Università della Svizzera Italiana

Information Security

AA 2021/2022

InfoSec Summary

Author:
Massimo Parisi

June 13, 2022

Contents

1 Basic Crypto

1.1 InfoSec objectives

The main objectives of information security are:

- **Confidentiality**: allow only authorized access to information
- **Integrity**: No unauthorized alteration of data
- **Availability**: information is available to authorized users at all times
- **Authenticity**:
  - *Data origin authentication*: messages originates from claimed source
  - *Entity authentication*: identification of current source and/or destination
- **Non-repudiation**: prevents an entity from denying previous commitment or actions (e.g. using a digital signature)

1.2 Terminology

- **Threat**: a potential for violation of security, possible action taken against you
- **Attack**: an assault on system security, a deliberate attempt to exploit one or more vulnerabilities
- **Vulnerability**: specific weakness in (or lack of) security services; a way in which threat can be realized
- **Cryptography**: study of mathematical techniques to enforce security properties
- **Cryptanalysis**: study of how to break cryptographic systems
- **Diffusion**: dissipate plaintext structure into long-range stats in ciphertext (each plaintext bit should affect many ciphertext bits). Its goal is to achieve *Avalanche effect* (switch 1 bit $\rightarrow$ change at least half ciphertext)

- **Confusion**: Ensure complex ciphertext—key relationship (each key bit should affect many ciphertext bits). Its goal is to achieve *Completeness* (each ciphertext-bit depends on every key-bit)

- **Kerckhoffs‘ Principle**: the security of the encryption scheme must depend only on the secrecy of the key K_e, and not on the secrecy of the algorithms.

1.3 Historic Ciphers

- **Caesar**: a shift cipher in which each character get shifted by a fixed quantity: $c_i = m_i + 3(\text{mod}26)$ (only 26 possible encryption)

- **Improved Caesar**: remove all non-alphabet characters, use random order of letters (*monoalphabetic* cipher). The key space is much bigger than the previous version ($\approx 2^{88}$) and a simple bruteforce will not work. If we know the language of the message, we can use frequency analysis

- **Playfair**: encode digrams with 5×5 matrix, insert X in between repeating letters. If letters form square, take opposite corners (use letter on same row). If letters are on one row, shift right; if letters are in one column, shift down (wrap around if needed). We can still use frequency analysis to learn some information

- **Vigenère**: different shift for each position, based on a secret keyword. Is it possible to exploit repeating segments of ciphertext (usually digrams/trigrams) and learn the length of the key. If the key length is known, we can try to break multiple monoalphabetic ciphers (using frequency analysis). One possible improvement is *autokeying*, so after the key we continue to encrypt with the plaintext as key. This can still be broken with frequency analysis (English text encrypted with English text)

- **Vernam (OTP)**: it is basically Vigenère with a random key that as the same length of the plaintext. OTP has been mathematically shown to be unbreakable (if you never EVER repeat the key). Nonetheless, it has serious limitation (we should have truly random keys without reuse, really big keys, key distribution problem)

2 Feistel Ciphers

An ideal (block) cipher should use ideal keys that can map any plain text block to any cipher block, but these kind of keys would need $2^n \cdot n$ bits, and that is impractical. For this reason we started using the so-called **Feistel Networks**, used in order to build an <u>invertible</u> random permutation $E : P \leftrightarrow C$ using a <u>non-invertible</u> random function f as a building block. The basic building block of a Feistel cipher are permutation, substitution and XOR. Given a list of round keys $[K_0, ..., K_n]$, a Feistel cipher usually perform the following steps:

- Split plaintext block P into two equal halves L_0 and R_0
- For each round i, it alternately
  - Apply non-invertible function F to one half (using K_i)
  - XOR the result onto the other half

  (e.g. $L_{i+1} = L_i$ and $R_{i+1} = R_i \oplus F(K_i, L_i)$)
- After n rounds, concatenate L_n and R_n to obtain the ciphertext C

Feistel ciphers presents different advantages, like the fact that encryption and decryption use the same algorithm: in order to decrypt, we use the ciphertext as input and reverse the order of the keys.

2.1 DES

Data Encryption Standard, or DES, is one of the most famous Feistel cipher. It consists of the following steps:

1. Initial permutation (IP)

   Note: unless we are using 3-DES, this step does not help
2. 16 Feistel rounds
3. Swap and Inverse permutation (IP^{-1})

In the Feistel rounds we use the *Mangler* function as F, that consist of:

1. Expansion ($32 \rightarrow 48$ bits)

2. XOR with K_i
3. <u>Non-linear</u> Substitution (with S-boxes)

   Note: S-boxes design is unclear (slight changes can affect security)
4. Permutation (with P-boxes)

2.1.1 Multi-DES

Simple DES has been cracked (bruteforce), so stronger version of DES has been created:

- 2DES: encrypt twice (with 2 different keys)

  Note: it can be subject to a *"Meet in the middle"* attack
- 3DES: encrypt three times (with 2 or 3 different keys)

3 AES

Advance Encryption Standard, or AES, is a non-Feistel cipher.

3.1 General info

- Block size: 128 bits
- Key size: 128/192/256 bits
- Rounds: 10/12/14

3.2 Number Theory

[skip modular addition, multiplication,etc.]

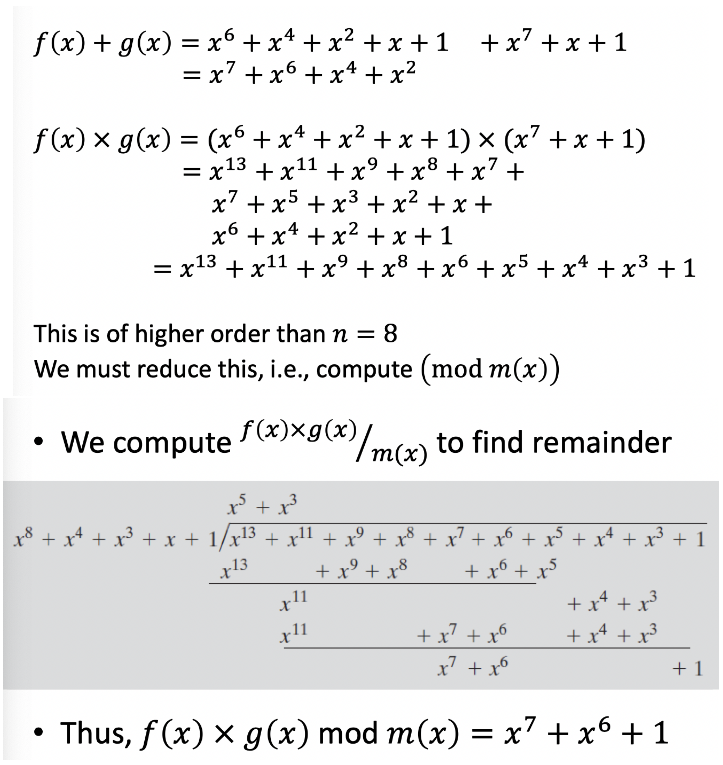
3.3 Galois Fields

For a given prime p, we can define the *finite field of order* p, $GF(p)$, as the set Z_p of integers $0, ..., p-1$ together with the arithmetic operations $(\text{mod } p)$. In order to not waste bit and "fit" directly a finite field into our bytes, we need $GF(2^n)$, but they works differently when using operations.

