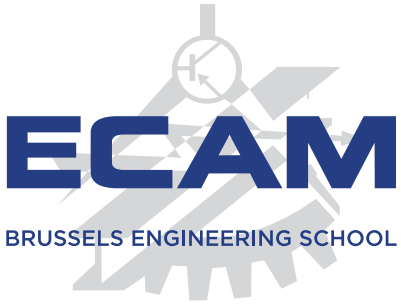

I5020 Computer Security

Session 3

Hash Function, Asymmetric Encryption and Signature

Sébastien Combéfis

Fall 2019

Objectives

- Data integrity with cryptographic hash functions

  Iterated hash functions, SHA-1 and CBC-MAC

- Asymmetric encryption techniques
  - Comparing shared key with public/private key pairs
  - RSA, Rabin and ElGamal cryptosystems

- Digital signature of messages

  Signature schemes and certificate

Cryptographic Hash

Hash Function

- Ensuring data integrity to detect modification

  Construction of a short fingerprint of data (message digest)

- Hash function h generates a fingerprint $y = h(x)$

  The data x is a binary string of any length

- The fingerprint is also a binary string (typically 160 or 256 bits)
  - Storing x and $h(x)$ at different places
  - The fingerprint $h(x)$ must be stored in a secure place

Hash Family (1)

- Hash family with keyed hashed functions h_K

  Often used for message authentication code (MAC)

- Alice and Bob share a secret key K
  - Alice and Bob know the hash function h_K
  - Sending the pair (x, y) with $y = h_K(x)$ on an insecure channel
  - Computing $y = h_K(x)$ to check that x and y has not changed

- Important to use a secure hash family

  No need to securely store the fingerprint that can be transmitted

Hash Family (2)

- Represented by a four-tuple $\langle \mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H} \rangle$
  1. $\mathcal{X}$ set of possible messages
  2. $\mathcal{Y}$ finite set of possible message digests/authentication tags
  3. $\mathcal{K}$ finite set of possible keys (*keyspace*)
  4. $\forall K \in \mathcal{K} : \exists (h_K : \mathcal{X} \to \mathcal{Y}) \in \mathcal{H}$
- If $\mathcal{X}$ is finite, it is referred to as a compression function

  We assume that $|\mathcal{X}| > |\mathcal{Y}|$, *or even* $|\mathcal{X}| \geq 2|\mathcal{Y}|$

Hash Function Security (1)

- A pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$ is valid if and only if $h(x) = y$

  It means that y is a message digest/authentication tag for x

- Three problems hard to solve for a hash function to be good

  - **Preimage**

    Given $h(\cdot)$ and y,
    find x such that $h(x) = y$ $\qquad$ ((x, y) *valid*)

  - **Second preimage**

    Given $h(\cdot)$ and x,
    find x' such that $x' \neq x$ et $h(x') = h(x)$ $\qquad$ ($(x', h(x))$ *valid*)

  - **Collision**

    Given $h(\cdot)$,
    find x, x' such that $x' \neq x$ and $h(x') = h(x)$

Hash Function Security (2)

- The only way to get $h(x)$ for x should be by evaluating h

  Even if we already have several computed hashes $h(x_1)$, $h(x_2)$...

- For example, linear hash function is not secure

  $h : \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_n : (x, y) \mapsto ax + by \mod n$ *($a, b \in \mathbb{Z}_n$ and $n \geq 2$)*

- Knowing $h(x, y)$ at two points allows to calculate others
  - Let $h(x_1, y_1) = z_1$ and $h(x_2, y_2) = z_2$
  - Let $r, s \in \mathbb{Z}_n$, we have that:

$$\begin{aligned} & h(rx_1 + sx_2 \mod n, ry_1 + sy_2 \mod n) \\ = \; & rh(x_1, y_1) + sh(x_2, y_2) \mod n = rz_1 + sz_2 \mod n \end{aligned}$$

