙ
- iteratively apply a base function (f)
- Vₖ = f (Vₖ₋₁, Mₖ) with V₀ = IV and h = Vₙ

message (split in blocks)

<table>
<thead>
<tr>
<th>IV</th>
<th>f</th>
<th>V₁</th>
<th>f</th>
<th>V₂</th>
<th>f</th>
<th>V₃</th>
<th>hash value</th>
</tr>
</thead>
</table>

Cryptographic hash algorithms

<table>
<thead>
<tr>
<th>name</th>
<th>block</th>
<th>digest</th>
<th>definition</th>
<th>note</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD2</td>
<td>8 bit</td>
<td>128 bit</td>
<td>RFC-1319</td>
<td>obsolete</td>
</tr>
<tr>
<td>MD4</td>
<td>512 bit</td>
<td>128 bit</td>
<td>RFC-1320</td>
<td>obsolete</td>
</tr>
<tr>
<td>MD5</td>
<td>512 bit</td>
<td>128 bit</td>
<td>RFC-1321</td>
<td>obsolete</td>
</tr>
<tr>
<td>RIPEMD</td>
<td>512 bit</td>
<td>160 bit</td>
<td>ISO/IEC 10118-3</td>
<td>good</td>
</tr>
<tr>
<td>SHA-1</td>
<td>512 bit</td>
<td>160 bit</td>
<td>FIPS 180-1</td>
<td>semi-good</td>
</tr>
<tr>
<td>SHA-224</td>
<td>512 bit</td>
<td>224 bit</td>
<td>RFC-3174</td>
<td></td>
</tr>
<tr>
<td>SHA-256</td>
<td>512 bit</td>
<td>256 bit</td>
<td>FIPS 180-2</td>
<td>good (?)</td>
</tr>
<tr>
<td>SHA-384</td>
<td>1024 bit</td>
<td>384 bit</td>
<td>/ RFC-4634</td>
<td></td>
</tr>
<tr>
<td>SHA-512</td>
<td>1024 bit</td>
<td>512 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SHA-1 broken

February 15, 2005
SHA-1 has been broken. Not a reduced-round version. Not a simplified version. The real thing.
The research team of Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu (mostly from Shandong University in China) have been quietly circulating a paper describing their results:
- collisions in the full SHA-1 in 2^69 hash operations, much less than the brute-force attack of 2^80 operations based on the hash length.
- collisions in SHA-0 in 2^39 operations.
- collisions in 58-round SHA-1 in 2^33 operations.
This attack builds on previous attacks on SHA-0 and SHA-1, and is a major, major cryptanalytic result. It pretty much puts a bullet into SHA-1 as a hash function for digital signatures (although it doesn’t affect applications such as HMAC where collisions aren’t important).
The paper isn’t generally available yet. At this point I can’t tell if the attack is real, but the paper looks good and this is a reputable research team.

http://www.schneier.com/blog/archives/2005/02/sha1_broken.html
Digest length

- important to avoid aliasing (=collisions):
  - \( \text{md1} = h(m1) \)
  - \( \text{md2} = h(m2) \)
  - if \( m1 \neq m2 \) then we'd like to have \( \text{md1} \neq \text{md2} \)
- if the algorithm is well designed and generates a digest of \( N \) bits, then the probability of aliasing is:
  \[ P_A \propto \frac{1}{2^{N \text{bit}}} \]
- thus, digests with many bits are required (because statistical events are involved)

The birthday paradox

- if there are at least 23 persons in the same room, then the probability that 2 of them were born in the same day is greater than 50\%; with 30 persons the probability is greater than 70%.
- why? subtract from certainty (1) the probability that the 2nd, 3rd, 4th, … person was not born on the same day of any of the preceding ones
  - \( P(2) = 1 - \frac{364}{365} \)
  - \( P(3) = 1 - \frac{364}{365} \cdot \frac{363}{365} \)
  - \( P(N) = 1 - \frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \left( \frac{365-N+1}{365} \right) / 365^{N-1} \)

