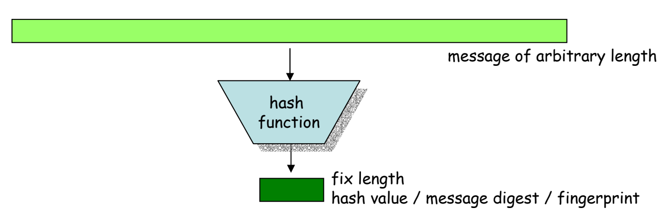
Hash functions

- definition and properties
- birthday paradox
- a provably secure construction
- iterative hash functions
- hash functions based on block ciphers
- customized hash functions (SHA-1)

Definition



- a hash function maps bit strings of arbitrary finite length to bit strings of fixed length (n bits)
- many-to-one mapping → collisions are unavoidable
- however, finding collisions are difficult → the hash value of a message can serve as a compact representative image of the message (similar to fingerprints)

Properties

- compression
  - by definition
- ease of computation
  - given an input x, the hash value h(x) of x is easy to compute
- weak collision resistance (2^{nd} preimage resistance)
  - given an input x, it is computationally infeasible to find a second input x' such that h(x') = h(x)
- strong collision resistance (collision resistance)
  - it is computationally infeasible to find any two distinct inputs x and x' such that h(x) = h(x')
- one-way hash function (preimage resistance)
  - given a hash value y (for which no preimage is known), it is computationally infeasible to find any input x s.t. h(x) = y

Motivation for the properties

- weak collision resistance
  - assume that the hash-and-sign paradigm is used
  - signed message: (m, σ_B(h(m)))
  - if an attacker can find m' such that h(m') = h(m), then he can forge a signed message (m', σ_B(h(m'))) = (m', σ_B(h(m)))
- strong collision resistance
  - the same setup as above but assume that the attacker can choose the message that B signs
  - now it is enough to find a collision pair (m, m')
  - the attacker obtains the signature σ_B(h(m)) on m from B and claims that m' has been signed by presenting (m', σ_B(h(m)))
- one-way property
  - RSA signature on y is y^d mod n
  - the attacker chooses a random value z and computes y = z^e mod n
  - if the attacker can find an x, such that h(x) = y, then he can forge a signed message (x, $(h(x))^d$ mod n) = (x, z)

Relationship between the properties

- strong collision resistance implies weak collision resistance
  - assume that h is strongly collision resistant but not weakly collision resistant
  - given an input x, one can find an x', such that h(x) = h(x')
  - (x, x') is a collision pair → contradicts the assumption that h is strongly collision resistant

- strong collision resistance implies the one-way property
  - if one can find preimages easily, then she can also find collisions easily
  - here's a Las Vegas algorithm for finding collisions:

```
choose a random x
compute h(x)
find x' such that h(x') = h(x)   // this is easy by assumption
if x' = x then output "failure"
else output the collision (x, x')
```

Hash functions / Properties

Relationship between the properties

- THEOREM: Let $h: X \rightarrow Y$, where $|X| \geq 2|Y|$. The success probability of the above algorithm is at least ½.
- proof:
  - let $Z_y = \{x : h(x) = y\}$
  - given h(x), one can find an x' such that h(x') = h(x)
  - the probability of $x \neq x'$ is $(|Z_{h(x)}|-1)/|Z_{h(x)}|$
  - the success probability of the algorithm is

$$\Sigma_{x \in X} (1/|X|) ((|Z_{h(x)}|-1)/|Z_{h(x)}|) =$$
$$(1/|X|) \Sigma_{y \in Y} \Sigma_{x \in Z_y} (|Z_{h(x)}|-1)/|Z_{h(x)}| =$$
$$(1/|X|) \Sigma_{y \in Y} \Sigma_{x \in Z_y} (|Z_y|-1)/|Z_y| =$$
$$(1/|X|) \Sigma_{y \in Y} (|Z_y|-1) =$$
$$(|X|-|Y|)/|X| \geq$$
$$(|X|-|X|/2)/|X| = \tfrac{1}{2}$$

Hash functions / Properties

Birthday paradox

Two variants:

- when drawing elements randomly (with replacement) from a set of N elements, with high probability a repeated element will be encountered after ~sqrt(N) selections
- if we have a set of N elements, and we randomly select two subsets of size ~sqrt(N) each, then with high probability, the intersection of the two subsets will not be empty

These facts have a profound impact on the design of hash functions (and other cryptographic algorithms and protocols)!