3.3.1 Polynomial Arithmetic

$GF(p^n)$ means that

- we want coefficients in $GF(p)$ ($p = 2 \rightarrow$ coefficients can only be 0 or 1)
- the result of each operations as to be reduced using $(\text{mod } m(x))$ where $m(x)$ is some irreducible polynomial with order n



$$f(x) + g(x) = x^6 + x^4 + x^2 + x + 1 \quad + x^7 + x + 1$$
$$= x^7 + x^6 + x^4 + x^2$$

$$f(x) \times g(x) = (x^6 + x^4 + x^2 + x + 1) \times (x^7 + x + 1)$$
$$= x^{13} + x^{11} + x^9 + x^8 + x^7 +$$
$$x^7 + x^5 + x^3 + x^2 + x +$$
$$x^6 + x^4 + x^2 + x + 1$$
$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

This is of higher order than $n = 8$
We must reduce this, i.e., compute $(\text{mod } m(x))$

- We compute ${}^{f(x) \times g(x)}/_{m(x)}$ to find remainder

$$\begin{array}{r l}
 & x^5 + x^3 \\
x^8 + x^4 + x^3 + x + 1 \,\big) & x^{13} + x^{11} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + 1 \\
 & \underline{x^{13} \phantom{+ x^{11}} + x^9 + x^8 + x^6 + x^5} \\
 & \phantom{x^{13} +} x^{11} + x^4 + x^3 \\
 & \phantom{x^{13} +} \underline{x^{11} + x^7 + x^6 + x^4 + x^3} \\
 & \phantom{x^{13} + x^{11} + x^9 + x^8 +} x^7 + x^6 + 1
\end{array}$$

- Thus, $f(x) \times g(x) \bmod m(x) = x^7 + x^6 + 1$

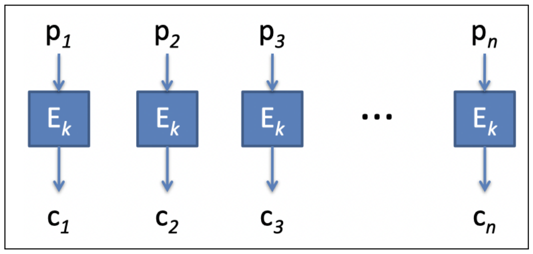
3.4 AES Structure

1. **Substitute bytes**: uses an S-box to perform a b byte substitution of the block
2. **ShiftRows**: A simple permutation
3. **MixColumns**: A substitution that makes use of arithmetic over GF(2^8)
4. **AddRoundKey**: A simple bitwise XOR of the current block with a portion of the expanded key

AES begins with AddRoundKey, followed by 9 rounds (128-bit) with all 4 stages and final round with only 3 stages (skip MixColumns). In order to have the right length for the key, we also perform a **key expansion**.

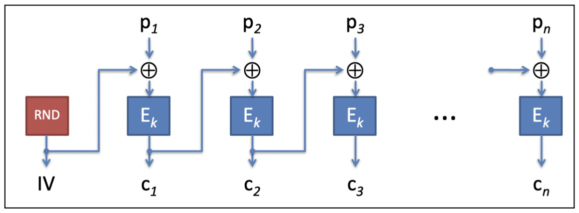
4 Block Modes

4.1 Electronic Codebook - ECB



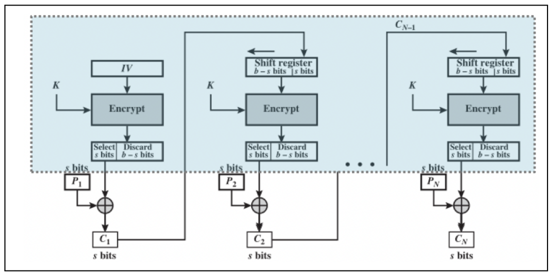
Simplest block mode, just cut the message into $m-$bit blocks and apply the cipher function to each block. The problem is that if we repeat a plaintext block, we obtain the same cipher blocks. A first way to improve ECB would be to XOR each plaintext block with random number, but still we will have problems: now we have twice as much data to transmit and message modification is easy.

4.2 Cipher Block Chaining - CBC



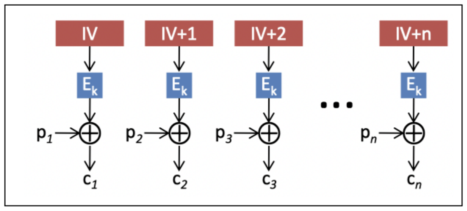
Each block of plaintext is XORed with the preceeding block of ciphertext before encryption. It uses a random *initialisation vector IV* that should never be reused and should stay secret.

4.3 Cipher Feedback - CFB



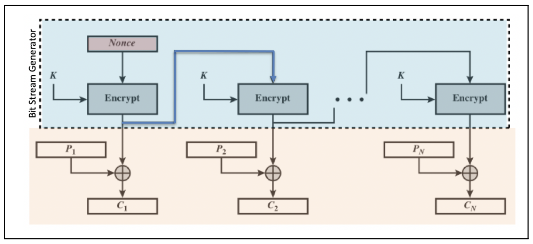
Preceeding ciphertext is used as input to encryption algorithm, to produce pseudorandom output that is XORed with plaintext. Can work on subset of j bits instead of entire blocks.

4.4 Counter Mode - CTR



Each plaintext block is XORed to encrypted counter. Counter gets incremented for each subsequent block.

4.5 Output Feedback - OFB



Similar to CFB, except that instead of ciphertext, it uses preceeding output of encryption function as input to the successive encryption function.

5 Stream Ciphers

A **stream cipher** operates on one plaintext symbol at a time and use generator function to create pseudo-random keystream. Ideally, we should use a *true random number generator* (TRNG), but they are usually slow, for this reason we use *pseudo-random number generator* (PRNG). A PRNG should have *forward and backward unpredictability.* Let's see some famous PRNGs.

5.1 Linear Congruential Generator

The random number sequence $\{X_n\}$ is obtained as $X_{n+1} = (a \cdot X_n + c) (\text{mod } m)$ It goes without saying that in this case the selection of values for (a, c, m) is critical in developing a good PRNG.

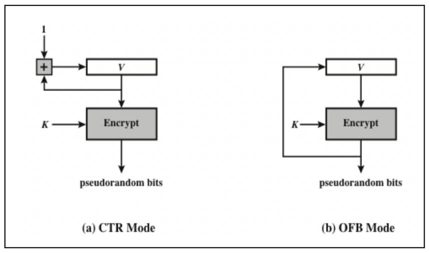
5.2 Blum Blum Shub (BBS) Generator

One of the most commonly used PRNGs. The random number sequence $\{X_n\}$ is obtained as follow

1. Choose two prime (p, q) s.t. $p \equiv q \equiv 3 (\text{mod } 4)$
2. Let $n = p \times q$
3. Choose random s such that $\texttt{gcd}(s, n) = 1$
4. Start from $X_0 = s^2 (\text{mod } n)$
5. If we want n bits, we repeat n times the following
   (a) Compute $X_i = (X_{i-1})^2 (\text{mod } n)$
   (b) Take only the LSB from X_i

5.3 PRNGs using block modes

It is also possible to create a PRNG starting from a CTR or an OFB mode: they just need a key K and an initial vector V_0 (that can be fix, e.g. $V_0 = 0$)