The birthday attack

- a \( N \)-bit digest algorithm is insecure when more than \( 2^{N/2} \) digests are generated because the probability to have two messages with the same digest is \( P_A \approx 50\% \)
- a cryptosystem is “balanced” when the encryption and digest algorithms have the same resistance:
  - SHA-256 and SHA-512 have been designed for use respectively with AES-128 and AES-256
  - SHA-2 = (SHA-224, SHA-256, SHA-354, SHA-512)
  - SHA-224/-354 are the truncation of SHA-256/-512
  - SHA-256 uses a 32 bit word, SHA-512 uses 64 bit
  - note: SHA-1 (i.e. SHA-160) matched Skipjack-80
The NIST team praised the Keccak algorithm for its many admirable qualities, including its elegant design and its ability to run well on many different computing devices. The clarity of Keccak’s construction lends itself to easy analysis (during the competition all submitted algorithms were made available for public examination and criticism), and Keccak has higher performance in hardware implementations than SHA-2 or any of the other finalists.

<table>
<thead>
<tr>
<th>SHA-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>candidates: 64 (oct'08) &gt; 51 (dec'08) &gt; 14 (jul'09)</td>
</tr>
<tr>
<td>five finalists (dec'10)</td>
</tr>
<tr>
<td>- BLAKE, Grøstl, JH, Keccak, Skein</td>
</tr>
<tr>
<td>(2-oct-2012) and now the winner is …</td>
</tr>
<tr>
<td>– Keccak (pronounce: “catch-ack”)</td>
</tr>
<tr>
<td>authors = G.Bertoni, J.Daemen, G. Van Assche (STM), M.Peeters (NXP)</td>
</tr>
</tbody>
</table>

KDF (Key Derivation Function)

- a cryptographic key must be random (each bit has 50% probability to be 0 or 1)
- users typically insert passwords (or better passphrases) guessable and not random
- \( K = KDF \ (P, S, I) \)
  - \( P \) = password or passphrase
  - \( S \) = salt (to make \( K \) difficult to guess given \( P \))
  - \( I \) = no. of iterations of the base function (to slow down the computation and make life complex for attackers)
- KDF based upon cryptographic hash functions:
  - PBKDF2 (RFC-2898) uses SHA-1, \(|S| \geq 64, I \geq 1000\)
  - HKDF (RFC-5869) uses HMAC

MAC, MIC, MID

- to guarantee the integrity of messages, a code is added to the message:
  - MIC (Message Integrity Code)
- often integrity is not useful without authentication, thus the code (ensuring both security properties) is named:
  - MAC (Message Authentication Code)
- to avoid replay attacks, a unique identifier can be added to the message:
  - MID (Message IDentifier)
**Authentication by symmetric encryption**

- send also an encrypted copy of data
- only who knows the (secret) key can compare the copy with the original data
- disadvantage: the same data are sent twice

**Authentication by digest and symmetric encryption**

- send also an (encrypted) digest of the data
- A → B : mex, { digest (mex) } S
- only who knows the (secret) key can compare the transmitted digest with the digest calculated on the received data
- disadvantage:
  - two operations (digest + encryption)
- advantage:
  - few additional data

**Digest + symmetric encryption**

Sender

- sent data
- hash
- \( md \)
- symmetric encryption

Receiver

- received data
- hash
- \( md_R \)
- symmetric decryption
- \( md_R = ? \)
**Authentication by means of keyed-digest**

- send also a digest calculated not only on data but also on a secret key
- A → B : mex, digest (mex, S)
- only who knows the key can compare the transmitted digest with the digest calculated on the received data
- advantages:
  - only one operation (digest)
  - few additional data

**Keyed-digest: possible mistakes**

- if \( kd = H(K || M) \) then I can change the message adding at its end one or more blocks:
  - \( kd^* = H(K || M || M') = f(kd, M') \)
- if \( kd = H(M || K) \) then I can change the message adding before it a suitable block:
  - \( kd = H(M' || M || K) \) choosing \( M' \) s.t. IV = f(IV, M')
- protection:
  - insert in the digested data also the length of \( M \)
  - define \( kd = H(K || M || K) \)
  - use a standard keyed-digest
Keyed-digest: some standards

- RFC-1828 (historic) = keyed-md5
  - md5( K || keyfill || data || K || MD5fill )
- RFC-1852 (obsolete) = keyed-sha1
  - sha1( K || keyfill || datagram || K || SHAfill )
- RFC-2841 = keyed-sha1 (revised)
  - sha1( K || keyfill || data || datafill || K || sha1fill )

