Basics of ICT security

Diana Berbecaru
< diana.berbecaru @ polito.it >

Politecnico di Torino
Dip. Automatica e Informatica

Cryptography (terminology)

- Crypt = (Greek: κρύπτω kryptó) = hidden/secret
- Graph = (Greek: γράφω gráfo) = writing

- Cryptography is the science / art of transforming meaningful information into unintelligible text
- Cryptanalysis is the science / art of breaking cryptographic codes
- Cryptology is the science / art / study of both cryptography and cryptanalysis

What is Cryptography anyway?

- Initially, cryptography was concerned solely with message confidentiality (i.e., encryption) = conversion of messages from a comprehensible form into an incomprehensible one, and back again at the other end, rendering it unreadable by interceptors or eavesdroppers without secret knowledge.
- Lately, the field expanded to include techniques for message integrity checking, sender/receiver identity authentication, digital signatures, a.s.o.

Cryptography refers to numerical algorithms, implementation of those algorithms and various mathematical and programming tools to meet security goals.
Cryptography

sender
message (in clear)
key-1

crypt
encryption

receiver
message (in clear)
key-2

crypt
decryption

message (encrypted)

Terminology

- message in clear:
  - plaintext
  - 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; 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 to keep secret the encryption and decryption algorithms

- on the contrary it is better to make the algorithms public so that they can be widely analysed and their possible weaknesses discovered

Security through obscurity (STO)

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

Secret key cryptography
- a single key
- 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!)
- \( C = \text{enc}(K, P) \) or \( C = \{ P \} K \)
- \( P = \text{dec}(K, C) = \text{enc}^{-1}(K, C) \)
### Symmetric algorithms

<table>
<thead>
<tr>
<th>name</th>
<th>key (bit)</th>
<th>block (bit)</th>
<th>note</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES</td>
<td>56</td>
<td>64</td>
<td>obsolete</td>
</tr>
<tr>
<td>3-DES</td>
<td>112</td>
<td>64</td>
<td>56-112 bit strength</td>
</tr>
<tr>
<td>3-DES</td>
<td>168</td>
<td>64</td>
<td>112 bit strength</td>
</tr>
<tr>
<td>IDEA</td>
<td>128</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>RC2</td>
<td>8-1024</td>
<td>64</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</td>
<td>1-256</td>
<td>optimal when B=2W</td>
</tr>
<tr>
<td>AES</td>
<td>128-256</td>
<td>128</td>
<td>alias Rjindael</td>
</tr>
</tbody>
</table>

### Block vs. Stream ciphers
- **Block ciphers** process messages in blocks, each of which is then en/decrypted. Many ciphers are block ciphers, and they can be used in modes of operation that result in the same behavior as stream ciphers.
- **Stream ciphers** process messages one bit or byte (8 bits) at a time when encrypting/decrypting.

### Block algorithms
- Encryption of one block of data at a time.
- Need to partition the message into blocks.
- Need to pad small blocks.
- Key K is used many times before it is changed.
- E should scramble the data and the key as well.
- With enough text and time – key can be revealed.
Block algorithms features

- **block size**: in general larger block sizes mean greater security.
- **key size**: larger key size means greater security (larger key space).
- **number of rounds**: multiple rounds offer increasing security.
- **encryption modes**: define how messages larger than the block size are encrypted, very important for the security of the encrypted message.

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>⊕</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></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 bit)**
- **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 \( K_1 = K_2 = K_3 \))
- 3DES with 2 keys
  \[ C' = \text{enc}(K_1, P) \quad C'' = \text{dec}(K_2, C') \quad C = \text{enc}(K_1, C'') \]
- 3DES with 3 keys
  \[ C' = \text{enc}(K_1, P) \quad C'' = \text{dec}(K_2, C') \quad C = \text{enc}(K_3, C'') \]
- standard FIPS 46/3 and ANSI X9.52