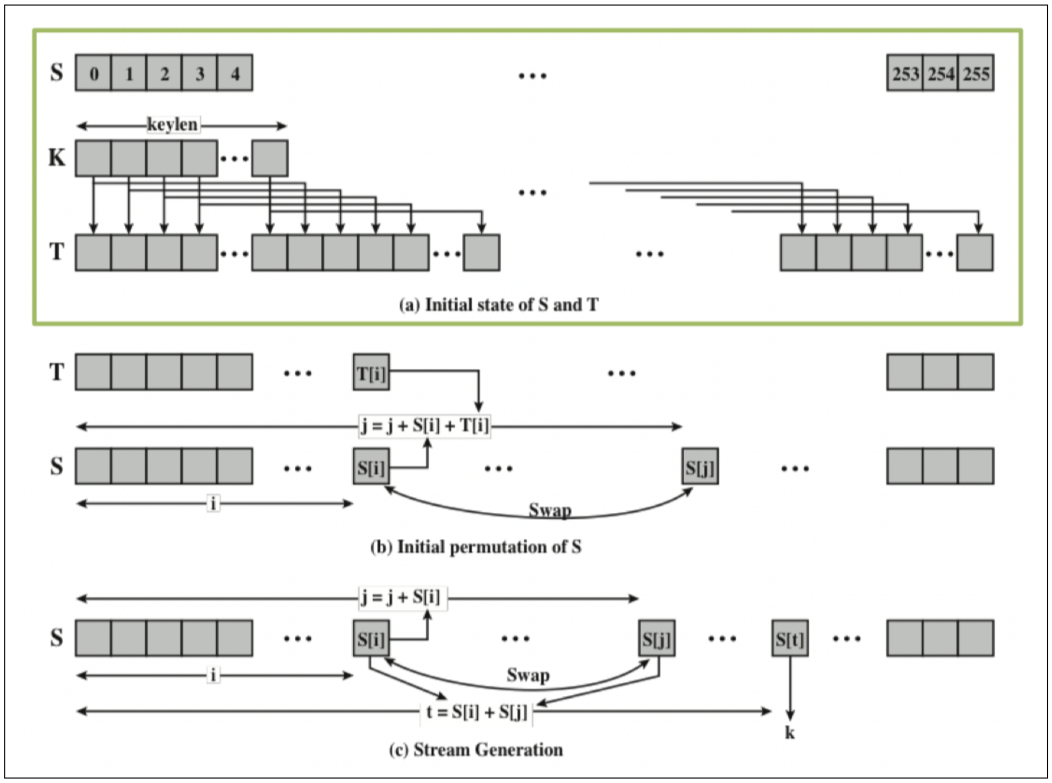


5.4 RC4

RC4 is a variable key size stream cipher. To generate the key stream, the cipher makes use of a secret internal state which consists of two parts:

- A permutation of all 256 possible bytes (S)
- Two 8-bit index-pointers (i and j).

The permutation is initialized with a variable length key using the key-scheduling algorithm (KSA). Then the stream of bits is generated by a pseudo-random generation algorithm.



6 Public Key Crypto

Public-key cryptography (or asymmetric cryptography) is a cryptographic system that uses pairs of keys. Each pair consists of a *public key*, which may be known to others, and a *private key*, which may not be known by anyone except the owner. The generation of such key pairs depends on cryptographic algorithms which are based on mathematical problems termed *one-way functions*.

6.1 Diffie-Hellman-Merkle (DHM) Key Exchange

In order to understand how DHM works, we need to introduce some concepts.

Primitive root A number a is a *primitive root* of prime number p if the numbers $a(\text{mod } p), a^2(\text{mod } p), ..., a^{p-1}(\text{mod } p)$ generate all the integers from 1 to $p-1$ in some permutation (not all integers have primitive roots)

Discrete Logarithm For any integer b and primitive root a of prime p we can find a unique exponent i s.t. $b \equiv a^i(\text{mod } p)$, with $i \in [0, p-1]$

6.1.1 DHM steps

In order to obtain a shared secret, Alice and Bob perform the following steps

1. They publicly agree on g (base) and p (prime), where g is primitive root of p.
2. They both pick a secrete value, respectively A and B
3. They send each other respectively $g^A(\text{mod } p)$ and $g^B(\text{mod } p)$
4. They both raise the received message to the power of their private key, computing the same value $g^{AB}(\text{mod } p)$, that is now their shared secret

The only public information that an attacker has are (g, p), so he needs to find (A, B) to compute the secret, but this is an hard problem to solve (*discrete logarithm problem*). g has to be a primitive root of p, in order to maximize the keyspace.

6.1.2 Man-in-the-Middle Attack

After Alice and Bob choose publicly (g, p) and privately (A, B), Mallory (an attacker) could do the following

1. When Alice and Bob try to communicate, Mallory can intercept their message and answer to them with $g^M (\text{mod } p)$ (where M is a private value of Mallory)
2. Both Alice and Bob end up computing a shared secret with Mallory, respectively $g^{AM} (\text{mod } p)$ and $g^{BM} (\text{mod } p)$, and Mallory does the same
3. Alice and Bob think that they are now sharing a secret, but they are actually communicating with Mallory, that can read every message and send it to the actual receiver

6.2 RSA

While DHM provides only encryption, RSA allow both encryption and authentication:

- if Alice wants to encrypt a message m to make it decryptable only for Bob, she can just encrypt it with Bob's public key
- if Alice wants to send a message m adding her digital signature, she can just encrypt it with her private key

6.2.1 Mathematical background

In order to understand how RSA works, we need to learn some concepts

Totient the totient $\phi(n)$ of a number n is the amount of number coprime with n. If $n = pq$ with (p, q) primes, we know that $\phi(n) = (p - 1)(q - 1)$

Euler's Theorem In general

- $a^b (\text{mod } n) = a^{(b \mod \phi(n))} (\text{mod } n)$
- $b = 1 (\text{mod } \phi(n)) \implies a^b (\text{mod } n) = a$

6.2.2 RSA steps

1. Choose two large primes (p, q)
2. Compute $n = pq$
3. Choose e s.t. $\texttt{gcd}(e, \phi(n))$
4. Find the multiplicative inverse d s.t. $d \cdot e = 1 (\text{mod } \phi(n))$
5. Public key $E = [e, n]$, private key $D = [p, q, d]$
   - **Encrypt**: $c = m^e (\text{mod } n)$
   - **Decrypt**: $m = c^d (\text{mod } n)$
   - **Verify**: $m = s^e (\text{mod } n)$
   - **Sign**: $s = m^d (\text{mod } n)$

6.2.3 PowerMod

PowerMod is a technique to easily compute large powers in modular arithmetic. Given a base g, an exponent e and a modulo p, the steps to follow are the following:

1. Convert e to base 2
2. Initialize $x = g$, start from the leftmost bit of e (skip the first 1)
   - if the current bit is 0, $x = x^2 (\text{mod } p)$
   - if the current bit is 1, $x = (x^2 (\text{mod } p)) \cdot g (\text{mod } p)$

6.2.4 Extended Euclidean Algorithm

We use this algorithm to solve the problem of finding the multiplicative inverse. Its goal is to find (x, y) s.t.

$$ax + by = k = \texttt{gcd}(a, b)$$

This helps us in finding the multiplicative inverse because, if $\texttt{gcd}(a, b) = 1$, we have that $ax + by = 1$. If we set $a = \phi(n) (= \phi)$ and $b = e$, we can say

$$\begin{aligned} [\cancel{(\phi x (\text{mod } \phi))} + (ey (\text{mod } \phi))] (\text{mod } \phi) &\equiv 1 (\text{mod } \phi) \\ ey &\equiv 1 (\text{mod } \phi) \end{aligned}$$

This confirms that $\texttt{gcd}(e, \phi) = 1$ and finds $d = y$

Steps

- Initialization
  - $r_{-1} = a$ — $r_0 = b$
  - $x_{-1} = 1$ — $x_0 = 0$
  - $y_{-1} = 0$ — $y_0 = 1$
- Each step, compute
  - $r_{i+1} = r_{i-1} (\text{mod } r_i)$
  - $-q_{i+1} = \lfloor r_{i-1}/r_i \rfloor$
  - $x_{i+1} = x_{i-1} - q_{i+1}x_i$
  - $-y_{i+1} = y_{i-1} - q_{i+1}y_i$
- Stop and find $d = y_n$ when $r_{n+1} = 0$

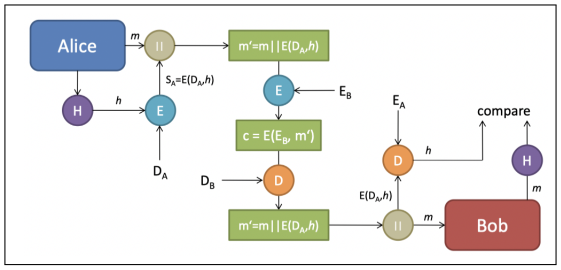
i	r$_i$	q$_i$	x$_i$	Y$_i$
-1	1759		1	0
0	550		0	1
1	109	3	1	-3
2	5	5	-5	16
3	4	21	106	-339
4	1	1	-111	**355**
5	0	4		

Assume we are looking for multiplicative inverse of $e = 550(\text{mod } p = 1759)$. See the result on the table.