Keyed-digest: HMAC

- RFC-2104 (also FIPS-198)
  - base hash function H:
    - B byte block, L byte output, with B > L
  - definitions:
    - ipad = 0x36 repeated B times
    - opad = 0x5C repeated B times
    - deprecated those keys s.t. | K | < L
    - if | K | > B then K' = H( K ) else K' = K
    - if L < | K' | < B then K' is 0-padded up to B bytes
  - hmac = H( K' ⊕ opad || H( K' ⊕ ipad || data ) )

CBC-MAC

- exploits a block-oriented symmetric encryption algorithm, in CBC mode with null IV, taking as MAC the last encrypted block
- message M split in N blocks M1 … MN
- iterations:
  - V0 = 0
  - for (k=1…N) do Vk = enc( K, Mk ⊕ Vk-1 )
- cbc-mac = VN
- DES-based CBC-MAC is the Data Authentication Algorithm (standard FIPS 113, ANSI X9.17)
- secure only for fixed-length messages (for other cases use CMAC)
**Integrity and secrecy: how to combine?**

- distinct operations by hypothesis:
  - secrecy = symmetric encryption with $K_1$
  - integrity = keyed-digest (MAC) with $K_2$

- option 1 – authenticate-and-encrypt (A&E)
  - $\text{enc}(p, K_1) || \text{mac}(p, K_2)$
  - must always decrypt before checking integrity (possible DoS attack)
  - may leak info about the plaintext
  - e.g. used by SSH

- option 2 – authenticate-then-encrypt (AtE)
  - $\text{enc}(p || \text{mac}(p, K_2), K_1)$
  - must always decrypt before checking integrity (possible DoS attack)
  - e.g. used by SSL and TLS

- option 3 – encrypt-then-authenticate (EtA)
  - $\text{enc}(p, K_1) || \text{mac}(\text{enc}(p, K_1), K_2)$
  - can avoid decryption if MAC is wrong
  - e.g. used by IPsec

**Integrity and secrecy: security?**

- improper combination of secure algorithms may lead to ... an insecure result!
- authenticate-and-encrypt (A&E)
  - insecure unless performed in a single step
- authenticate-then-encrypt (AtE)
  - secure only with CBC or stream encryption
- encrypt-then-authenticate (EtA)
  - the most secure mode ... but beware of implementation errors
    - always include IV and algos in the MAC
- current efforts towards a joint AE algorithm
Authenticated encryption

- one single operation for privacy and authentication (and integrity):
  - just one key and one algorithm
  - better speed
  - less error likelihood in combining the two functions
- needed by applications (e.g. e-mail messages, network packets)
- the normal encryption modes are subject to chosen-ciphertext attacks when used on-line:
  - the attacker modifies a ciphertext
  - then observes if the receiver signals an error or not (e.g. padding oracle or decryption oracle attack)

RFC 5116 - Interface and algorithms for authenticated encryption

Authenticated Encryption with Associated Data (AEAD)

- error on every block after the mangled one
- easy to add a last control block (e.g. all zeroes, counter of the number of blocks)

IGE (Infinite Garble Extension)
Authenticated encryption: standards

- ISO/IEC 19772:2009 defines 6 standard modes:
  - OCB 2.0 (Offset Codebook Mode) [single-pass AEAD, patented]
  - AESKW (AES Key Wrap)
  - CCM (CTR mode with CBC-MAC)
  - EAX (Encrypt then Authenticate then X(translate)) = CTR + OMAC [double-pass AEAD, free]
  - Encrypt-then-MAC
  - GCM (Galois/Counter Mode)
- other modes exist and are/will be recommended by other bodies (e.g. NIST, IETF)

Authenticated encryption: applications

- 802.11i uses CCM
- ZigBee uses CCM* (=CCM + auth-only + enc-only)
- ANSI C12.22 (network transmission of electronic measures, e.g. house power meter) uses EAX'
  - broken !!! Minematsu, Morita and Iwata
  - EAX had a formal proof of its security … but ANSI sacrificed it for 3-5 less encryption steps and 40 bytes less memory usage (so important for embedded systems?)