Iterated Hash Function

- Iterated hash function to be used with infinite domain

  Compared to compression function for which $\mathcal{X}$ is finite

- Given a compression function $c : \{0,1\}^{m+t} \rightarrow \{0,1\}^m$

  1. **Preprocessing** for x with $|x| \geq m + t + 1$, we construct:
     $y = y_1 \| y_2 \| ... \| y_r$, with $|y| \equiv 0 \pmod{t}$ and $|y_i| = t$ $(1 \leq i \leq r)$
  2. **Processing** with IV, public bistring of length m:
     $z_0 \leftarrow IV$
     $z_1 \leftarrow c(z_0 \| y_1)$
     ...
     $z_r \leftarrow c(z_{r-1} \| y_r)$
  3. **Optional output transformation** $h(x) = g(z_r)$
     with public function $g : \{0,1\}^m \rightarrow \{0,1\}^\ell$

Iterated Hash Function Example

- Given the following parameters and values
  - $IV = 101$ ($m = 3$), $y = 111001$ and $t = 2$
  - $c : \{0,1\}^5 \rightarrow \{0,1\}^3 : \mathbf{x} \mapsto (x_1 \oplus x_3, x_2 \oplus x_4, x_3 \oplus x_5)$
  - $g : \{0,1\}^3 \rightarrow \{0,1\}^3 : \mathbf{x} \mapsto \mathbf{x}$
- Computing the message digest
  1. **Preprocessing** $y = 11\|10\|01$
  2. **Processing**
     $z_0 \leftarrow 101$
     $z_1 \leftarrow c(101\|11) = 010$
     $z_2 \leftarrow c(010\|10) = 000$
     $z_3 \leftarrow c(000\|01) = 001$
  3. **Optional output transformation** $h(001) = 001$

Preprocessing

- Ensuring length of x multiple of t with padding function

  $y = x \| pad(x)$

- The application $x \mapsto y$ must be injective
  - Avoiding collision and finding $x \neq x'$ with $y = y'$
  - Note that it results in $|y| = rt \geq |x|$

Merkle-Damgård Construction (1)

- Security properties with Merkle-Damgård construction

  As long as the compression function is good

- Given that $x \in \mathcal{X}$ are bitstrings

  $$x = x_1 \| x_2 \| ... \| x_k \text{ with } \begin{array}{ll} |x_1| = ... = |x_{k-1}| = t - 1 & \\ |x_k| = t - 1 - d & \text{(where } 0 \leq d \leq t - 2\text{)} \end{array}$$

- Padding function fills on the right with d zeroes

  h will be collision resistant if c also resists to collisions

Merkle-Damgård Construction (2)

- Compression function $c : \{0,1\}^{m+t} \rightarrow \{0,1\}^m$, where $t \geq 2$

Algorithm 1: Merkle-Damgård Construction

Function $MD(x)$

$n \leftarrow |x|$
$k \leftarrow \lceil n/(t-1) \rceil$
$d \leftarrow k(t-1) - n$
for $i \leftarrow 1$ **to** $k-1$ **do**
  $y_i \leftarrow x_i$
$y_k \leftarrow x_k \| 0^d$
$y_{k+1} \leftarrow$ the binary representation of d
$g_1 \leftarrow c(0^{m+1} \| y_1)$
for $i \leftarrow 1$ **to** k **do**
  $g_{i+1} \leftarrow c(g_i \| 1 \| y_{i+1})$
return g_{k+1}

Secure Hash Algorithm (SHA-1) (1)

- Iterated hash function Secure Hash Algorithm (SHA-1)

  160-*bit message digest, operations on* 32-*bit words*

- $|x| \leq 2^{64} - 1$ to limite the binary representation of $|x|$

  Padding function fills with 0 *to reach* 64 *bits*

Algorithm 2: SHA-1 Padding Function

Function *SHA*-1-*PAD*(x)

$d \leftarrow (447 - |x|) \mod 512$

$\ell \leftarrow$ the binary representation of $|x|$ (where $|\ell| = 64$)

$y \leftarrow x \| 1 \| 0^d \| \ell$

return y

Secure Hash Algorithm (SHA-1) (2)

- Cutting the y string in n blocs of 512 bits

  $$y = M_1 \| M_2 \| ... \| M_n$$

- Definition of 80 functions $f_0, ..., f_{79}$

$$f_t(B, C, D) = \begin{cases} (B \wedge C) \vee ((\neg B) \wedge D) & \text{if } 0 \leq t \leq 19 \\ B \oplus C \oplus D & \text{if } 20 \leq t \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{if } 40 \leq t \leq 59 \\ B \oplus C \oplus D & \text{if } 60 \leq t \leq 79 \end{cases}$$