7 Hashes

As we have seen in the previous chapters, if we want to guarantee both confidentiality and authentication with public crypto, the sender needs to encrypt the message with his private key (authenticity) and with the receiver public key (confidentiality). To reduce the computational burden we can use hash functions.
A hash function is a function that maps an arbitrary length input into a finite length output of 2^n bits (called *message digest* or *hash*). If Alice wants to send a message to Bob, we can exploit hash functions in the following way:

1. Alice computes the hash h of the message m, encrypts it with her private key, and creates the new message $m' = m||E(D_A, h)$
2. Alice encrypts m' with the public key of Bob, generating $c = E(E_B, m')$
3. Bob receives c, decrypts it with his private key, obtaining m'
4. Bob separates m from $E(D_A, h)$ and decrypts the latter with Alice's public key, obtaining h
5. Bob verify the authenticity of the message simply computing the hash of the message m and comparing it with h



7.1 Hash functions requirements

A good hash function has by definition the following property:

- **One-Wayness**: For any given image h it is computationally infeasible to find pre-image x s.t. $H(x) = h$
- **Weak collision resistance**: For any given pre-image x it is computationally infeasible to find a pre-image $y \neq x$ with $H(x) = H(y)$
- **Strong collision resistance**: it is computationally infeasible to find any pair of pre-images (x, y) s.t. $H(x) = H(y)$

7.2 Birthday Attack

It is clear that we need weak collision resistance property to obtain authenticity, but why do we need the strong collision resistance? The reason is that, without this property, it is possible to perform a birthday attack. A birthday attack is a type of cryptographic attack that exploits the mathematics behind the *birthday problem*.

7.2.1 Birthday Problem

The birthday problem asks for the probability that, in a set of n randomly chosen people, at least two will share a birthday. The birthday paradox is that, counterintuitively, the probability of a shared birthday exceeds 50% in a group of only 23 people. This can be proved using statistics:

- Given n people and k days in a year, we have a probability $P = 1 - \frac{(k-1)!}{k^{n-1}(k-n)!}$ that two of them share the birthday
- with $n = 23$ and $k = 365$, $P = 1 - \frac{364!}{365^{22}(365-23)!} \approx 1 - 0.49 = 0.51$

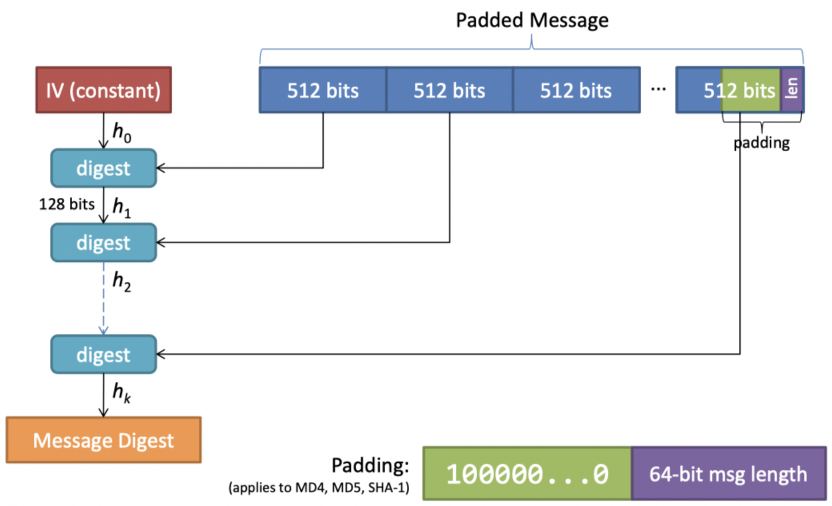
7.2.2 Applications

We can exploit the birthday problem in the following way:

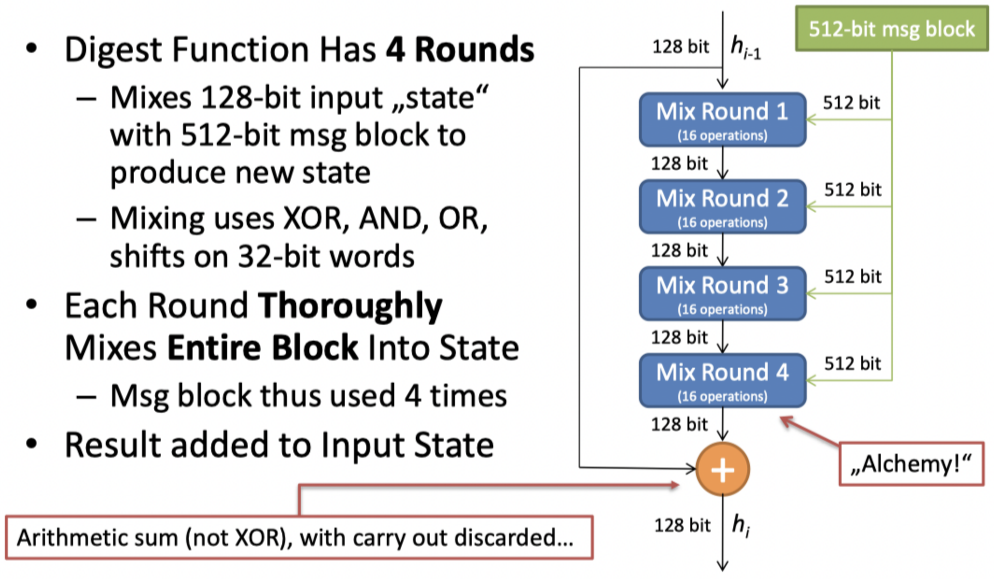
- Prepare $2^{m/2}$ variations of valid and fraudulent messages each
- This should give us a 50% chance of finding collision between a valid and a fraudulent message
- Now we can get someone to sign hash the valid message and then replace it with the fraudulent one

7.3 Hash Algorithms

A generic hash algorithm works in the following way:



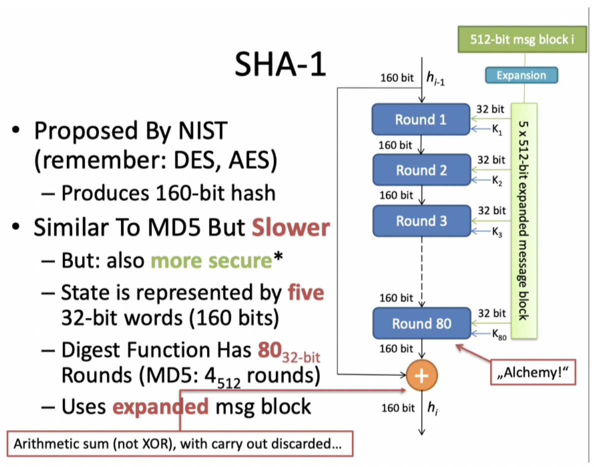
7.3.1 MD5



- Digest Function Has **4 Rounds**
  - Mixes 128-bit input „state“ with 512-bit msg block to produce new state
  - Mixing uses XOR, AND, OR, shifts on 32-bit words
- Each Round **Thoroughly** Mixes **Entire Block** Into State
  - Msg block thus used 4 times
- Result added to Input State

Arithmetic sum (not XOR), with carry out discarded...