Comparison of AE algorithms

- GCM: the most popular, on-line single-pass AEAD, parallelizable, used by TLS, present in openssl
- OCB 2.0: the fastest one, on-line single-pass AEAD, patented (butGPL) so scarcely used, now free for all uses but military ones
- EAX: on-line double-pass AEAD, slow but small (uses just the encryption block) so very good for constrained systems
- CCM: off-line double-pass, the slowest one
- note: double-pass is 2x slower than one-pass (in software)
Competition for AE

- CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness
- http://competitions.cr.yp.to/caesar.html
- final selection expected on December 2017
- will select an algorithm portfolio

NIST recommendations for using block encryption algorithms

- NIST SP 800-38
- part A (dec'01) = five confidentiality modes
  - ECB, CBC, CFB, OFB, and CTR
- addendum (oct'10) = three CBC ciphertext stealing
- part B (may'05) = one authentication mode
  - CMAC (~OMAC > XCBC > CBC-MAC)
- part C (jul'07) = auth. encryption = CCM
- part D (nov'07) = high-speed auth. enc. = GCM
- part E (jan’10) = confidentiality for block memories = XTS-AES
- draft = use of AESKW for auth. enc. of keys

Authentication by digest and asymmetric cryptography

- send also a digest (encrypted with the private key of the sender)
- those who know the public key can compare the transmitted digest with the digest calculated on the received data
- A → B : mex, { digest(mex) } priA

- DIGITAL SIGNATURE !!!
Digital signature

sender's key pair

PRI               PUB

hash

md

asymmetric encryption

sender sent data

hash

asymmetric decryption

receiver received data

hash

mdR

mdS

hash

receiver received data

mdR = ?

digital signature

Signature creation and verification

signer

data

digest

D

asymmetric encryption

private key of signer

verifier

data

data

digest

D

asymmetric encryption

public key of signer

signature

R

data

RSA signatures and hash functions

- the hash function to be used in a RSA-based signature schema must be:
  - resistant to collisions (obvious, even just to avoid generating accidentally the same signature)
  - difficult to invert (less obvious)
    - to create a fake signature of the key (E, N)
    - ... choose S randomly
    - ... compute R = S^E mod N
    - ... find X such that h(X) = R, that is X = h^{-1}(R)
    - ... we may state that S is the digital signature of X verifiable with the public key (E, N)
PKCS#1

- (nov'93) v1.5 ~ RFC-2313
- (sep'98) v2.0 ~ RFC-2437
- (jun'02) v2.1 ~ RFC-3447
- Definition of primitives for using RSA:
  - Conversion and representation of large integers
  - Base algorithms for ciphering/deciphering
  - Base algorithms for signature creation/verification
- These primitives must be used in an appropriate way to create a secure "cryptographic schema"

PKCS#1 schemas

- Encryption schemas:
  - RSAES-OAEP (Optimal Asymmetric Encryption Padding, by Bellare and Rogaway)
  - RSAES-PKCS1-v1_5 (old base schema)
- Signature (with appendix) schemas:
  - RSASSA-PSS (Probabilistic Signature Scheme, by Bellare and Rogaway)
  - RSASSA-PKCS1-v1_5 (old base schema)
  - Related encodings (EMSA-PSS, EMSA-PKCS1-v1_5)
- The signature schemas are "with appendix" because RSA does not sign the data rather their "digest" (hash) since RSA handles directly only data < N

Authentication and integrity: analysis

- By means of a shared secret:
  - Useful only for the receiver
  - Cannot be used as a proof without disclosing the secret key
  - Not useful for non repudiation
- By means of asymmetric encryption:
  - Being slow it is applied to the digest only
  - Can be used as a formal proof
  - Can be used for non repudiation
  - = Digital signature
Digital vs. handwritten signature

- digital signature = authentication + integrity
- handwritten signature = authentication
- thus the digital signature is better, because it is tightly bound to the data
- note: each user does not have a digital signature but a private key, which can be used to generate an infinite number of digital signatures (one for each different document)

Public key certificate

"A data structure used to securely bind a public key to some attributes"

- typically it binds a key to an identity ... but other associations are possible too (e.g. IP address)
- digitally signed by the issuer: the Certification Authority (CA)
- limited lifetime
- can be revoked on request both by the user and the issuer