Birthday paradox

- Given a set of N elements, from which we draw k elements randomly (with replacement). What is the probability of encountering at least one repeating element?
- first, compute the probability of no repetition:
  - the first element x_1 can be anything
  - when choosing the second element x_2, the probability of $x_2 \neq x_1$ is 1-1/N
  - when choosing x_3, the probability of $x_3 \neq x_2$ and $x_3 \neq x_1$ is 1-2/N
  - …
  - when choosing the k-th element, the probability of no repetition is 1-(k-1)/N
  - the probability of no repetition is (1 - 1/N)(1 - 2/N)…(1 – (k-1)/N)
  - when x is small, $(1-x) \approx e^{-x}$
  - $(1 - 1/N)(1 - 2/N)\ldots(1 - (k-1)/N) = e^{-1/N}e^{-2/N} \ldots e^{-(k-1)/N} = e^{-k(k-1)/2N}$
- the probability of at least one repetition after k drawing is

$$1 - e^{-k(k-1)/2N}$$

Birthday paradox

- How many drawings do you need, if you want the probability of at least one repetition to be ε?
- solve the following for k:

$$\varepsilon = 1 - e^{-k(k-1)/2N}$$
$$k(k-1) = 2N \ln(1/1-\varepsilon)$$
$$k \approx \mathrm{sqrt}(2N \ln(1/1-\varepsilon))$$

- examples:
  - $\varepsilon = \frac{1}{2} \rightarrow k \approx 1.177\ \mathrm{sqrt}(N)$
  - $\varepsilon = \frac{3}{4} \rightarrow k \approx 1.665\ \mathrm{sqrt}(N)$
  - $\varepsilon = 0.9 \rightarrow k \approx 2.146\ \mathrm{sqrt}(N)$
- origin of the name “birthday paradox”:
  - elements are dates in a year (N = 365)
  - among 1.177 sqrt(365) $\approx$ 23 randomly selected people, there will be at least two that have the same birthday with probability $\frac{1}{2}$

Hash functions / Birthday paradox

Choosing the output size of a hash function

- good hash functions can be modeled as follows:
  - given a hash value y, the probability that a randomly chosen input x maps to y is $\sim 2^{-n}$
  - the probability that two randomly chosen inputs x and x’ map into the same hash value is also $\sim 2^{-n}$
  - → n should be at least 64, but 80 is even better
- birthday attacks
  - among $\sim\mathrm{sqrt}(2^n) = 2^{n/2}$ randomly chosen messages, with high probability there will be a collision pair
  - it is easier to find collisions than to find preimages or 2nd preimages for a given hash value
  - → in order to resist birthday attacks, n should be at least 128, but 160 is even better

Hash functions / Birthday paradox

A discrete log hash function

- construction:
  - let p be a large prime such that q = (p-1)/2 is also prime
  - let a and b be two primitive elements of Z_p*
  - computing x such that a^x mod p = b is difficult (discrete log problem)
  - let h: {0, 1, …, q-1}x{0, 1, …, q-1} → Z_p* be the following:

$$h(x_1, x_2) = a^{x_1} b^{x_2} \bmod p$$

  - if p is k bit long, then h maps 2(k-1) bits into k bits

- THEOREM: if one can find a collision for h, then she can efficiently compute $dlog_a b$