7.3.2 SHA1

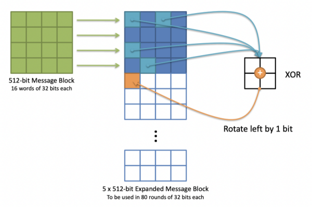


SHA-1

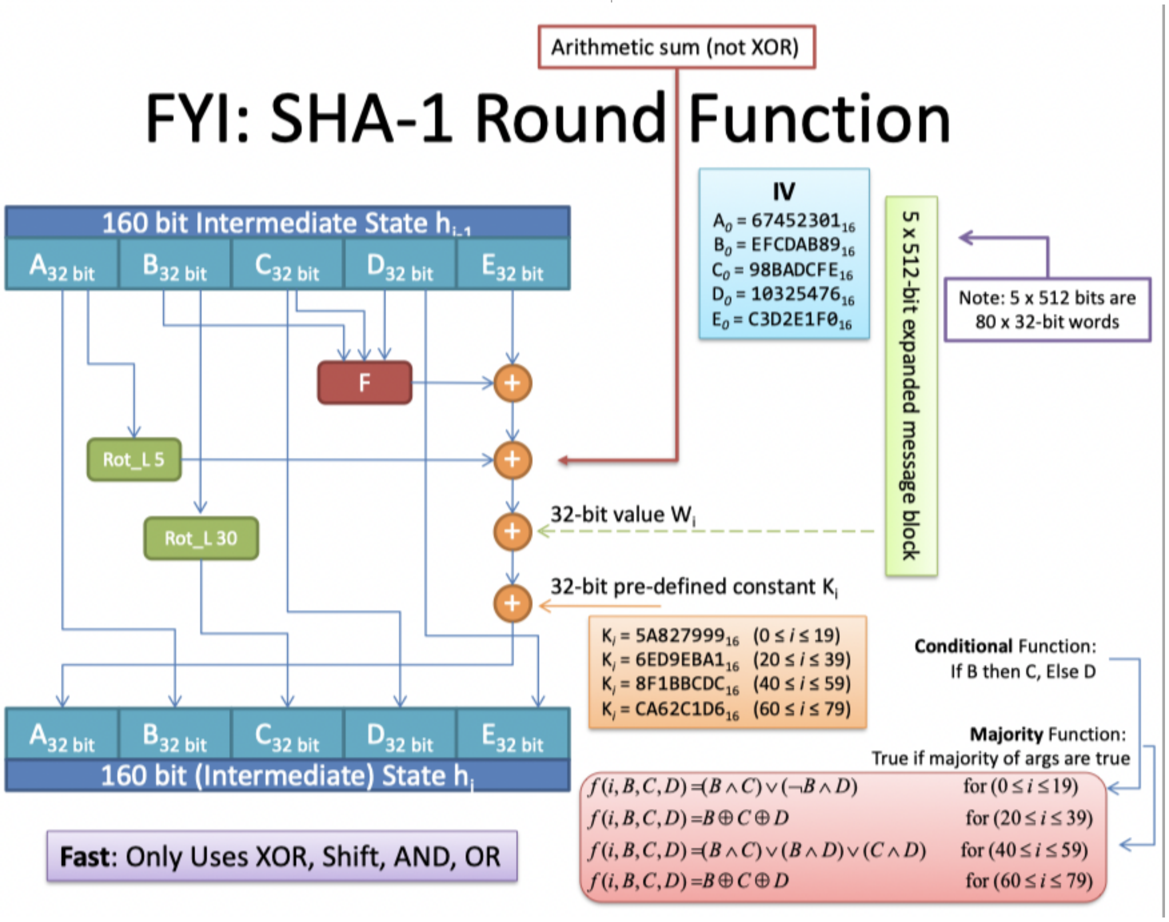
- Proposed By NIST (remember: DES, AES)
  - Produces 160-bit hash
- Similar To MD5 But **Slower**
  - But: also **more secure***
  - State is represented by **five** 32-bit words (160 bits)
  - Digest Function Has **$80_{32\text{-bit}}$** Rounds (MD5: 4_{512} rounds)
  - Uses **expanded** msg block

Arithmetic sum (not XOR), with carry out discarded…

FYI: SHA-1 Expansion Function


512-bit Message Block
16 words of 32 bits each
XOR
Rotate left by 1 bit
5 x 512-bit Expanded Message Block
To be used in 80 rounds of 32 bits each

FYI: SHA-1 Round Function



Arithmetic sum (not XOR)

IV

$A_0 = 67452301_{16}$
$B_0 = EFCDAB89_{16}$
$C_0 = 98BADCFE_{16}$
$D_0 = 10325476_{16}$
$E_0 = C3D2E1F0_{16}$

Note: 5 x 512 bits are 80 x 32-bit words

$K_i = 5A827999_{16}$ $(0 \le i \le 19)$
$K_i = 6ED9EBA1_{16}$ $(20 \le i \le 39)$
$K_i = 8F1BBCDC_{16}$ $(40 \le i \le 59)$
$K_i = CA62C1D6_{16}$ $(60 \le i \le 79)$

Conditional Function: If B then C, Else D

Majority Function: True if majority of args are true

$f(i,B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$ for $(0 \le i \le 19)$
$f(i,B,C,D) = B \oplus C \oplus D$ for $(20 \le i \le 39)$
$f(i,B,C,D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ for $(40 \le i \le 59)$
$f(i,B,C,D) = B \oplus C \oplus D$ for $(60 \le i \le 79)$

Fast: Only Uses XOR, Shift, AND, OR

8 Macs

A Message Authentication Code (MAC) is a short piece of information used for authenticating a message. It uses a shared secret key to provide integrity and authenticity, but differently from symmetric encryption it is not reversible. To do so, we can use hash functions or symmetric block cipher.
We can use hashes as MAC, but we have to be cautious on constructing them: if we use MAC= $MD(K_{AB}||msg)$, someone could perform a *length extension attack*:

- Alice sends "message, MAC" to Bob
- an attacker can construct an extension MAC'= $MD(MAC||msg') = MD(K_{AB}||msg||msg')$

To avoid this, we can construct the MAC differently:

- MAC= $MD(msg||K_{AB})$
- Use only half of MD (not extensible)
- Use MAC= $MD(MD(K_{AB}||msg))$

8.1 HMAC

HMAC is the standard MAC used in IPSec. It can use different hash functions, and what it does is to compute $h(K_{AB} \oplus a||h(K_{AB} \oplus b||m))$, where a and b are two constants. With this structure, even if the attacker found m' such that $m \neq m'$ and $h(m) = h(m')$, he would still be unable to "swap out" m due to the prefixed key and the double hash.

8.2 CBC-MAC

An alternative to HMAC is CBC-MAC, that is simply the last ciphertext block of CBC (with $IV = 0$). CBC-MAC is only secure for fixed length messages, because with variable-length messages with the same key k, an attacker who knows the MACs of two different messages m_1 and m_2 can easily compute the MAC of a third message m_3, even if he does not know the key k.