- Definition of 80 constant words $K_0, ..., K_{79}$

$$K_i = \begin{cases} \texttt{5A827999} & \text{if } 0 \leq t \leq 19 \\ \texttt{6ED9EBA1} & \text{if } 20 \leq t \leq 39 \\ \texttt{8F1BBCDC} & \text{if } 40 \leq t \leq 59 \\ \texttt{CA62C1D6} & \text{if } 60 \leq t \leq 79 \end{cases}$$

Secure Hash Algorithm (SHA-1) (3)

Algorithm 3: SHA-1

Function $SHA\text{-}1(x)$

- $y \leftarrow SHA\text{-}1\text{-}PAD(x)$ (M_i a 512-bits block)
- Given $y = M_1 \| M_2 \| ... \| M_n$
- $H \leftarrow [67452301, EFCDAB89, 98BADCFE, 10325476, C3D2E1F0]$
- **for** $i \leftarrow 1$ **to** n **do**
  - Given $M_i = W_0 \| W_1 \| ... \| W_{15}$ (W_i a 32-bit word)
  - **for** $t \leftarrow 16$ **to** 79 **do**
    - $W_t \leftarrow ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$
  - $(A, B, C, D, E) \leftarrow (H_0, H_1, H_2, H_3, H_4)$
  - **for** $t \leftarrow 0$ **to** 79 **do**
    - $temp \leftarrow ROTL^5(A) + f_t(B, C, D) + E + W_t + K_t$
    - $(E, D, C, B, A) \leftarrow (D, C, ROTL^{30}(B), A, temp)$
  - $(H_0, H_1, H_2, H_3, H_4) \leftarrow (H_0 + A, H_1 + B, H_2 + C, H_3 + D, H_4 + E)$
- **return** $H_0 \| H_1 \| H_2 \| H_3 \| H_4$

Message Authentication Code

Message Authentication Code (MAC)

- Incorporating a secret key K as part of the message

  Cannot be included anywhere to avoid attacks

- Hash function h_K with $IV = K$ with $|K| = m$
  - Given a message x and the corresponding MAC $h_K(x)$
  - Given a bitstring x' with length t and the message $x \| x'$
  - The computed tag is $h_K(x \| x') = c(h_K(x) \| x')$
  - Length extension attack allows attacker to find $h_K(x \| x')$...

Nested MAC

- Composition of two (keyed) hash families

  Families $\langle \mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{G} \rangle$ *and* $\langle \mathcal{Y}, \mathcal{Z}, \mathcal{L}, \mathcal{H} \rangle$ *parametrised by keys*

- The composition is a family $\langle \mathcal{X}, \mathcal{Z}, \mathcal{M}, \mathcal{G} \circ \mathcal{H} \rangle$
  - $\mathcal{M} = \mathcal{K} \times \mathcal{L}$ and $\mathcal{G} \circ \mathcal{H} = \{g \circ h : g \in \mathcal{G}, h \in \mathcal{H}\}$
  - where $(g \circ h)_{(K,L)}(x) = h_L(g_K(x))$

- Two conditions to have a secure nested MAC
  - $\langle \mathcal{Y}, \mathcal{Z}, \mathcal{L}, \mathcal{H} \rangle$ is secure as a MAC, given fixed (unknown) key
  - $\langle \mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{G} \rangle$ is collision resistant, given fixed (unknown) key

HMAC

- Construction of a nested MAC from (unkeyed) hash function

  Algorithm proposed in a FIPS standard in March, 2002

- For example, HMAC constructed from SHA-1
  - Using a 512-bit key denoted K, and
  - 512-bit constants: $ipad = \texttt{3636}\ldots\texttt{36}$ and $opad = \texttt{5C5C}\ldots\texttt{5C}$

- Defining a 160-bit MAC as follows

  $$HMAC_K(x) = SHA\text{-}1((K \oplus opad) \| SHA\text{-}1((K \oplus ipad) \| x))$$

CBC-MAC

- Construction of a MAC using a block cipher in CBC mode

  With a fixed (public) initialisation vector

- Initialisation defining $y_0 = IV$, then constructing the y_i with:

  $y_i = e_K(y_{i-1} \oplus x_i)$

Algorithm 4: CBC-MAC

Function *CBC-MAC*(x, K)
- Given $x = x_1 \| ... \| x_n$
- $IV \leftarrow 00...0$
- $y_0 \leftarrow IV$
- **for** $i \leftarrow 1$ **to** n **do**
  - $y_i \leftarrow e_K(y_{i-1} \oplus x_i)$
- **return** y_n

Authenticated Encryption

- Authenticated encryption combines encryption and MAC

  Provide secrecy and data integrity at the same time

- At least three ways to proceed with authenticated encryption
  - **MAC-and-encrypt**: transmit $(e_{K_2}(x), h_{K_1}(x))$
  - **MAC-then-encrypt**: transmit $y = e_{K_2}(x \| h_{K_1}(x))$
  - **encrypt-then-MAC**: transmit $(y, h_{K_1}(y))$ with $y = e_{K_2}(x)$
- Third way to proceed is usually preferred
  - Claimed to be secure if its components are secure
  - Avoid unnecessary decryption if message has been modified

Asymmetric Encryption

Symmetric Encryption

- Using the same secret key K with symmetric encryption

  The key defines the encryption e_K and decryption d_K functions

- Exposure of either e_K or d_K renders the system insecure

  Also, e_K and d_K are typically very close

- Require secure channel between Alice and Bob to exchange K

  Very difficult if they live far away or do not know each other

Asymmetric Encryption

- d_K impossible to find from e_K with asymmetric encryption
  - Public key e_K to encrypt a plaintext
  - Private key d_K to decrypt a ciphertext
- No need for a key exchange on a secure channel

  Only Bob can decrypt a plaintext encrypted with e_K
- Several public-key cryptosystem do exist

  Diffie-Hellman, RSA and ElGamal (and their variants)

Security

- Brute-force attack is possible once knowing e_K et y

  Testing all possible messages x until obtaining $y = e_K(x)$

- Encryption should be a trapdoor one-way function
  - The e_K function must be easily computable
  - Computing the inverse function should be hard (*one-way*)
  - Should be invertible given a secret information (*trapdoor*)

- No injective function proven to be one-way function

  $f(x) = x^b \mod n$ *supposed to be (if n product of primes)*

RSA Cryptosystem

- Computations in $\mathbb{Z}_n$ with $n = pq$ and p, q distinct primes

  Also, we define $\phi(n) = (p-1)(q-1)$

- Description formelle du chiffrement RSA
  - $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$
  - $\mathcal{K} = \{(n, p, q, a, b) \mid ab \equiv 1 \pmod{\phi(n)}\}$
  - $e_{(n,p,q,a,b)}(x) = x^b \mod n$
  - $d_{(n,p,q,a,b)}(y) = y^a \mod n$

- The public key is (n, b) and the private key is (p, q, a)

RSA Cipher Example

- Generating RSA parameters
  - Choosing $p = 101, q = 113$, that is, $n = 11413, \phi(n) = 11200$
  - Choosing $b = 3533$, so $a \equiv b^{-1} \equiv 6597 \pmod{11200}$
  - Public key $(11413, 3533)$ and private key $(101, 113, 6597)$
- Message exchange example:
  - Message: $\mathbf{x} = 9726$
  - Encryption: $\mathbf{y} = 9726^{3533} \mod 11413 = 5761$
  - Decryption: $\mathbf{x} = 5761^{6597} \mod 11413 = 9726$

RSA Properties

- Belief that the $e_K(x)$ function is a one-way function

  Impossible for an opponent to decrypt a ciphertext

- Bob's trapdoor is the factorisation $n = pq$

  Can compute $\phi(n)$ *and then obtain* $a \equiv b^{-1} \pmod{\phi(n)}$

- Breaking RSA is equivalent to factoring n

  It is important that n is large enough to make it difficult

- Encryption/decryption requires a modular exponentiation

  Time complexity of $x^c \mod n$ *in* $\mathcal{O}(\ell k^2)$ *(with k-bit x and* ℓ*-bit c)*

RSA Parameters Generation

- RSA parameters generation before being able to communicate

  Each speaker must generate their own key pair

- Generation algorithm can take time, in practice

  In particular because of the pseudo-random number generator

Algorithm 5: RSA Parameters Generation

Function *Gen-RSA-Params*()

