Public Key Cryptography

Lecture 9

Digital Signatures

Index

Hash functions

Definition

A *hash function* is a function which transforms data of arbitrary length into data of fixed length.

- if $h : D \to R$ and $|D| > |R|$, then there exist *collisions*, that is, the same value in the codomain corresponds to different values in the domain.
- a hash value serves as a representative compact image of a message (*digital fingerprint* or *message digest*) of input data, and may be used as being (uniquely) identifiable with the data
- hash functions are used for data integrity together with digital signature schemes

Properties

In what follows h is a hash function.

- *compression*: h transforms an input x of arbitrary length into an output $h(x)$ of fixed length.
- *easily computable*: for any input x, $h(x)$ is easily computable.
- *avalanche effect*: if the input is slightly changed, then the output is significantly changed.
- *preimage resistance*: given any output y, it is computationally infeasible to find an input x with $h(x) = y$.
- *weak collision resistance*: given an input x, it is computationally infeasible to find an input $x' \neq x$ with $h(x) = h(x')$.
- *strong collision resistance*: it is computationally infeasible to find any two inputs $x \neq x'$ having the same output $h(x) = h(x')$.

Properties: examples

(a) Computing the sum of every string of 32 bits of data is a function which is easily computable and ensures compression, but not preimage resistance.

(b) The function

$$g(x) = x^2 \bmod n$$

ensures preimage resistance, but neither compression nor weak collision resistance.

(c) The function

$$f(x) = E_k(x) \oplus x,$$

where E is some block cipher, $\oplus$ is EXCLUSIVE OR and k is a known fixed key, ensures preimage resistance and weak collision resistance, but not compression.

Examples of hash functions

The most known examples are the families MD and SHA.

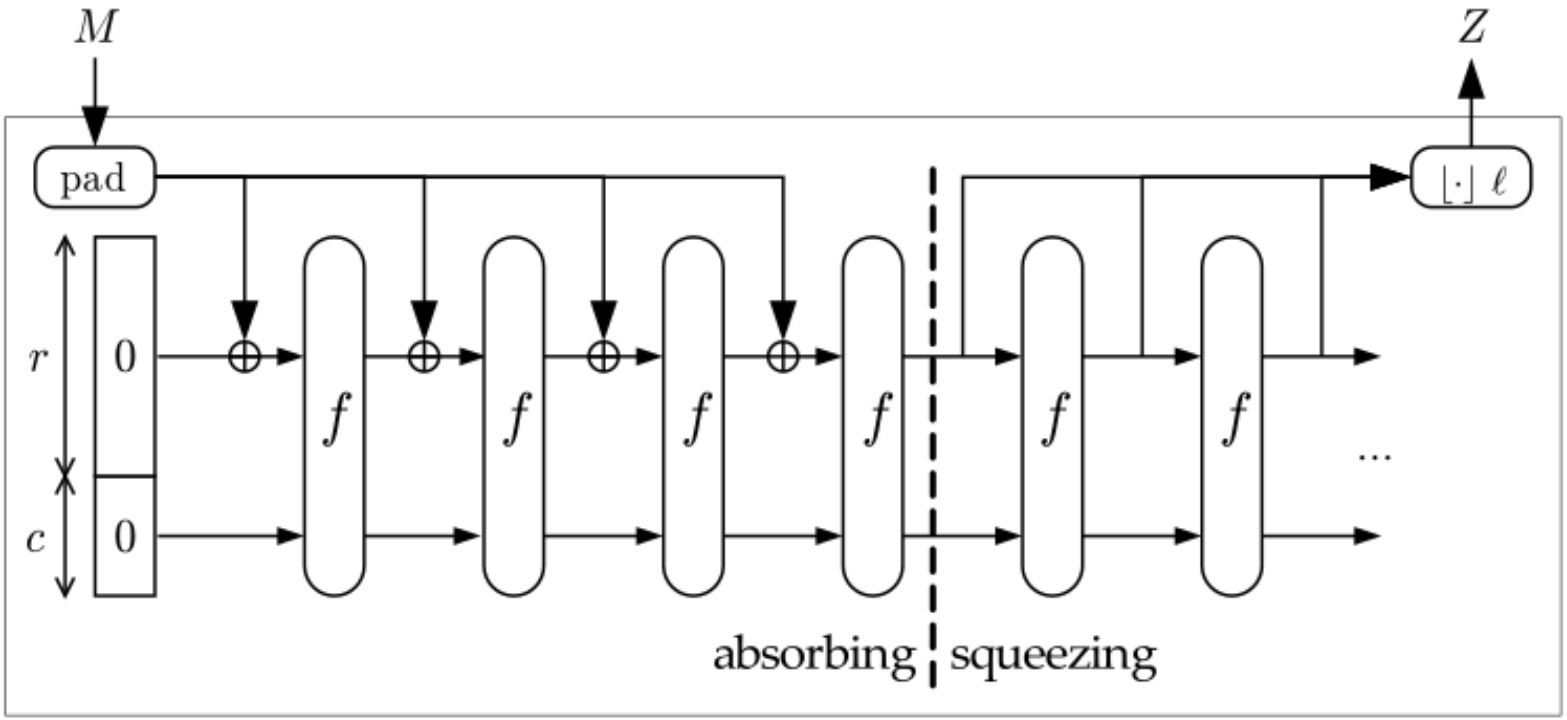
MD family:

- MD5 (1992) has been the most popular
- MD5 computes a hash value of 128 bits
- collisions have been found; not used in cryptography anymore

SHA family:

- SHA-3 has been the present hash standard since din 2015
- SHA-3 computes a hash value of 224, 256, 384 or 512 bits
- Examples of hash values computed with SHA-3 on 512 bits:
  "*statistic*" has hash value:
  a61ac830eeef120ae71cffa34fc95385e7b9f54764ed80a1fec2a12a123f0d3b
  8c54ed24529fb281d36f01b72dbb8cdd343c2041b48e910ddf1631f6a93f9ef2
  "*statistics*" has hash value:
  886829d000ed99f19cd24a7a1d66a4636a0aac755a6f17e44ae08a3078b9a075
  d7d2836f55a1f1428a387e5c298b83cbedcee8d0c46cf5fada03a264715469c2

SHA-3 principles



sponge

Didactic example: ToyHash

ToyHash Algorithm

Given a message m, denote:

- v = number of vowels of m
- c = number of consonants of m
- w = number of words of m

The ToyHash value of m is

$$h(m) = v^2 + c \cdot v + w \pmod{19}.$$

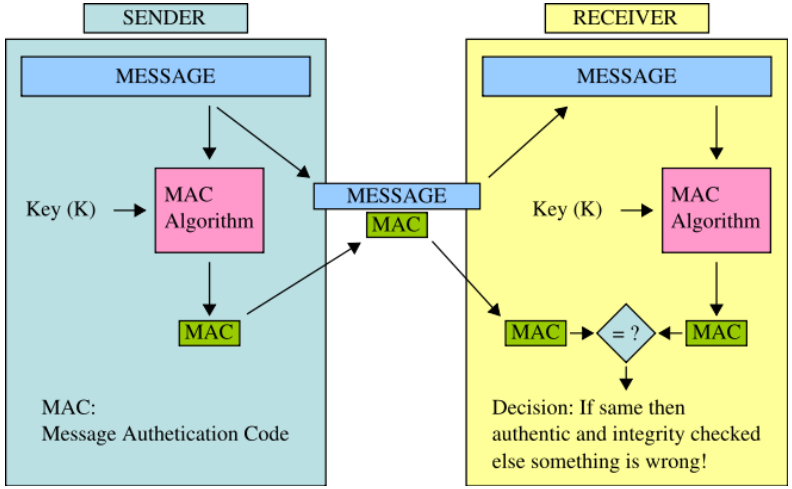
Example. Let us compute the ToyHash value of the messages: $m_1 =$"*Public Key Cryptography*" and $m_2 =$"*Values*".