Proof of the theorem

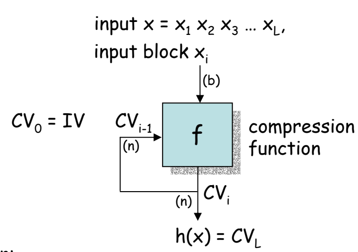
- suppose there's a collision $h(x_1, x_2) = h(x_3, x_4)$
- then we know that $a^{x_1-x_3} \equiv b^{x_4-x_2}$ (mod p)
- since $((x_1, x_2), (x_3, x_4))$ is a collision, $(x_1, x_2) \neq (x_3, x_4)$
- without loss of generality, assume that $x_2 \neq x_4$
- let $d = gcd(x_4-x_2, p-1)$
- since p-1 = 2q, and q is prime, there are four cases:
  - d = 1
  - d = 2
  - d = q
  - d = 2q = p-1
- but $0 \leq x_2, x_4 < q$, and therefore, $-q < x_4-x_2 < q$
- in addition, we know that $x_4-x_2 \neq 0$
- this means that q and 2q cannot divide x_4-x_2
- hence, two cases remain: d = 1 and d = 2

Proof of the theorem

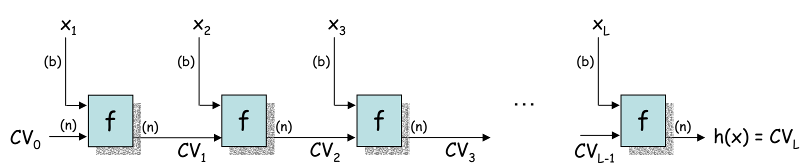
- d = 1
  - this means that x_4-x_2 and p-1 are relative primes, and thus, x_4-x_2 has an inverse mod p-1
  - $y = (x_4-x_2)^{-1} \pmod{p-1}$
  - $b^{(x_4-x_2)y} = b^{k(p-1)+1} = b(b^{p-1})^k \equiv b \pmod{p}$
  - $b^{(x_4-x_2)y} \equiv a^{(x_1-x_3)y} \pmod{p}$
  - thus, $b \equiv a^{(x_1-x_3)y} \pmod{p}$, and so $\mathrm{dlog}_a b = (x_1-x_3)y \pmod{p-1}$
- d = 2
  - $b^{p-1} = (b^q)^2 \equiv 1 \pmod{p}$ → b^q is a square root of 1 (mod p)
  - b^q cannot be 1, since b is a primitive element → $b^q \equiv -1 \pmod{p}$
  - since $\gcd(x_4-x_2, 2q) = 2$, we must have $\gcd(x_4-x_2, q) = 1$
  - let $y = (x_4-x_2)^{-1} \pmod{q}$
  - $b^{(x_4-x_2)y} = b^{kq+1} = b(b^q)^k \equiv b(-1)^k = \pm b \pmod{p}$
  - thus, either
    - $b \equiv b^{(x_4-x_2)y} \equiv a^{(x_1-x_3)y} \pmod{p}$, or
    - $b \equiv -b^{(x_4-x_2)y} \equiv -a^{(x_1-x_3)y} \equiv a^q\, a^{(x_1-x_3)y} = a^{q+(x_1-x_3)y} \pmod{p}$

Iterated hash functions

- input is divided into fixed length blocks
- last block is padded if necessary
- each input block is processed according to the following scheme



alternative illustration:



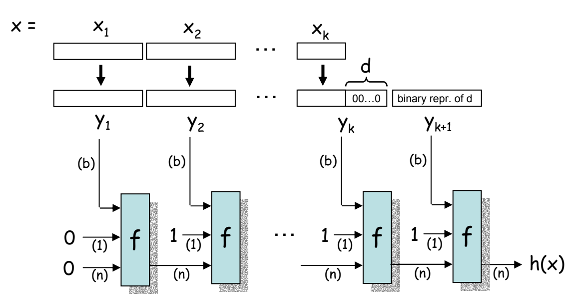
Exercise

- Assume that an iterated hash function h has a small output size such that h is not collision resistant (the birthday attack works). One may try to increase the output size by using the last two chaining variables as the output:

$$h'(x) = CV_{L-1}|CV_L$$

Prove that this is insecure by showing that h' is still not collision resistant.