Formats for public key certificates

- X.509:
  - v1, v2 (ISO)
  - v3 (ISO + IETF)
- non X.509:
  - PGP
  - SPKI (IETF)
- PKCS-6:
  - RSA, partly compatible with X.509
  - obsolete
Structure of a X.509 certificate
- version: 2
- serial number: 1231
- signature algorithm: RSA with MD5, 1024
- issuer: C=IT, O=Polito, OU=CA
- validity: 1/1/97 - 31/12/97
- subject: C=IT, O=Polito, CN=Antonio Lioy
  Email=lioy@polito.it
- subjectPublicKeyInfo: RSA, 1024, xx...x
- CA digital signature: yy...y

PKI (Public-Key Infrastructure)
- is the infrastructure ...
- technical and administrative ...
- put in place for the creation, distribution and revocation of public key certificates

Certificate revocation
- any certificate can be revoked before its expiration date:
  - on request from the owner (subject)
  - autonomously by the creator (issuer)
- when a signature is verified, the receiver must check that the certificate was valid at signature time
- this kind of check is the responsibility of the receiver (relying party, RP)
Revocation mechanisms

- CRL (Certificate Revocation List)
  - list of revoked certificates
  - signed by the CA or by a delegated party
- OCSP (On-line Certificate Status Protocol)
  - response containing information about the certificate status
  - signed by the server

Structure of a X.509 CRL

1

- version
- signature algorithm RSA with MD5, 1024
- C=IT, O=Polito, OU=CA
- thisUpdate 15/10/2000 17:30:00
- userCertificate 1496
  - revocationDate 13/10/2000 15:56:00
- userCertificate 1574
  - revocationDate 4/6/1999 23:58:00
- CA digital signature yy...y

Verification of a signature / certificate

- how to verify that a public-key certificate (signed by CA1) is authentic?
  - the public-key certificate of CA1 is required (which will be signed by CA2)
- how to verify the last one?
  - the public-key certificate of CA2 is required (which will be signed by CA3)
  - and so on...
  - it becomes thus necessary to have an infrastructure (hierarchical?) for certification and distribution of public-key certificates
Verification of a signature / certificate

- data
- signature (of X)
- hash
- digest (data)
- decrypt
- digest (extracted from signature)
- certificate subject: X
- key: KPUB(X)
- signature (of CA)
- KPUB(X)
- digest (extracted from signature)
- = ?

Certification hierarchy

- CA EU
  - ds (EU)
  - CA IT
    - ds (EU)
  - CA ES
    - ds (EU)
- CA Torino
  - ds (IT)
- CA Milano
  - ds (IT)
- TLCA = Top-Level CA
  - (Trusted) Root CA
  - "God"

EuroPKI

- EuroPKI Austria
- EuroPKI Slovenia
- EuroPKI Italy
- EuroPKI TLCA
- Politecnico di Torino CA
- CSP CA
- Comune di Modena CA
- persons
- server
Performance

- Cryptographic performance does not depend on RAM but on CPU (architecture and instruction set) and cache size.
- Performance is not a problem on clients (except when they are overloaded by local applications).
- Performance can become a problem on the servers and/or on the network nodes (e.g., router):
  - Use cryptographic accelerators.
  - Special-purpose accelerators (e.g., SSL, IPsec) or generic ones.

Performance (I5 @ 3.1 GHz)

<table>
<thead>
<tr>
<th></th>
<th>[64 B/packet]</th>
<th>[1024 B/packet]</th>
</tr>
</thead>
<tbody>
<tr>
<td>hmac (md5)</td>
<td>127.4 MB/s</td>
<td>532.7 MB/s</td>
</tr>
<tr>
<td>des cbc</td>
<td>79.7 MB/s</td>
<td>82.9 MB/s</td>
</tr>
<tr>
<td>des ede3</td>
<td>28.9 MB/s</td>
<td>29.3 MB/s</td>
</tr>
<tr>
<td>aes-128</td>
<td>131.2 MB/s</td>
<td>136.7 MB/s</td>
</tr>
<tr>
<td>rc4</td>
<td>660.6 MB/s</td>
<td>726.1 MB/s</td>
</tr>
</tbody>
</table>

rsa 1024 1558.1 signs/s 26403.1 verifies/s
rsa 2048 236.9 signs/s 7576.5 verifies/s

Performance (P4 @ 1.7 GHz)