Generate two large prime numbers p, q such that $p \neq q$
$n \leftarrow pq$
$\phi(n) \leftarrow (p-1)(q-1)$
Choose a random b $(1 < b < \phi(n))$ s.t. $gcd(b, \phi(n)) = 1$
$a \leftarrow b^{-1} \mod \phi(n)$
return *public key* (n, b) *and private key* (p, q, a)

Rabin Cryptosystem (1)

- Secure against a chosen-plaintext attack

  Provided that the modulus $n = pq$ cannot be factored

- Formal description of the Rabin cryptosystem
  - $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n^*$
  - $\mathcal{K} = \{(n, p, q) \mid n = pq, p, q \text{ primes and } p, q \equiv 3 \pmod 4\}$
  - $e_{(n,p,q)}(x) = x^2 \mod n$
  - $d_{(n,p,q)}(y) = \sqrt{y} \mod n$

- The public key is n and the private key is (p, q)

Rabin Cryptosystem (2)

- The encryption function e_K is not an injection
  - Decryption cannot be done in an unambiguous fashion
  - There are four square roots of y modulo n
- Bob receives a ciphertext y and want to determine the x s.t.:

  $x^2 \equiv y \pmod{n}$
- Quadratic equation in $\mathbb{Z}_n$ in the unknown x
  - Solving the two congruences $z^2 \equiv y \pmod{p}$
  - and $z^2 \equiv y \pmod{q}$

Rabin Cipher Example

- Given the public key $n = 77$ and the private $(p, q) = (7, 11)$
  - Encryption: $e_K(x) = x^2 \mod 77$
  - Decryption: $d_K(y) = \sqrt{y} \mod 77$
- Message exchange example:
  - Ciphertext: $\mathbf{y} = 23$
  - Square roots of 23 modulo 7 and 11:

    $23^{(7+1)/4} \equiv 2^2 \equiv 4 \pmod 7$ and $23^{(11+1)/4} \equiv 1^3 \pmod{11}$ and so $\pm 10, \pm 32 \pmod{77}$
  - Four possible plaintext: $\mathbf{x} = 10, 32, 45$ and 67

Hybrid Cryptography

- Data is encrypted with symmetric encryption, such as AES

  Suitable and fast enough for "long" messages

- Key is encrypted with asymmetric encryption

  Suitable for very short messages because slow

- Message exchange example steps:

  1. Alice chooses L and computes $y = e_L(x)$
  2. Alice computes $z = e_{K_{Bob}}(L)$ and then transmits (y, z)
  3. Bob computes $L = d_{K_{Bob}}(z)$
  4. Bob computes $x = d_L(y)$

Discrete Logarithm

Discrete Logarithm

- Definition of the discrete logarithm in a group
  - Given a multiplicative group $(G, \cdot)$
  - Given an element $\alpha \in G$ having order n and a $\beta \in \langle \alpha \rangle = \{\alpha^i \mid 0 \leq i \leq n-1\}$ (*cyclic subgroup of G*)
  - Find the unique integer a, with $0 \leq a \leq n-1$ such that $\alpha^a = \beta$ (*will be denoted* $a = \log_\alpha \beta$)
- Used in cryptography because difficult to find discrete log

  But inverse operation of exponentiation efficient

ElGamal Cryptosystem

- Choosing a prime p s.t. discrete log in $(\mathbb{Z}_p^*, \cdot)$ difficult

  Moreover, we take a primitive element $\alpha \in \mathbb{Z}_p^*$

- Formal description of ElGamal cryptosystem
  - $\mathcal{P} = \mathbb{Z}_p^*$ et $\mathcal{C} = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$
  - $\mathcal{K} = \{(p, \alpha, a, \beta) \mid \beta \equiv \alpha^a \pmod{p}\}$
  - $e_{(p,\alpha,a,\beta)}(x, k) = (\alpha^k \mod p, x\beta^k \mod p)$ $\quad (k \in \mathbb{Z}_{p-1}$ secret$)$
  - $d_{(p,\alpha,a,\beta)}(y_1, y_2) = y_2(y_1^a)^{-1} \mod p$