$$h(m_1) = 8^2 + 8 \cdot 13 + 3 = 0 \pmod{19}.$$

$$h(m_2) = 3^2 + 3 \cdot 3 + 1 = 0 \pmod{19}.$$

Classes of hash functions

- *unkeyed*, having as the single input a message (e.g. MD5, SHA-3). A subclass is MDC (*modification detection codes*), whose purpose is to provide a representative image or hash of a message, satisfying certain additional properties.
- *keyed*, whose specification dictates two distinct inputs: a message and a secret key. A subclass is MAC (*message authentication codes*), whose purpose is to facilitate, without the use of any additional mechanisms, assurances regarding both the source of a message and its integrity.

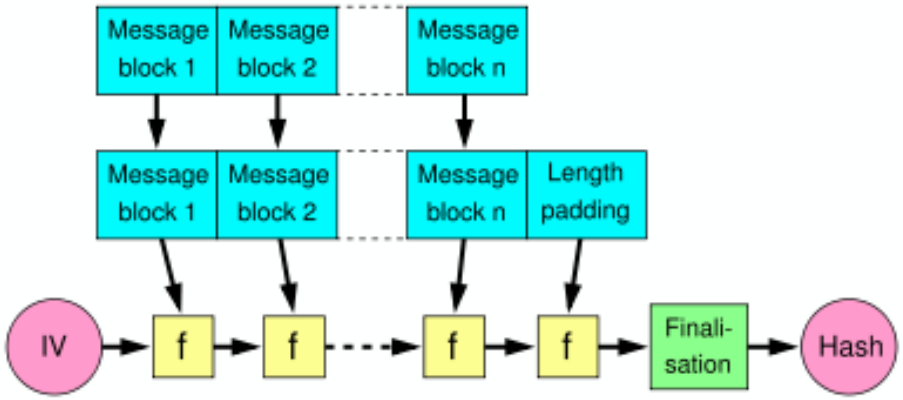


General constructions of hash functions

Extension

Any compression function f which is collision resistant can be extended to a collision resistant hash function h (taking arbitrary length inputs).

Example: *Merkle's meta-method*:



Cascading hash functions

If either h_1 or h_2 is a collision resistant hash function, then $h(x) = h_1(x)||h_2(x)$ (their concatenation) is a collision resistant hash function.

Applications of hash functions

- A typical use: Alice poses a tough math problem to Bob, and claims she has solved it. Bob would like to try it himself, but would yet like to be sure that Alice is not bluffing. Therefore, Alice writes down her solution, appends a random nonce, computes its hash and tells Bob the hash value (whilst keeping the solution and nonce secret). This way, when Bob comes up with the solution himself a few days later, Alice can prove that she had the solution earlier by revealing the nonce to Bob.
- Verification of message integrity: determining whether any changes have been made to a message (or a file), for example, can be accomplished by comparing message digests calculated before, and after, transmission (or any other event).
- A message digest can also serve as a means of reliably identifying a file; several source code management systems use some hash of various types of content (file content, directory trees, ancestry information, etc.) to uniquely identify them.

Applications of hash functions (cont.)

- Password verification: passwords are usually not stored in cleartext, for obvious reasons, but instead in digest form. To authenticate a user, the password presented by the user is hashed and compared with the stored hash. This is sometimes referred to as one-way encryption.
- For both security and performance reasons, most digital signature algorithms specify that only the digest of the message be "signed", not the entire message. Hash functions can also be used in the generation of pseudorandom bits.
- Hashes are used to identify files on peer-to-peer file sharing networks. For example, a hash is combined with the file size, providing sufficient information for locating file sources, downloading the file and verifying its contents.

Digital signatures - a general scheme

Definition

A *digital signature* is a data string which associates a message (in digital form) with some originating entity.
A *digital signature scheme* consists of key generation, signature and verification.

A digital signature is applied to the hash value of a message, because its use on a long message may be computationally very expensive.