---

**IDEA**

- International Data Encryption Algorithm
- patented but with low royalty (only for commercial use)
- 128 bits key
- 64 bits data block
- famous because used in PGP
- operations used:
  - XOR
  - addition modulo 16
  - multiplication modulo \( 2^{16} + 1 \)

---

**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
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
  - for best performance the RC5 word size (W) should match the register size of the CPU, e.g. 32 bit word will produce a B=64 bit block size
- operations used:
  - shift, rotate, modular addition
- used in Wireless Application Protocol – WAP (enables access to Internet from a mobile phone or PDA)

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:
  \[ C_i = \text{enc} (K, P_i) \]
- suggested NOT to be used on long messages
- uses: secure transmission of single values (e.g. session key encrypted using a master key)
ECB - decrypt

- formula for the i-th block:
  \[ P_i = \text{enc}^{-1}(K, C_i) \]

Pros and cons of ECB

- repetitions in message may show in ciphertext
  - particularly with data such as graphics
  - or with messages that change very little
- reordering ciphertext results in reordered plaintext
- errors in one block do not propagate
- main use of ECB is sending a few blocks of data

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)
- an error in transmission generates an error at the decryption of two blocks
CBC - decryption

- formula for the i-th block:
  \[ P_i = enc^{-1}(K, C_i) \oplus C_{i-1} \]
- requires \( C_0 \) (i.e. IV) to be known by the receiver
- uses: bulk data encryption, authentication

Pros and cons of CBC Mode

- randomized encryption: repeated text gets mapped to different encrypted data
- reordering: ciphertext block depends on all preceding plaintext blocks; reorder affects decryption
- error propagation: error in transmission of one (cipher)block generates errors of the decryption of two blocks
- sequential encryption: cannot use parallel hardware
- usage: requires to choose a random IV and to protect the integrity of IV

Padding (aligning)

- dimension of algorithm’s block \( B \)
- size of data to process \( D < B \)
- add bits until dimension \( B \) is reached

- problems:
  - value of padding bits? (padding techniques)
  - transmit more data \( B \) than the minimum required \( D \)
Padding techniques
- (if length is known or it can be obtained – e.g. a C string) add null bytes
  - … 0x00 0x00 0x00
- (original DES) one 1 bit followed by many 0
  - … 10000000
- one byte with value 128 followed by null bytes
  - … 0x80 0x00 0x00
- last byte’s value equal to the length of padding
  - … 0x?? 0x?? 0x03
  - what about the value of the other bytes?

Padding with explicit length (N)
- (Schneier) null bytes:
  - e.g. … 0x00 0x00 0x03
- (SSL/TLS) bytes with value N:
  - 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 N-1:
  - e.g. … 0x02 0x02 0x02

Use block algorithms to construct stream algorithms
- Cipher Feedback (CFB)
- Output Feedback (OFB)
- Counter Mode (CTR) – the most used
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 an entire block

\[
P_{bi} \oplus C_{bi} \leftarrow K
\]

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

\[
P_{bi} \oplus C_{bi} \leftarrow K
\]
**Output FeedBack (OFB) of DES**

Source: W. Stallings: Cryptography and Network Security

**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, …) shared by sender and receiver
- a transmission error — causes error only in one group

\[
P_{bi} \oplus \text{nonce } \oplus \text{ counter (1 byte)} \downarrow K \downarrow \text{leftmost byte} \downarrow \Phi \rightarrow C_{bi}
\]

**Stream algorithms**
- operate on a stream of data without requiring the division on blocks, typically on one bit or one byte at a time
- message is encrypted as it comes
- ideal algorithm:
  - one-time pad (requires a key which is as long as the message to protect!)
- real algorithms:
  - use pseudo-casual key generators, synchronized between the sender and the receiver
  - example: RC4
  - with enough text and time — the key can be revealed
Algorithms of type stream

- key generator
- seed
- key stream
- plaintext data stream
- encryption
- ciphertext data stream
- decryption
- plaintext data stream