<table>
<thead>
<tr>
<th></th>
<th>[64 B/packet]</th>
<th>[1024 B/packet]</th>
</tr>
</thead>
<tbody>
<tr>
<td>hmac (md5)</td>
<td>31.5 MB/s</td>
<td>152.1 MB/s</td>
</tr>
<tr>
<td>des cbc</td>
<td>28.7 MB/s</td>
<td>28.9 MB/s</td>
</tr>
<tr>
<td>des ede3</td>
<td>10.8 MB/s</td>
<td>10.9 MB/s</td>
</tr>
<tr>
<td>aes-128</td>
<td>38.0 MB/s</td>
<td>37.8 MB/s</td>
</tr>
<tr>
<td>rc4</td>
<td>61.2 MB/s</td>
<td>62.0 MB/s</td>
</tr>
</tbody>
</table>

rsa 1024 133.7 signs/s 2472.1 verifies/s
**Performance (P3 @ 800 MHz)**

<table>
<thead>
<tr>
<th></th>
<th>64 B/packet</th>
<th>1024 B/packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>hmac(md5)</td>
<td>19.2 MB/s</td>
<td>83.6 MB/s</td>
</tr>
<tr>
<td>des cbc</td>
<td>14.4 MB/s</td>
<td>14.5 MB/s</td>
</tr>
<tr>
<td>des ede3</td>
<td>5.2 MB/s</td>
<td>5.2 MB/s</td>
</tr>
<tr>
<td>aes-128</td>
<td>15.6 MB/s</td>
<td>15.9 MB/s</td>
</tr>
<tr>
<td>rc4</td>
<td>80.9 MB/s</td>
<td>86.4 MB/s</td>
</tr>
<tr>
<td>rsa 1024</td>
<td>94.8 signs/s</td>
<td>1682.0 verifies/s</td>
</tr>
</tbody>
</table>

**NSA suite B**
- for COTS products that treat SBU (Sensitive But Unclassified) and Classified information
- contains the following algorithms:
  - (symmetric encryption) AES-128 e AES-256
  - (hash) SHA-256 e SHA-384
  - (key agreement) ECDH e ECMQV
  - (digital signature) ECDSA
- for information up to Secret level:
  - AES-128 + SHA-256 + EC P-256
- for information at Top Secret level:
  - AES-256 + SHA-384 + EC P-384

**Length of keys and digest**
- equivalence defined in NIST SP800-57 (part 1, rev.3)
- FFC = Finite Field Cryptography (e.g. DSA, D-H)
- IFC = Integer Factorization Cryptography (e.g. RSA)

<table>
<thead>
<tr>
<th>symm.</th>
<th>FFC</th>
<th>IFC</th>
<th>ECC</th>
<th>hash</th>
<th>years</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>1024</td>
<td>1024</td>
<td>160</td>
<td>160</td>
<td>&lt; 2010</td>
</tr>
<tr>
<td>112</td>
<td>2048</td>
<td>2048</td>
<td>224</td>
<td>224</td>
<td>&lt; 2030</td>
</tr>
<tr>
<td>128</td>
<td>3072</td>
<td>3072</td>
<td>256</td>
<td>256</td>
<td>&gt; 2030</td>
</tr>
<tr>
<td>192</td>
<td>7680</td>
<td>7680</td>
<td>384</td>
<td>384</td>
<td>&gt;&gt; 2030</td>
</tr>
<tr>
<td>256</td>
<td>15360</td>
<td>15360</td>
<td>512</td>
<td>512</td>
<td>&gt;&gt;&gt; 2030</td>
</tr>
</tbody>
</table>

© Antonio Lioy - Politecnico di Torino (2005-2014)
Length of keys and digest
(ECRYPT II, 2011)

<table>
<thead>
<tr>
<th></th>
<th>symm.</th>
<th>hash</th>
<th>asymm.</th>
<th>ECC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>160</td>
<td>1248</td>
<td>160</td>
<td>very short term</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>192</td>
<td>1776</td>
<td>192</td>
<td>legacy</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>224</td>
<td>2432</td>
<td>224</td>
<td>medium term</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>256</td>
<td>3248</td>
<td>256</td>
<td>long term (2011-40)</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>512</td>
<td>15424</td>
<td>512</td>
<td>foreseeable future (quantum computers)</td>
<td></td>
</tr>
</tbody>
</table>

www.keylength.com

Why don’t we buy everything from USA?