Consider an arbitrary public key cryptosystem. Denote:

- U: a user
- f_U: the public enciphering transformation of U
- f_U^{-1}: the secret deciphering transformation of U
- h: a hash function h, which is public for every user

A general scheme - signature

Alice signs and sends a message to Bob

If Alice wants to sign for Bob a message m, then she sends:

$$c = f_B(m||f_A^{-1}(f_B(h(m)))),$$

where $||$ is concatenation and $f_A^{-1}(f_B(h(m)))$ is her signature.

- Alice's signature cannot be obtained by another user, since it depends on f_A^{-1} (known only by Alice).
- The signature depends on the signed message m and is short, because of the fingerprint $h(m)$. So no one can alter m and it is impossible to use this signature for other messages.
- The signature depends on Bob (because of f_B), so Bob cannot send the message m with Alice's signature to a third person. Alice signed m only for Bob.

A general scheme - verification

Bob checks the authenticity of Alice's signature

- Bob gets c and computes (using his secret key) $f_B^{-1}(c)$, which is an English text m' with a nonsense scratch s in the end.
- He checks the equality

$$h(m') = f_B^{-1}(f_A(s)).$$

[We have $f_B^{-1}(f_A(s)) = f_B^{-1}(f_A(f_A^{-1}(f_B(h(m))))) = h(m)$.]

- If this is true, then the signature is authentic. Otherwise, the message or the signature was altered or forged.

Properties of digital signatures

Common properties with the hand-written signature:

- *uniqueness*: the signature should belong to a unique person
- *inimitability*: only the user can generate the signature
- *easy to authentify*: any legal recipient and any referee can verify the authenticity of the signature
- *impossibility of negation*: no legal user can deny his/her own signature
- *easy to generate*

Differences with respect to the hand-written signature:

- The hand-written signature is the same independent of the document. The digital signature depends on the signer and the document.
- A copy of an electronic document (including the signature) is identical with the original. The copy of a document on paper has a different value than the original.

RSA signature scheme

- security: Integer Factorization Problem
- uses a public hash function h
- sends the plaintext m together with the digital signature for the hash value $h(m)$

Key generation. Alice creates a public key and a private key.

1. Generates 2 random large distinct primes p, q of approximately same size.
2. Computes $n = pq$ and $\varphi(n) = (p-1)(q-1)$ (the Euler function).
3. Randomly selects $1 < e < \varphi(n)$ with $gcd(e, \varphi(n)) = 1$.
4. Computes $d = e^{-1} \pmod{\varphi(n)}$.
5. Alice's public key is (n, e); her private key is d.

RSA signature scheme (cont.)

Signature generation and verification.

1. *Signature generation.* Alice signs a message m.
   1. Computes $\tilde{m} = h(m) \in \mathbb{Z}_n$.
   2. Computes $s = \tilde{m}^d \pmod{n}$.
   3. Alice's signature for m is s.
2. *Verification.* Bob verifies Alice's signature and recovers the message m from the signature.
   1. Gets Alice's public key (n, e).
   2. Computes $m' = s^e \pmod{n}$.
   3. Computes $\tilde{m} = h(m)$.
   4. Checks if $m' = \tilde{m}$. If yes, he accepts the signature; if not, he rejects it.

Proof of correctness. We have $s = \tilde{m}^d \pmod{n}$. Since $ed = 1 \pmod{\varphi(n)}$, we have $m' = s^e = (\tilde{m})^{ed} = \tilde{m} \pmod{n}$ (see the proof of correctness of RSA).

RSA signature scheme (cont.)

Example. Use the RSA signature scheme.

- Alice creates a public key and a private key.
- Alice sends the plaintext $m =$ "*Cryptography*" together with the digital signature for $h(m)$, where h is ToyHash.
- Bob verifies Alice's signature.

Key generation. Alice:

1. Selects primes $p = 29$ and $q = 31$.
2. Computes $n = pq = 899$ and $\varphi(n) = (p-1)(q-1) = 840$.
3. Chooses $e = 11$ (note that $gcd(11, 840) = 1$).
4. Computes

$$d = e^{-1} \pmod{\varphi(n)} = 11^{-1} \pmod{840} = \ldots = 611.$$

5. Alice's public key is $(n, e) = (899, 11)$.

Alice's private key is $d = 611$.