Course: Computer Systems Security – 2012

OTP – One-Time Pad

- message M of length n bits
- key K of length n bits (same as M)
- key K is random bit by bit
- key K is used only once
- key K is known only to sender and receiver

- encryption (by sender): \( C = M \oplus K \) (bit-wise XOR)
- decryption (by receiver): \( M = C \oplus K \) (bit-wise XOR)

Course: Computer Systems Security – 2012

One-Time Pad

Examples of paper one-time pads


Course: Computer Systems Security – 2012

© Diana Berbecaru - Politecnico di Torino (2007-2012)
**OTP – One-Time Pad**
- If a truly random key as long as the message is used, the cipher will be secure.
- Is **unbreakable** since ciphertext bears no statistical relationship to the plaintext (the ciphertext does not reveal any information about the plaintext).
- Since for any plaintext & any ciphertext there exists a key mapping one to other.
- Unconditional security! Why look any further??
  - Key distribution and protection
  - Practical problem of making large quantities of random keys.

**Symmetric encryption requirements**
- Two requirements for secure use of symmetric encryption:
  - A strong encryption algorithm
  - A secret key known only to sender / receiver
- Assume encryption algorithm is known
- Implies a **secure channel** to distribute key.

**Symmetric cryptography**
- Single and secret key
- One key for each couple / group of users

© Diana Berbecaru - Politecnico di Torino (2007-2012)
Key distribution for symmetric cryptography

- for a complete private communication between \( N \) parties, \( \frac{N \times (N-1)}{2} \) 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 – \( N \) bit in length – are kept secret
- … then the only possible attack is the brute force (exhaustive) attack which requires a number of trials equal to \( 2^N \) bit

Length of cryptographic keys

| Symmetric | 40  | 64  | 128 | ...
|-----------|-----|-----|-----|-----
| Asymmetric| 256 | 512 | 1024| ...

© Diana Berbecaru - Politecnico di Torino (2007-2012)
**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%)
  - 1 special-purpose processing system (DES Cracker) developed by Electronic Frontier Foundation - EFF (cost=250,000 $) plus a few thousands of workstations of various types
    - 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
- A German bank condemned 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)
- The USA government initiated the procedure for selecting a new symmetrical algorithm: the 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](http://www.nist.gov/aes)

AES = RIJNDAEL
- 2 October 2000
- RIJNDAEL chosen as winner
- Published in November 2001 as FIPS-197
Public key cryptography

- key-1 ≠ key-2
- asymmetric algorithms
- pair of keys (public and private)
- one of the keys is used for encryption and the other one is 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 (Kpri, Kpub)
- keys with inverse functionality; data encrypted with one key can be decrypted only with the other key

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 with the other key
**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 **authentication and integrity** of data

![Digital signature diagram]

**Confidentiality without shared secrets**

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

![Confidentiality without shared secrets diagram]

**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)
RSA – the algorithm

- public module $N = P \times Q$ known to anybody
- $P$ and $Q$ are prime, large and secret
- public key $E$ arbitrarily chosen so that it is relatively prime with respect to $P-1$ and $Q-1$
- private key $D = E^{-1} \mod (P-1) \times (Q-1)$

- text to encrypt: $t < N$
- encrypt: $c = t^E \mod N$
- decrypt: $t = c^D \mod N$
- roles of $E$ and $D$ are interchangeable because $(x^D)^E \mod N = (x^E)^D \mod N$

Modular arithmetic

- $X = A \mod N$
- if $X$ is the rest of the integer division of $A$ by $N$
- examples:
  - $7 \mod 5 = 2$
  - $13 \mod 5 = 3$
- optimal for security applications because it is not invertible in a unique way:
  - given $Y \mod 5 = 2$
  - who is $Y$?
  - responses: $7, 12, 17, 22, 27, ...$

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 $5 \times 5 \mod 4 = 25 \mod 4 = 1$
  - $5 \times 9 \mod 4 = 45 \mod 4 = 1$