- Assume the attacker know two messages and their MACs (m_1, t_1) and (m_2, t_2)
- They create $m_3 = m_1 || m_2^*$, where m_2^* is simply m_2 with the first block XORed with t_1
- They know the MAC of m_3 must be t_2

8.3 Digital signatures vs. MAC

We have to remember that there is a clear difference between digital signatures and MACs: a digital signature provides *non-repudiation*, while MAC provides *authenticity*

9 Security Protocols

Now that we have all the security primitives tools, we have to establish how should we use and combine them in order to have security protocols. We will examine three main type of protocols: authentication protocols, key distribution protocols and key establishment protocols.

9.1 How to store sensible data

Let's examine different login procedures, starting from naive approaches, understanding their problems and building up more safe ones.

9.1.1 Password list

A list containing all user password is stored, and if a user try to login with password *pwd*, we search for *pwd*.

9.1.2 Hashed password list

A list of hashed password is stored, if a user try to login with password *pwd*, we compute $hash(pwd)$ and search for it.

Attack 1 Pre-compute a lookup table of hashes for the most common passwords. It grows very large and very quickly

Attack 2 Use a **Rainbow table**:

1. Find hashed password
2. Use reduction function to convert to (pass-)word
3. Try to find that word as an *hash-chain end* in the table
   (a) If found, start from beginning of that hash-chain entry and hash/reduce/hash/reduce/etc, until you find password that hashes to observed hash
   (b) If not found, hash the word and reduce it, then restart at step 2

(N.B. a reduction function does not simply hash inverse, but map back the hash to the original password space W. Usually a rainbow table use different reduction function for each step)

9.1.3 Salted hash

For each password choose a *salt*, that is a random number, and save the salt alongside $hash(salt||pwd)$. Salt increases rainbow-table effort: for each bit of salt we need to double the number of rainbow tables, so with a large salt (e.g. 128 bit) it becomes unfeasible.

Attack Nonetheless, if the attacker gets access to the database the rainbow attacks becomes feasible (he would know the salt, that is saved in clear)

9.1.4 Salted hash + pepper

To avoid the aforementioned problem, we can improve security using *pepper*, another random number that is stored separately from the password database and that is common for all the users.

Attack However, if even only one (`salt + cleartext pwd`) is known, the attacker could try to brute-force the pepper, so it must withstand brute-force attack (e.g. 112 bits)

9.2 Authentication protocols

In this section we will analyze how two entities can authenticates each other, avoiding attacks from malicious entities. We will focus on a family of protocols called *Challenge-Response*, in which two nodes sends each other a value N (the challenge) and expect some answer $f(K, N)$ (the response) from the other. As before, we will see different approaches, from the more naive to the more effective ones, keeping in mind that a good protocols should abide by the **Abadi-Needham** Protocols Design Guidelines:

- **Be explicit** - every message should say what it means and its interpretation should only depend on its content. It's important to know clearly how encryption is used and how the timeliness of messages is proved

- **State your assumptions** - the conditions for a message to be acted upon should be clearly set out so that someone reviewing a design may see whether they are acceptable or not

9.2.1 Mutual Authentication

1. $A \to B: \quad A||N_1$
2. $B \to A: \quad N_2||f(K_{AB}, N_1)$
3. $A \to B: \quad f(K_{AB}, N_2)$

Reflection Attack If the attacker T start this protocol twice, he can authenticate in the following way:

1. $T \to B: \quad A||N_1$
2. $B \to T: \quad N_2||f(K_{AB}, N_1)$
3. $T \to B: \quad A||N_2$
4. $B \to T: \quad N_3||f(K_{AB}, N_2)$
5. `[close]`
6. $T \to B: \quad f(K_{AB}, N_2)$

9.2.2 Mutual Authentication with Trusted Server

Let's suppose that we have a trusted Key Distribution Center, or KDC (we will talk about them later), so that each node needs only one key shared with KDC.

9.2.3 Woo&Lam

1. $A \to B: \quad A$
2. $B \to A: \quad N_B$
3. $A \to B: \quad E(K_{AS}, N_B)$
4. $B \to S: \quad E(K_{BS}, A||E(K_{AS}, N_B))$
5. $S \to B: \quad E(K_{BS}, N_B)$

This authentication does not follow the Abadi-Needham rules.

9.2.4 Fixed Woo&Lam

1. $A \to B: \quad A$
2. $B \to A: \quad N_B$
3. $A \to B: \quad E(K_{AS}, N_B)$
4. $B \to S: \quad \mathbf{A}||E(K_{AS}, N_B)$
5. $S \to B: \quad E(K_{BS}, \boldsymbol{A}||N_B)$

Now the nonce N_B is used only for freshness (and not for association), and there is no double encryption in step 4

9.3 Key distribution protocols

First of all, we want to introduce the concept of session key. A **session key** is a single-use symmetric key used for encrypting all messages in one communication session. There are different advantages on using session keys:

- they limit the amount of ciphertext available
- different keys for authentication & integrity protection
- compromise of long-term secret $\not\Rightarrow$ decryption of old messages

Forward Secrecy

1. $A \to B: \quad \{N_A\}_{E_B}$
2. $B \to A: \quad \{N_B\}_{E_A}$
3. Session key is $K_S = N_A \oplus N_B$

Perfect Forward Secrecy (use signed DHM)

1. `Alice` $\to$ `Bob` : $\{g||p||A\}_{D_A} \qquad (A = g^a (\text{mod } p))$
2. `Bob` $\to$ `Alice` : $\{B\}_{D_B} \qquad (B = g^b (\text{mod } p))$
3. $K_S = g^{ab} (\text{mod } p)$

9.4 Key establishment protocols

9.4.1 Needham-Schroeder

1. $A \to B: \quad A||B||N_1$
2. $S \to A: \quad E(K_{AS}, [N_1, ||B||K_{AB}|| \underbrace{E(K_{BS}, [K_{AB}||A])}_{\texttt{ticket}}])$
3. $A \to B: \quad E(K_{AB}, N_2)||\texttt{ticket}$
4. $B \to A: \quad E(K_{AB}, N_2 - 1||N_3)$
5. $A \to B: \quad E(K_{AB}, N_3 - 1)$

Attack If we use ECB mode, an attacker can impersonate A. Moreover, the attacker could have cracked old session key K_AB.

Timestamp-based alternative

1. $A \to B: \quad A||B$
2. $S \to A: \quad E(K_{AS}, [\mathbf{T}||B||K_{AB}|| \underbrace{E(K_{BS}, [K_{AB}||A||T])}_{\texttt{ticket}}])$
3. $A \to B: \quad E(K_{AB}, N_1)||\texttt{ticket}$
4. $B \to A: \quad E(K_{AB}, N_1 - 1||N_2)$
5. $A \to B: \quad E(K_{AB}, N_2 - 1)$

In this way, is harder for an attacker to reply to message 3. much later

9.4.2 Kerberos Protocol

Kerberos is an authentication protocol, that introduces two new entities:

- **Authentication Server (AS)** - it provides authentication credentials aka *ticket granting ticket* (TGT)
- **Ticket Granting Server (TGS)**: it provides the access to different services based on the TGT received

Overview

1. User logs on to workstation and requests service on host
2. AS verifies user's access right in database, creates TGT and session key. Results are encrypted using key derived from user's password
3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS
4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server
5. Workstation sends ticket and authenticator to server
6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator

Secure Protocol Design Principles?

10 Certificates

There are several techniques that have been proposed for the distribution of public keys. Virtually all these proposals can be grouped into the four general schemes

- **Public announcement**: low overhead, no infrastructure required, but no trust (anyone can announce anything)
- **Public Directory**: there is a trusted entity to manage central service, but it needs to be online all the time, it fails if records get compromised and the registration requires an identity check
- **Public Key Authority**: it automates key exchange process and provides freshness through timestamps, but has the same problems of public directory
- **Public Key Certificates**: distribution does not require central authority

Definition A certificate is a digitally-signed statement that binds a public key to some identifying information. The signer of the certificate is called its *issuer* and the entity talked about in the certificate is the *subject* of the certificate.