RSA signature scheme (cont.)

1. *Signature generation.* Alice:
   1. Computes $\tilde{m} = h(m) = 4^2 + 4 \cdot 8 + 1 = 11$.
   2. Computes

$$s = \tilde{m}^d \pmod{n} = 11^{611} \pmod{899} = \ldots = 520.$$

   3. Alice's signature for m is $s = 520$.
2. *Verification.* Bob:
   1. Gets Alice's public key $(n, e) = (899, 11)$.
   2. Computes

$$m' = s^e \pmod{n} = 520^{11} \pmod{899} = \ldots = 11.$$

   3. Computes $\tilde{m} = h(m) = 11$.
   4. Accepts the signature because $m' = \tilde{m}$.

Rabin signature scheme

- security: Modular Square Root Problem
- uses a public hash function h
- sends the plaintext m together with the digital signature for the hash value $h(m||U)$, where $m||U$ denotes concatenation and U is a random padding of the plaintext m

Key generation. Alice creates a public key and a private key.

1. Generates 2 random large distinct primes p, q of approximately same size.
2. Computes $n = pq$.
3. Alice's public key is n; her private key is (p, q).

Rabin signature scheme (cont.)

1. *Signature generation.* Alice:
   1. Chooses a random padding U until $\tilde{m} = h(m||U)$ is a quadratic residue modulo n.
   2. Solves $x^2 = \tilde{m} \pmod{n}$.
   3. Alice's signature for m is (U, a), where a is one of the square roots modulo n.
2. *Verification.* Bob:
   1. Gets Alice's public key n.
   2. Computes $m' = a^2 \bmod n$.
   3. Computes $\tilde{m} = h(m||U)$.
   4. Accepts the signature, because $m' = \tilde{m}$.

Rabin signature scheme (cont.)

Example. Use the Rabin signature scheme.

- Alice creates a public key and a private key.
- Alice sends the plaintext $m =$ "*Cryptography*" together with the digital signature for $h(m)$, where h is ToyHash.
- Bob verifies Alice's signature.

Key generation. Alice:

1. Selects primes $p = 29$ and $q = 31$.
2. Computes $n = pq = 899$.
3. Alice's public key is $n = 899$; her private key is $(p, q) = (29, 31)$.

Rabin signature scheme (cont.)

1. *Signature generation.* Alice:
   1. Chooses the padding $U =$" pk cr" and $m||U =$"Cryptography pk cr".
   Then $\tilde{m} = h(m||U) = 4^2 + 4 \cdot 12 + 3 = 67$ is a quadratic residue modulo $n = 899$, because we have $\left(\frac{67}{899}\right) = 1$ (Legendre symbol).
   2. Solves $x^2 = \tilde{m} \pmod{n}$, that is, $x^2 = 67 \pmod{899}$.
   Alice gets the solutions 316, 409, 490 and 583 [...].
   3. Alice's signature for m is $(U, a) = (\text{" } pk\ cr'', 409)$.
2. *Verification.* Bob:
   1. Gets Alice's public key $(p, q) = (29, 31)$.
   2. Computes $m' = a^2 \bmod n = 409^2 \bmod 899 = 67$.
   3. Computes $\tilde{m} = h(m||U) = 67$.
   4. Accepts the signature, because $m' = \tilde{m}$.

ElGamal signature scheme

- security: Discrete Logarithm Problem, Diffie-Hellman Problem
- ElGamal requires a hash function $h : \{0,1\}^* \to \mathbb{Z}_p$ for some large prime p, where $\{0,1\}^*$ denotes the set of strings of bits
- sends the plaintext m together with the digital signature for the hash value $h(m)$
- Data Signature Algorithm (DSA), adopted as Digital Signature Standard (DSS) in 1993 and revised afterwards, is in fact a version of the ElGamal signature scheme

Key generation. Alice creates a public key and a private key.