© Diana Berbecaru - Politecnico di Torino (2007-2012)
RSA - an example

- chosen \( P=3, Q=5 \) we have \( N = 15 \)
- \( E \) (relative prime to 2 and 4) = 7
- \( D = 7^{-1} \mod 8 = \{7, 15, 23, 31, \ldots\} = 23 \)

- text to encrypt: 1 2 3
- \( c_1 = 1^7 \mod 15 = 1 \)
- \( c_2 = 2^7 \mod 15 = 128 \mod 15 = 8 \)
- \( c_3 = 3^7 \mod 15 = 2187 \mod 15 = 12 \)
- \( t_1 = 1^{23} \mod 15 = 1 \)
- \( t_2 = 8^{23} \mod 15 = 2 \)
- \( t_3 = 12^{23} \mod 15 = 3 \)

DoS attack on RSA

- usually all public keys have exponent equal to 3 or to the Fermat number 65537 (0x00010001)
- 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

Length of the public keys

- 256 bits can be attacked in a couple of weeks
- 512 bits can be attacked in a couple of months
- 1024 bits offer an appropriate security level for a couple of centuries

- experimentally proved by means of the RSA challenges:
RSA challenges

- solved challenges (old style):
  - 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):
  - 3-dec-2003, RSA-576 (174 decimal digits)
  - 2-nov-2005, RSA-640 (193 decimal digits)

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
- 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

© Diana Berbecaru - Politecnico di Torino (2007-2012)
**Diffie-Hellman**

- A and B choose two big integers \( n \) and \( g \) so that: 
  \[ 1 < g < n \]
- A chooses a number \( x \) arbitrarily and computes: 
  \[ X = g^x \bmod n \]
- B chooses a number \( y \) arbitrarily and computes: 
  \[ Y = g^y \bmod n \]
- A and B exchange among themselves (publish) \( X \) and \( Y \)
- A computes \( K = Y^x \bmod n \)
- B computes \( K' = X^y \bmod n \)
- but \( K = K' = g^{xy} \bmod n \)

**Diffie-Hellman (DH)**

\[ A = g^x \bmod n \]
\[ B = g^y \bmod n \]

\[ A \rightarrow \quad X \]
\[ B \rightarrow \quad Y \]

\[ K_A = B^x \bmod n \]
\[ K_B = A^y \bmod n \]

\[ K_A = K_B = g^{xy} \bmod n = K_{AB} \]

**Diffie-Hellman**

- first public-key algorithm invented
- frequently used to agree on a secret key (key agreement)
- patented in the USA but the patent expired on 29 April 1997
- resistant to the sniffing attack
- if the attacker can manipulate the data then it is possible to make a man-in-the-middle attack; in this case it requires pre-authentication
  - certificates for DH keys
  - authenticated DH = MQV (Menezes-Qu-Vanstone) patented by CertiCom
DH: man-in-the-middle attack

A = g^x mod n

B = g^y mod n

M = g^z mod n

K_A = M^x mod n

K_B = M^y mod n

K_{AM} = g^{xz} mod n ≠ K_{BM} = g^{yz} mod n

Who's who in crypto

Adi Shamir
Ron Rivest
Len Adleman
Ralph Merkle
Martin Hellman
Whit Diffie

Message integrity

- an attacker that intercepts an encrypted communication cannot read it ...
- ... but can modify it in an unpredictable way!
- problem: B tries to decrypt the message but it doesn’t understand what A says and consequently B could take erroneous decisions

we meet at 19:30

we meet x?k13+e7#
Message integrity with digests

- **sent data** → ??? → **received data**
- **message digest (h1)**
- **message digest (h2)**
- **digest OK?**

- sender computes the value \( h_1 = h(M) \) and sends it along with the message \( M \)
- receiver:
  - computes \( h_2 = h(M) \)
  - checks if \( h_1 = h_2 \) ?
  - Yes accept the message, no reject the message