- The public key is (p, α, β) and the private key is a

  And secret random number $k \in \mathbb{Z}_{p-1}$ *chosen at encryption*

ElGamal Cipher Example

- Public key $(p, \alpha, \beta) = (2579, 2, 949)$ and private $a = 765$

  β *is found by solving* $\beta = 2^{765} \mod 2579 = 949$

- Message exchange example:
  - Plaintext: $\mathbf{x} = 1299$ and secret value $k = 853$
  - Ciphertext:

    $(2^{853} \mod 2579, 1299 \cdot 949^{853} \mod 2579) = (435, 2396)$
  - Plaintext: $\mathbf{x} = 2396 \cdot (435^{765})^{-1} \mod 2579 = 1299$

Elliptic Curve (1)

- ElGamal cryptosystem can be implemented in any group

  Provided that the discrete logarithm problem is infeasible

- Non-singular elliptic curves over reals
  - Let $a, b \in \mathbb{R}$ constants such that $4a^3 + 27b^2 \neq 0$
  - Set $\mathcal{E}$ of solutions $(x, y) \in \mathbb{R} \times \mathbb{R}$ of the equation $y^2 = x^3 + ax + b$
  - together with special point $\mathcal{O}$ (*point at infinity*)

Elliptic Curve (2)

- Elliptic curves can also be defined on $\mathbb{Z}_p$ instead of reals

  Replacing all the operations over $\mathbb{R}$ by operations in $\mathbb{Z}_p$

- Elliptic curves over $\mathbb{Z}_p$, with $p > 3$ prime
  - Let $a, b \in \mathbb{Z}_p$ constants such that $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$
  - Set $\mathcal{E}$ of solutions $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ of the equation $y^2 = x^3 + ax + b$
  - together with special point $\mathcal{O}$ (*point at infinity*)


Signature

Signature Scheme (1)

- Digitally sign a document with a signature scheme

  Adding the signature to the message, not "writing on top of it"

- Problem with the verification of a signature

  How is it possible to compare a signature with the "original" one

- Signed document can be used several times

  For example, authorisation for an action (withdraw 100 *euros)*

Signature Scheme (2)

- Composed of two algorithms to sign and verify a signature

  Verification for (x, y) *if* y *valid signature for* x *or not*

- Formal description of signature scheme
  - $\mathcal{P}$ finite set of possible messages
  - $\mathcal{A}$ finite set of possible signatures
  - $\mathcal{K}$ finite set of possible keys (*keyspace*)
  - $\forall K \in \mathcal{K} : \exists (sig_K : \mathcal{P} \to \mathcal{A}) \in \mathcal{S},$ (*signing algorithm*)
  - $(ver_K : \mathcal{P} \times \mathcal{A} \to \{true, false\}) \in \mathcal{V} :$ (*verification algorithm*)

    $$\forall x \in \mathcal{P}, y \in \mathcal{A} : ver_K(x, y) = \begin{cases} true & \text{si } y = sig_K(x) \\ false & \text{si } y \neq sig_K(x) \end{cases}$$

RSA Signature Scheme

- RSA cryptosystem can be used as a signature scheme

  To be used "upside down" to have RSA signature scheme

- Formal definition of RSA signature scheme
  - Given $n = pq$ with p, q primes and $\mathcal{P} = \mathcal{A} = \mathbb{Z}_n$
  - $\mathcal{K} = \{(n, p, q, a, b) \mid ab \equiv 1 \pmod{\phi(n)}\}$
  - $sig_{(n,p,q,a,b)}(x) = x^a \mod n$
  - $ver_{(n,p,q,a,b)}(x, y) = true \iff x \equiv y^b \pmod{n}$

- Verification algorithm is public and signature is private

  Both should be polynomial-time functions

Attack Model

- Possible to forge a signature to build valid $y = sig_K(x)$

  Choosing a random signature y and computes $x = e_K(y)$

- Several existing types of attacks depending on what has Eve
  - **Key-only attack**: Alice's ver_K
  - **Known message attack**: List of signed messages (x_i, y_i)
  - **Chosen message attack**: Alice's *sig* to sign list of messages

- Can always be attacked by brute-force approach

  Testing all possible signatures $y \in \mathcal{A}$

Signature and Hash Function

- Combining signature with secure cryptographic hash function
  1. Message x to be sent is hashed $z = h(x)$
  2. Signature is computed on the hash: $y = sig_K(z)$
  3. Alice send the pair (x, y) to Bob
  4. Bob computes $z = h(x)$ and checks the signature $ver_K(z, y)$
- The hash function must be secure enough for a good process