- export of cryptographic material is subject to the same restrictions as nuclear material (!)
- … unless the protection level is very low:
  - symmetric key restricted to 40 bits (2^{40} trials = few CPU hours)
  - asymmetric key restricted to 512 bits
- example: Netscape, Internet Explorer, … (export version)

Key escrow: a novelty?

- December 1996: the USA government authorizes the export of semi-robust (56 bits) cryptographic products if they incorporate key-escrow functions
- does not apply to internal USA products
- key escrow = possibility to recover a key even without the consent of the owner
- example: Lotus Notes 4.x was using 64 bits symmetric keys, but 24 bits of them were encrypted with the NSA public-key
- problem: who decides when it is necessary to recover a key?
The many editions of Notes

Aside from encryption process time, U.S. government export laws limit encryption key length. These laws are the driving force behind the three major editions of Notes: North American, International, and French. Despite the different names, the product functionality is exactly the same. The difference, however, lies in the length of the keys used for encryption.

The North American edition uses encryption keys that are 64-bits long. The U.S. Government, for reasons of national security, limits the length of encryption keys for export to 40 bits. To comply with these restrictions, we have the International edition. When we generate a 64-bit key for the International edition, the top 24 bits are encrypted using the U.S. Government's public key and stored in what is called the Workfactor Reduction Field (WRF). Splitting the key in this manner results in a key that's 40 bits for the U.S. Government and 64 bits for everyone else. This approach maintains a high level of security worldwide without violating the export laws of the U.S. Government.

Most countries are content with the way the International edition complies with U.S. encryption key export laws. The government of France, however, found the International edition unacceptable. To comply with French law, we created the French edition, which uses a plain 40-bit encryption key and can therefore be "broken" by attackers willing to apply considerable computing power (presumably, including the French government).

Changes in the USA cryptographic export regulations

- June 1997:
  - permission to export secure web client and server web only if used by foreign branches of USA companies or in financial environment (transactions)
  - to verify the real use, special certificates issued by Verisign must to be used

- September 1998:
  - permission extended to insurance and health institutions
  - no permission for keys up to 56 bits

Step-up (gated) cryptography

user

browser web

weak crypto

strong crypto

bank

server web

weak + strong crypto

Verisign cert
Key recovery: another novelty?

- export from USA of products with strong cryptography (e.g. 128 bits symmetric keys) is allowed if:
  - they incorporate key-recovery functions
  - with a recovery centre authorized by the USA government
  - the symmetric keys used by the users are encrypted with the public key of the recovery centre
  - in this way it becomes a political problem but key-recovery is a real problem

New USA export regulation

- January 2000
- permission to export ...
  - off-the-shelf products ...
  - that passed a “one-time review”
- or
  - products whose source code is freely available in Internet
- upgrades available for the main commercial products
- doubts about the existence of back-doors

Import / export / domestic controls

- controls still exist in many countries
- Bert-Jaap Koops produces an excellent survey: http://www.cryptolaw.org/
The largest European hacker club, “Chaos Computer Club” (CCC), has reverse engineered and analyzed a “lawful interception” malware program used by German police forces.

It has been found in the wild and submitted to the CCC anonymously.

The malware can not only siphon away intimate data but also offers a remote control or backdoor functionality for uploading and executing arbitrary other programs.

Significant design and implementation flaws make all of the functionality available to anyone on the Internet.


Bibliography

- B.Schneier: “Applied cryptography”
- W.Stallings: “Cryptography and network security”
- C.P.Pfleeger, S.Pfleeger: “Security in computing”
- S.Garfinkel, G.Spafford: “Practical Unix and Internet security”
- W.R.Cheswick, S.M.Bellovin: “Firewalls and Internet security” (2nd ed.)
Bibliografia (in Italiano)

- W. Stallings  
  "Sicurezza delle reti - applicazioni e standard"  
  Addison-Wesley Italia, 2004

- C. Pfleeger, S. Pfleeger  
  "Sicurezza in informatica"  
  Pearson Education Italia, 2004

- Fugini, Maio, Plebani  
  "Sicurezza dei sistemi informativi"  
  Apogeo

- S. Singh  
  "Codici e segreti"  
  BUR saggi