Message digest and hash functions

- the message digest is a **fixed-length** “summary” of the message to be protected
- it must be:
  - fast to compute
  - difficult to invert
- often digests are used in order to avoid to perform operations on the whole message, especially when the message is very long (e.g. because public-key cryptography is very slow)
- digest can be calculated in many ways, but usually a **hash function** is used

Hash functions (dedicated)

- **usually**:
  - split the message \( M \) in \( N \) blocks \( M_1 \ldots M_N \)
  - iteratively apply a base function \( f \)
  - \( V_k = f(V_{k-1}, M_k) \) with \( V_0 = IV \) and \( h = V_N \)
**Example: MD5 algorithm**

IV is fixed:
- Word A: 01 23 45 67
- Word B: 89 ab cd ef
- Word C: fe dc ba 98
- Word D: 76 54 32 10

---

**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>good</td>
</tr>
<tr>
<td>RIPEMD</td>
<td>512 bit</td>
<td>160 bit</td>
<td>ISO/IEC 10118-3</td>
<td>optimum</td>
</tr>
<tr>
<td>SHA-1</td>
<td>512 bit</td>
<td>160 bit</td>
<td>FIPS 180-1</td>
<td>RFC-3174</td>
</tr>
<tr>
<td>SHA-224</td>
<td>512 bit</td>
<td>224 bit</td>
<td>FIPS 180-2</td>
<td>optimum</td>
</tr>
<tr>
<td>SHA-256</td>
<td>512 bit</td>
<td>256 bit</td>
<td>FIPS 180-2</td>
<td>optimum</td>
</tr>
<tr>
<td>SHA-384</td>
<td>512 bit</td>
<td>384 bit</td>
<td>FIPS 180-2</td>
<td>optimum</td>
</tr>
<tr>
<td>SHA-512</td>
<td>512 bit</td>
<td>512 bit</td>
<td>FIPS 180-2</td>
<td>optimum</td>
</tr>
</tbody>
</table>

---

**Examples of hash values**

<table>
<thead>
<tr>
<th>Name</th>
<th>String</th>
<th>Hash value (as a hex byte string)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD4</td>
<td>&quot;a&quot;</td>
<td>1f8d010fc013b3c75e55e7539e9506d</td>
</tr>
<tr>
<td></td>
<td>&quot;a&quot;</td>
<td>8f5331e4e4e4e55e55e55e55e55e55e5</td>
</tr>
<tr>
<td></td>
<td>&quot;abcde&quot;</td>
<td>9e4cf1c5b4a0f2a726678a2177ed2b4c</td>
</tr>
</tbody>
</table>

---

© Diana Berbecaru - Politecnico di Torino (2007-2012)
**Digest length**

- Important to avoid *aliasing* (=collisions):
  - $md_1 = H(m_1)$
  - $md_2 = H(m_2)$
  - if $m_1 \neq m_2$ then we want $md_1 \neq md_2$

- 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).

---

**Digest length**

- A $N$-bits digest algorithm is not secure when more than $2^{\frac{N}{2}}$ digests are generated because the probability to have two messages with the same digest is $P_A \sim 50\%$.

---

**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.

MAC, MIC

- to guarantee the integrity of messages, a code is added to the message: 
  MIC (Message Integrity Code)
- hash functions (MICS) can provide data integrity, but no indication about where is data coming from or who generated the hash output:
  - hash function is public so anybody can generate the hash output
- often integrity is not useful without authentication, thus the code (ensuring both security properties) is named: 
  MAC (Message Authentication Code)

MID

- MACs do not prevent all traffic injections (for example do not prevent replay attacks)
- to avoid replay attacks, a unique identifier can be added to the message: 
  MID (Message IDentifier)

Authentication and Integrity
Authentication functions

- create an authenticator for a particular type of authentication service by involving functions of:
  - Entity name / Message
  - Time Stamp / Sequence Number / Random Value
  - Symmetric secret key / Asymmetric private key
- the sender computes and sends the authenticator as part of / in addition to the regular message
- the receiver compares the received authenticator with the one computed by himself

Authentication by symmetric encryption

- sender sends 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; the authenticator is as long as the encrypted message (which may be problematic for large messages)

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

Message digest sent by the sender is encrypted with a symmetric algorithm using a shared secret key. The receiver decrypts the message digest with the same symmetric algorithm and compares it with the calculated message digest. This ensures the integrity and authenticity of the message.