  Same conditions than in the previous applications

Certificate

- Mechanism to authenticate public keys with certificate

  Require some kind of Public Key Infrastructure (PKI)

- Relies on a trusted certification authority (CA)
  - Signs the public keys of all people in the network
  - Verification key ver_{CA} known "by magic" by everyone

- Signed certificate contains several information

  Name, email, address, list of public keys

Encryption and Signature (1)

- Important to securely combine signing and encryption

  Recommended method is called the "sign-then-encrypt"

- Alice wants to send a signed and encrypted message x to Bob

  1. Alice signs $y = sig_A(x)$ then encrypts $z = e_B(x, y)$
  2. The ciphertext z is transmitted to Bob
  3. Bob decrypts $(x, y) = d_B(z)$ then verifies $ver_A(x, y)$

- Malicious Bob may send message to Carol as Alice

  Decrypts $(x, y) = d_B(z)$, *send* $z' = e_C(x, y)$

Encryption and Signature (2)

- Another approach is the "encrypt-then-sign" process
  1. Alice encrypts $z = e_B(x)$ then signs $y = sig_A(z)$
  2. The pair (z, y) is transmitted to Bob
  3. Bob verifies $ver_A(z, y)$ then decrypts $x = d_B(z)$
- Malicious Eve replace signature with its own

  Intercepts (z, y) *and replace by* $(x, sig_E(z))$

ID Concatenation

- Solution is to concatenate public identification information
  - Before encrypting, concatenate ID for the sender
  - Before signing, concatenate ID for the receiver
- For "sign-then-encrypt" strategy:

  $y = sig_A(x, ID(B))$, $z = e_B(x, y, ID(A))$ *and sends* z
- For "encrypt-then-sign" strategy:

  $z = e_B(x, ID(A))$, $y = sig_A(z, ID(B))$ *and sends* $(z, y, ID(A))$

References

- Douglas R. Stinson, & Maura B. Paterson, *Cryptography: Theory and Practice* (Fourth Edition), CRC Press, 2017. (ISBN: 978-1-138-19701-5)
- Daniel, *Cryptographic Hash Functions Explained: A Beginner?s Guide*, August 14, 2018. https://komodoplatform.com/cryptographic-hash-function
- Tim Fisher, *Cryptographic Hash Function: Use a cryptographic hash function to verify the authenticity of data*, August 11, 2019. https://www.lifewire.com/cryptographic-hash-function-2625832
- ConsenSys, *Are you really using SHA-3 or old code?*, January 12, 2016. https://medium.com/@ConsenSys/are-you-really-using-sha-3-or-old-code-c5df31ad2b0
- Erik Ringsmuth, *Encrypt-then-MAC: How I learned AES encryption does not tamper-proof data*, June 15, 2014. https://medium.com/@ErikRingsmuth/encrypt-then-mac-fc5db94794a4
- Short Tech Stories, *How does RSA work?*, June 23, 2017. https://hackernoon.com/how-does-rsa-work-f44918df914b
- Josh Lake, *What is RSA encryption and how does it work?*, December 10, 2018. https://www.comparitech.com/blog/information-security/rsa-encryption
- Nick Sullivan, *A (relatively easy to understand) primer on elliptic curve cryptography*, October 24, 2013. https://arstechnica.com/information-technology/2013/10/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography
- Credits Blockchain, *Digital signature: Introduction*, June 20, 2018. https://medium.com/@credits/digital-signature-26897c00cf74

Credits

- Icons from https://icons8.com/icons.
- stu_spivack, March 3, 2010, https://www.flickr.com/photos/stuart_spivack/4425612269.
- Avinash Kumar, August 30, 2009, https://www.flickr.com/photos/avifotos/3889188265.
- Dan Backman, June 19, 2010, https://www.flickr.com/photos/dbackmansfo/4716003831.
- Ian Foss, March 18, 2006, https://www.flickr.com/photos/badboy69/2093177318.
- Sebastien Wiertz, April 29, 2010, https://www.flickr.com/photos/wiertz/4563720850.