In order to believe to a certificates, we need to trust the issuer/signer not to lie to use and we also need to believe that it is hard to forge the digital signature on a signed document

10.1 Strategies for CA Hierarchies

Generally,in order to obtain a certificate

- A user has to ask a *Certification Authority* (CA) that issues certificates and asks for one
- The CA verifies the claim the user is making
- CA then issues the user a certificate for his public key

There are different viable hierarchies model to do so

10.1.1 Monopoly

Choose a universally trusted organization, embed their public key in everything and give them universal monopoly to issue certificates, so that everyone get certificates from them. There are several problems with this model:

- If no one else is allowed to create certs, there is no incentive to offer competitive prices
- Getting certificates for remote user will be insecure or expensive
- The security of the world depends on the honesty and competence of that one organization, forever

Variation Adding *Registration Authorities* (RAs) that are trusted by the CA, so everyone can ask certificates to RAs, who in turn will ask to CA. In this way we solve the distance problem, but not the other issues.

10.1.2 Oligarchy

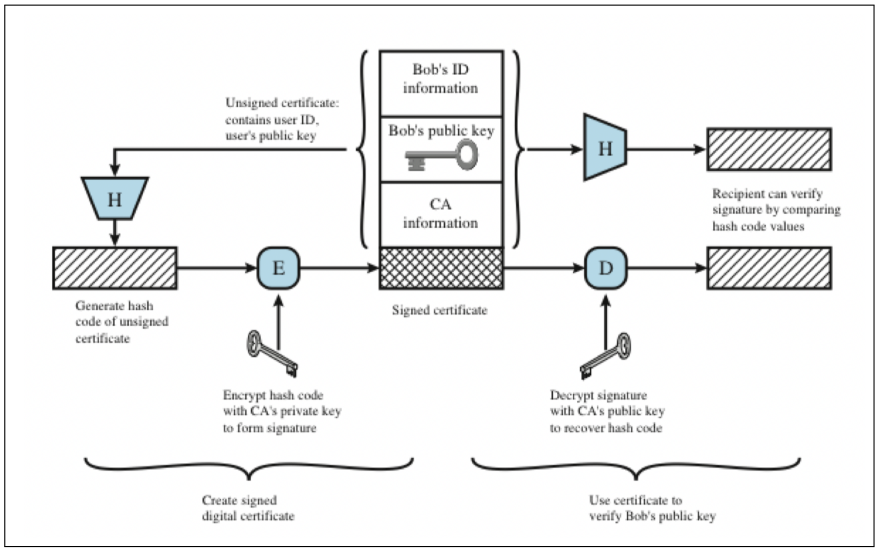
Have multiple trusted CAs.

Variation Allow trusted CAs to issue certificates for other CAs to the list of trusted CAs. This is similar to the monopoly variation, but is less efficient (end user must verify multiple certificates), faster (less delay for RA to get certificate) and less secure (multiple CA means multiple keys to trust).

10.1.3 Anarchy

Anyone signs certificates for anyone else. It is similar to Oligarchy variation, but user must consciously configure starting certificates. The problems are that it is hard to scale (computationally difficult to find path), it is not practical to decide if certain path should be trusted (too many decisions for the end user).

10.2 Certificate Format


Unsigned certificate: contains user ID, user's public key
Bob's ID information
Bob's public key
CA information
H
E
H
D
Signed certificate
Recipient can verify signature by comparing hash code values
Generate hash code of unsigned certificate
Encrypt hash code with CA's private key to form signature
Decrypt signature with CA's public key to recover hash code
Create signed digital certificate
Use certificate to verify Bob's public key

The most common certificate format is **X.509**, containing a lot of information among which: serial number, issuer & subject name, period of validity, signature.

10.3 Certificate Lifecycle Management

Lifecycle steps: enrollement, renewal & revocation.

10.3.1 Enrollement

Enrollment is the process of obtaining a certificate from a CA.

1. Alice generates a key pair, creates a message containing a copy of the public key and her identifying information, and signs the message with the private key (PKCS#10).

   Note: Signing the message provided "proof-of-possession" (POP) of the private key as well as message integrity

2. CA verifies Alice's signature on the message

3. (Optional) CA verifies Alice's ID through out-of-band means.

4. CA creates a certificate containing the ID and public key, and signs it with the CA's own key

   Note: CA has certified the binding between key and ID

5. Alice verifies the key, ID & CA signature

6. Alice and/or the CA publish the certificate

10.3.2 Revocation

Each certificate includes a period of validity, but it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:

- The user's private key is assumed to be compromised
- The user is no longer certified by this CA
- The CA's certificate is assumed to be compromised

Each CA must maintain a *Certification Revocation List* (CRL) consisting of all revoked but not expired certificates issued by that CA. Relying parties are expected to check CRLs before they rely on a certificate. Nevertheless, that are several problems with CRLs:

- Not issued frequently enough to be effective against a serious attack
- Expensive to distribute (size & bandwidth)
- Vulnerable to simple DOS attacks
- Can contain retroactive invalidity dates
- You can't revoke a CRL

For this reason, we use the *Online Certs Status Protocol* (OCSP), that is designed for real-time responses to queries about the status of a single certificate (like a "selective" CRL).

11 TLS

Transport Layer Security (TLS), the successor of the now-deprecated *Secure Sockets Layer* (SSL), is a cryptographic protocol designed to provide communications security over a computer network. Here is how it works:

1. $A \to B$: *I want to talk* (m_1, aka "client_hello")
2. $B \to A$: Certificate (m_2, aka "server_hello")
3. $A \to B$: (m_3 , aka "client key exchange" & "finished")
4. $B \to A$: (m_4, "finished")

TLS is composed by two layers of protocols:

1. Record Protocol
2. Handshake, Change Cipher Spec, Alert, Heartbeat Protocols

In order to understand how TLS operates, we need to learn two architecture concepts:

- **TLS connection**: connections are transient, every connection is associated with one session (one session can have multiple connections). The connection state parameters are: server and client random, server write key, client write key, server write MAC secret, client write MAC secret, IV (CBC mode), sequence numbers.
- **TLS session**: an association between a client and a server, created by the Handshake Protocol. Define a set of cryptographic security parameters which can be shared among multiple connections (used to avoid the expensive negotiation of new security parameters for each connection). The session state parameters are: session identifier, peer certificate, compression method, cipher spec, master secret, is_resumable

11.1 Record Protocol

The TLS Record Protocol provides two services for TLS connections

- Confidentiality: the Handshake Protocol defines a shared secret key that is used for conventional encryption of TLS payloads

- Message integrity: the Handshake Protocol also defines a shared secret key that is used to form a message authentication code (MAC)

Client and server each maintain a set of keys for sending and receiving (plus potentially an IV)

11.2 Heartbeat Protocol

The Heartbeat Protocol is a periodic signal generated by hardware or software to indicate normal operation or to synchronize other parts of a system (typically used to monitor the availability of a protocol entity). It consists of two message types, `hb_request` and `hb_response`. The heartbeat serves two purposes

- It assures the sender that the recipient is still alive
- The heartbeat generates activity across the connection during idle periods, which avoids closure by a firewall that does not tolerate idle connections

11.3 HTTPS

HTTPS refers to the combination of HTTP and SSL to implement secure communication between Web browser and server. When HTTPS is used, the following elements of the communication are encrypted:

- URL of the requested document
- Contents of the document
- Contents of browser forms
- Cookies sent
- Contents of HTTP header

12 Computer Security

Access Control (AC) is the set of (limited) action a user can do in a system. We use AC to prevent security breaches. The main access rights to data and programs are `read`, `write` and `execute`.