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

Message digest sent by the sender is encrypted with a symmetric algorithm using a shared secret key. The receiver decrypts the message digest with the same symmetric algorithm and compares it with the calculated message digest. This ensures the integrity and authenticity of the message.
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: possible mistakes (I)

Alice sent data

Bob received data

$kd = Hash(K || M)$

Bob calculates: $Hash(K||M||M')=f(kd, M')=kd'$

---

Keyed-digest: possible mistakes (II)

Alice sent data

Bob received data

$kd = Hash(M || K)$

Bob calculates: $Hash(M' || M || K)=kd$;
Keyed-digest: possible solution (I)

Alice sent data

Bob received data

\[ kd = \text{Hash}(K || M || K) \]

\[ kd'' = f(kd, M'') \]

Bob calculates: \( \text{Hash}(K || M || M'' || K) \neq f(kd, M'') = \text{Hash}(K || M || M'' || K) = kd'' \)

Keyed-digest: possible solution (II)

Alice sent data

Bob received data

\[ kd = \text{Hash}(K || M || K) \]

Finds \( M''' \), so that

\[ IV = f(IV, M''') \]

Bob calculates: \( \text{Hash}(K || M''' || M || K) \neq \text{Hash}(K || M || K) = kd \)

Keyed-digest: some standards

- **RFC-1828 (historic) = keyed-md5**
  - \( \text{md5}(K || \text{keyfill} || \text{data} || K || \text{MD5fill}) \)
- **RFC-1852 (obsolete) = keyed-sha1**
  - \( \text{sha1}(K || \text{keyfill} || \text{datagram} || K || \text{SHAfill}) \)
- **RFC-2841 = keyed-sha1 (revised)**
  - \( \text{sha1}(K || \text{keyfill} || \text{data} || \text{datafill} || K || \text{sha1fill}) \)
Authentication by MAC

MAC is a function of the message M and the secret key shared between the sender and the receiver. The output of MAC cannot be produced without knowing the secret key K. The sender computes $mac_1 = MAC(M, K)$ and sends it along with the message M. The receiver computes $mac_2 = MAC(M, K)$ and checks if $mac_1 = mac_2$? Yes: accept the message, no: reject. Because the $mac_1$ could have been generated only by someone that knew the secret key K, this mechanism provides also data source authentication (besides integrity).

Authentication by MAC – Properties (1)

- MAC should be a fixed-size code that is appended to the message;
  - typical sizes of MAC range from 64 to 256 bits
- messages M can be sent in clear without encryption
- MAC is
  - a function of the message and a secret key
  - should not be reversible
  - its strength depends on the function and the secrecy of the key
Keyed digest: HMAC

- HMAC is a scheme to create a MAC of a message M using a hash function H() and a key K
  - \( \text{HMAC}_K(M) = H((K^* \oplus \text{opad}) || H(K^* \oplus \text{ipad}) || M)) \)

where:

- \( K^* = \) secret K padded with zeros to the left, up to the block size of underlying hash function (e.g. MD5 or SHA1)
- \( \text{ipad} = 00110110 \) repeated to fill block (of underlying hash function)
- \( \text{opad} = 01011010 \) repeated to fill block (of underlying hash function)
- HMAC is specified in RFC 2104

Authentication by HMAC

- the size of the output of HMAC is the same as that of the underlying hash function (128 or 160 bits in the case of MD5 or SHA-1, respectively), although it can be truncated if desired.