1. Generates a large random prime p and a generator α of the (cyclic) multiplicative group $\mathbb{Z}_p^*$.
2. Selects a random integer a such that $1 \leq a \leq p-2$.
3. Computes $y = \alpha^a \bmod p$.
4. Alice's public key is (p, α, y); her private key is a.

ElGamal signature scheme (cont.)

Signature generation and verification.

1. *Signature generation.* Alice signs a message m.
   1. Selects a random integer k such that $1 \leq k \leq p-2$ with $gcd(k, p-1) = 1$.
   2. Computes $r = \alpha^k \bmod p$.
   3. Computes $k^{-1} \bmod (p-1)$.
   4. Computes $s = k^{-1}(h(m) - ar) \bmod (p-1)$.
   5. Alice's signature for m is the pair (r, s).
2. *Verification.* Bob verifies Alice's signature.
   1. Gets Alice's public key (p, α, y).
   2. Verifies that $1 \leq r \leq p-2$; if not then he rejects the signature.
   3. Computes $v_1 = y^r r^s \bmod p$.
   4. Computes $h(m)$ and $v_2 = \alpha^{h(m)} \bmod p$.
   5. Accepts the signature if and only if $v_1 = v_2$.

ElGamal signature scheme (cont.)

Proof of correctness. If (r, s) is the signature on the message m, then

$$s = k^{-1}(h(m) - ar) \pmod{p-1}.$$

Multiplying both sides by k we get

$$ks = h(m) - ar \pmod{p-1},$$

that is,

$$h(m) = ar + ks \pmod{p-1}.$$

This implies that

$$\alpha^{h(m)} = \alpha^{ar+ks} = (\alpha^a)^r r^s \pmod{p}.$$

Hence $v_1 = v_2$, as required.

ElGamal signature scheme (cont.)

Example. Use the ElGamal signature scheme.

- Alice creates a public key and a private key.
- Alice sends the plaintext $m =$ "*Cryptography*" together with the digital signature for $h(m)$, where h is ToyHash.
- Bob verifies Alice's signature.

Key generation. Alice:
1. Selects the prime $p = 2357$ and a generator $\alpha = 2$ of the (cyclic) multiplicative group $\mathbb{Z}_p^*$.
2. Selects a random integer $a = 1751$ such that $1 \leq a \leq p - 2$.
3 Computes $y = \alpha^a \bmod p = \cdots = 1185$.
4. Alice's public key is $(p, \alpha, y) = (2357, 2, 1185)$; her private key is $a = 1185$.

ElGamal signature scheme (cont.)

1. *Signature generation.* Alice:
   1. Selects a random number $k = 1529$ such that $1 \leq k \leq p-2$ with $gcd(k, p-1) = 1$.
   2. Computes $r = \alpha^k \bmod p = 2^{1529} \bmod 2357 = \cdots = 1490$.
   3. Computes $k^{-1} \bmod (p-1) = 1529^{-1} \bmod 2356 = \cdots = 245$.
   4. Computes $s = k^{-1}(h(m) - ar) \bmod (p-1) = 245 \cdot (11 - 1751 \cdot 1490) \bmod 2356 = \cdots = 1793$.
   5. Alice's signature for m is $(r, s) = (1490, 1793)$.
2. *Verification.* Bob:
   1. Gets Alice's public key $(p, \alpha, y) = (2357, 2, 1185)$.
   2. Verifies that $1 \leq r \leq p-2$, that is, $1 \leq 1490 \leq 2355$.
   3. Computes $v_1 = y^r r^s \bmod p = 1185^{1490} \cdot 1490^{1793} \bmod 2357 = \cdots = 2048$.
   4. Computes $h(m) = 11$ and $v_2 = \alpha^{h(m)} \bmod p = 2^{11} \bmod 2357 = \cdots = 2048$.
   5. Accepts the signature, because $v_1 = v_2$.

Selective Bibliography


M. Cozzens, S.J. Miller, *The Mathematics of Encryption: An Elementary Introduction*, American Mathematical Society, 2013.


A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997. [http://www.cacr.math.uwaterloo.ca/hac]


Wikipedia, Hash functions.