12.1 General AC Principles

- System must first authenticate a user seeking access
- Then, AC reference monitor determines if specific requested access by this user is permitted
- A security administrator maintains an authorization database
- An auditing function monitors and keeps a record of user accesses to system resources (accountability, flaws)

12.2 ACLs vs. Capabilities

The authorization database can be organized in different way:

- Access Control List (ACL), often used in filesystems, contains a list for each `object` containing the right of each `subject` on it
- Capabilities is the dual of ACL. It contains a list for each `subject` containing the right on each `object`

12.3 UNIX Permissions

UNIX used ACLs and for each `object`, it indicated the rights of the *owner*, the *group owner* and the *world*

SUID UNIX also offer the possibility for using **SUID** root, that allow users to run an executable with the filesystem permissions of the executable's owner or group owner. They are often used to allow users on a computer system to run programs with temporarily elevated privileges in order to perform a specific task. The introduction of SUID root led to several security vulnerability.

`sudo` A better solution could be to use `sudo`, allowing selected users to temporarily become root (authenticating with their `pwd`)

12.4 Types of AC Policies

- **Discretionary AC**: security decisions up to object owner, different defaults option possible, no coherent security policy enforceable
- **Mandatory AC**: system-wide security policy enforcement, all objects and subjects have "security labels", access granted iff security labels "match", allows control of information flows

12.4.1 Mandatory Access Policies Policy

BLP Model The Bell-La Padula Model consists of two properties:

- **Read Down**: a `subject`'s must dominate the security level of the `object` being read. More formally, S can read O iff $L_S \geq L_O \cap C_S \supseteq C_O$ and S has discretionary read access to O
- **Write Up**: a `subject`'s clearance must be dominated by the security level of the `object` being written to. More formally, S can write to O iff $L_S \leq L_O \cap C_S \subseteq C_O$ and S has discretionary write access to O

Problem High clearance subjects can never communicate with low clearance subjects, so we need dynamic security levels. BLP protects confidentiality, but not integrity

Biba Model It is a state machine model to ensure integrity with these property:

- **Read Down**: a `object`'s integrity level must dominate the integrity level of the `subject` reading it ($I_O \geq I_S$)
- **Write Up**: a `subject`'s integrity level must dominate the integrity level of the `object` being written to ($I_S \geq I_O$)

13 Web Security

13.1 HTTP Authentication

Basic The browser sends password in the clear (over encrypted connection HTTPS). The server has `.htpasswd` file containing hashed passwords.

Digest Uses nonces, timestamps, MD5 hashes, ”opaque” server data.

In practice, neither method is much used anymore, because they are not optimal for user experience (login process). The most used approach is to use HTTPS + HTML form.

13.2 DNS Spoofing

Domain Name Server (DNS) spoofing is an attack in which altered DNS records are used to redirect online traffic to a fraudulent website that resembles its intended destination. The strategy used is the following:

1. User wants to connect to `www.site.com`
2. Local DNS resolver sends UDP packet to DNS server for IP address

   Note: The first ”matching” reply will be used to cache IP, so if attacker is faster than DNS resolver, his IP gets used
3. The attacker listen to outgoing traffic, identify DNS packets and send ”fake” reply packets that match the query

DNS-over-TLS (DoT)/*-over-HTTP* (DoH) adds confidentiality to DNS.

13.3 Homograph Attacks

Homograph attack is a way a malicious party may deceive computer users about what remote system they are communicating with, by exploiting the fact that many different characters look alike (e.g. latin letter vs. cyrillic letter). These type of attack are usually done to commit *phishing* (obtain sensitive information by disguising as a trustworthy entity). A (partial) solution to this is *Punycode*: in case a internet hostname contains letter of different alphabets, it get displayed in a easily detectable format (this solution does not hold if we use letters of only one alphabet).

13.4 Man-in-the-Middle SSL Attack

Due to a bug on Internet Explorer 7, leaf certificates can sign other leaf certificates, so an attacker can sign any SSL certificate, "proving" to a client that it is using the authentic site.

13.5 XSS - Cross Site Scripting Attack

Inject malicious script code into a (trusted) webpage a user is viewing, so that user's browser executes malicious code (the user will probably assume that the behavior is part of the original site). XSS is possible exploiting JavaScript issues, such as the fact that it can access and change any part of a web page. For this reason JavaScript added the *Same Origin Policy* (SOP): origin needs to match domain, protocol & port in order to read/write on the site. Another powerful way to avoid XSS is NoScript, a browser extension that allow JavaScript on originating server only

13.6 SQL Injection

SQL injection is a code injection technique used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker). SQL injection must exploit a security vulnerability in an application's software (e.g. user input is incorrectly filtered for string literal escape characters embedded in SQL statements).

14 Software Security

14.1 The 7 Sins of Insecure Software

1. Improper input validation / representation
2. API abuse
3. Incorrect use/coding of security constructs
4. Incorrect assumptions re. time / state
5. Improper error handling According to
6. Incorrect assumptions regarding execution environment

14.2 Common Input Validation Errors

- **Buffer Over-Read**: Reading beyond boundaries of allocated memory (disclose secret data in adjacent memory cells)
- **Buffer Overflow**: Writing outside boundaries of allocated memory (crash, corrupt data, execute malicious code)
- **String Termination Error**: Relying on proper string termination (buffer overflow)
- **Format String**: Allowing an attacker to control printf format string (buffer overflow)
- **Integer Overflow**: Not accounting for integer overflow (logic errors, buffer overflow)
- **Command Injection**: Executing commands from untrusted source (execute malicious commands)
- **SQL Injection**: Exec SQL statements using unvalidated user input (execute arbitrary SQL commands)
- **Cross-Site Scripting**: Sending unvalidated data to Web client (execute malicious commands on trusted site)

14.3 Buffer Overflow

Buffer overflow is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations. By sending in data designed to cause a buffer overflow, it is possible to write into areas known to hold executable code and replace it with malicious code. It is possible to fit arbitrary-sized shellcode on the stack: start shellcode after return address and fill the rest with "NOPsled", in this way the return address needs to just point somewhere before shellcode.

Protecting the stack

- Make stack non-executable
- Use *Canary* to detect overflows
- Use separate stack for return address

Heap Corruption Occurs when a program damages the allocator's view of the heap.

Format String Occurs when the submitted data of an input string is evaluated as a command by the application. In this way, the attacker could execute code, read the stack, or cause a segmentation fault in the running application, causing new behaviors that could compromise the security or the stability of the system.

14.4 Incorrect use of Time/State

TOCTTOU *Time-of-check to Time-of-use* is a class of software bugs caused by a race condition involving the checking of the state of a part of a system (such as a security credential) and the use of the results of that check.

14.5 The Saltzer-Schroeder Security Principles

1. **Fail-Safe Defaults**: protection mechanism should deny access by default, and grant access only when explicit permission exists.
2. **Complete Mediation**: mechanism should check every access to every object.

3. **Least Privilege**: mechanism should force every process to operate with the minimum privileges needed to perform task.
4. **Open Design**: protection should not depend on attackers being ignorant of its design to succeed. It may however be based on the attackers' ignorance of specific information such as passwords or cipher keys.
5. **Separation of Privilege**: protection mechanism should grant access based on more than one piece of information.
6. **Economy of Mechanism**: the protection mechanism should have a simple and small design.
7. **Least Common Mechanism**: the protection mechanism should be shared as little as possible among users (e.g., process isolation).
8. **Psychological Acceptability**: the protection mechanism should be easy to use (at least as easy as not using it).