Authentication by digest and asymmetric cryptography

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

- DIGITAL SIGNATURE !!!
Digital signature

Sender

Message digest

Asymmetric algorithm

Symmetric algorithm

Receiver

Message digest

Key of sender

Asymmetric algorithm

Asymmetric algorithm

Digital signature

Signature and verification

Signer

Data

Digest

Asymmetric encryption

Verifier

Data

Signature

Digest

Asymmetric decryption

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 signature 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=Marco Rossi
  Email=rossi@polito.it
- subject public key info: 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

Simple PKI organization: CA hierarchy

- a CA can certify another CA or the “end entities”
- the certificates are organized in a tree of End Entities (EE) & Certification Authorities (CAs)
Example of CA hierarchy:
EuroPKI hierarchy
- EuroPKI Austria
- EuroPKI Slovenia
- EuroPKI Italy
- Politecnico di Torino CA
- CSP CA
- Comune di Modena CA

Certification hierarchy
- CA EU signs ds (EU)
- CA IT signs ds (EU)
- CA To ds (IT)
- CA Mi ds (IT)

Using certificates
- Root CA
- CA
- EE
- Alice
- Bob
- digitally signed message
Verification of a signature / certificate

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

- KPUB(Bob)

Verification of a certificate

- how to verify that a public-key certificate, for example Bob's 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 ... up to a trusted point (root)

Certificate revocation

- any certificate can be revoked before its expiration date:
  - on request from the owner (subject)
  - autonomously by the creator (issuer)
- when a message is received, the receiver must verify that the certificate is still valid
- the verification 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

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

Architecture of OCSP

- possible pre-calculated response – reduces the load on the server … but makes possible replay attacks!
- possible to get information from other sources (not from CRLs)
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.
  - Specific-purpose accelerators (e.g., SSL, IPsec) or generic ones.

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>rc4-128</td>
<td>61.2 MB/s</td>
<td>62.0 MB/s</td>
</tr>
<tr>
<td>rsa 1024</td>
<td>133.7 signs/s</td>
<td>2472.1 verifies/s</td>
</tr>
</tbody>
</table>
Performance (P3 @ 800 MHz)

<table>
<thead>
<tr>
<th>Method</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>desede3</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>rc4-128</td>
<td>80.9 MB/s</td>
<td>86.4 MB/s</td>
</tr>
</tbody>
</table>

rsa: 1024 signs/s, 1682.0 verifies/s

Length of keys (and of digest)

- equivalence defined in NIST SP800-57
- FFC = Finite Field Cryptography (e.g. DSA, D-H)
- IFC = Integer Factorization Cryptography (e.g. RSA)
- ECC = Elliptic Curve Cryptosystem

<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; 2030</td>
</tr>
<tr>
<td>256</td>
<td>15360</td>
<td>15360</td>
<td>512</td>
<td>512</td>
<td>&gt; 2030</td>
</tr>
</tbody>
</table>

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)
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 be used
- **September 1998:**
  - Permission extended to insurance and health institutions
  - No permission for keys up to 56 bits

USA cryptographic export regulations (December 1999)

- Symmetric algorithms with 56 bits keys
- Asymmetric algorithms with keys:
  - 1024 bits if used only for authentication
  - 512 bits if used also for key exchange
- Not all products conformed to these rules:
  - Netscape has 56 bits keys from version 4.6
  - IE 5.0 has 56 bits keys only in the Win2k version
  - Both generate 512 bits asymmetric keys

Novelty in the 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

Check out: http://recht.vw.fra/koops/cryptolaw/CLS2.HTM for a detailed Crypto Law Survey (dated July 2008), country by country, including the EU regulations
Bibliography

(http://security.polito.it/books.html)

- B. Schneier: “Applied cryptography”
- W. Stallings: “Cryptography and network security” (3rd ed.)
- S. Garfinkel, G. Spafford: “Practical Unix and Internet security”
- W. R. Cheswick, S. M. Bellovin: “Firewalls and Internet security” (2nd ed.)
- W. Ford, M. S. Baum: “Secure electronic commerce”
- C. P. Pfleeger, S. Pfleeger: “Security in computing” (3rd ed.)