Merkle-Damgard (MD) strengthening



THEOREM: if f is strongly collision resistant, then h is strongly collision resistant too

Proof of the MD theorem

- let's assume that one has found a collision pair (x, x') for h
- there are three possible cases:
  1. $|x| \not\equiv |x'| \pmod b$

  2a. $|x| \equiv |x'| \pmod b$ and $|x| = |x'|$

  2b. $|x| \equiv |x'| \pmod b$ but $|x| \neq |x'|$

- case 1:
  - $d \neq d' \rightarrow y_{k+1} \neq y'_{k'+1}$
  - $f(cv_k|1|y_{k+1}) = h(x) = h(x') = f(cv'_{k'}|1|y'_{k'+1})$
  - $(cv_k|1|y_{k+1}, cv'_{k'}|1|y'_{k'+1})$ is a collision for f
  - this contradicts with the assumption that f is collision resistant

Hash functions / Iterated hash functions

Proof of the MD theorem

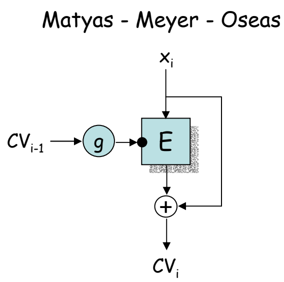
- case 2a:
  - $y_{k+1} = y'_{k+1}$
  - $f(cv_k|1|y_{k+1}) = h(x) = h(x') = f(cv'_k|1|y'_{k+1})$
  - $cv_k = cv'_k$ since otherwise we found a collision for f
  - $f(cv_{k-1}|1|y_k) = cv_k = cv'_k = f(cv'_{k-1}|1|y'_k)$
  - $cv_{k-1} = cv'_{k-1}$ and $y_k = y'_k$ since otherwise we found a collision for f
  - …
  - $f(0^{n+1}|y_1) = cv_1 = cv'_1 = f(0^{n+1}|y'_1)$
  - $y_1 = y'_1$ since otherwise we found a collision for f
  - this means that $y_i = y'_i$ for all i = 1, 2, …, k+1
  - hence x = x', but this contradicts with the assumption that (x, x') is a collision pair

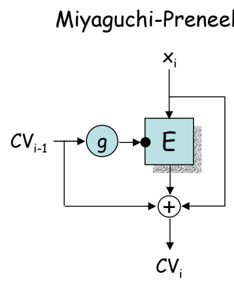
Hash functions / Iterated hash functions

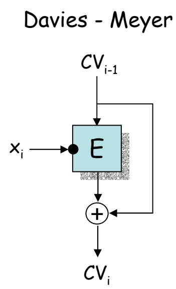
Proof of the MD theorem

- case 2b:
  - $y_{k+1} = y'_{k'+1}$
  - $f(cv_k|1|y_{k+1}) = h(x) = h(x') = f(cv'_{k'}|1|y'_{k'+1})$
  - $cv_k = cv'_{k'}$ since otherwise we found a collision for f
  - $f(cv_{k-1}|1|y_k) = cv_k = cv'_{k'} = f(cv'_{k'-1}|1|y'_{k'})$
  - $cv_{k-1} = cv'_{k'-1}$ and $y_k = y'_{k'}$ since otherwise we found a collision for f
  - …
  - assume that $k < k'$
  - …
  - $f(0^{n+1}|y_1) = cv_1 = cv'_{k'-k+1} = f(cv'_{k'-k}|1|y'_{k'-k+1})$
  - $(0^{n+1}|y_1 , cv'_{k'-k}|1|y'_{k'-k+1})$ is a collision pair for f, because they differ in their (n+1)st bits
  - this contradicts with the assumption that f is collision resistant

Hash functions based on block ciphers



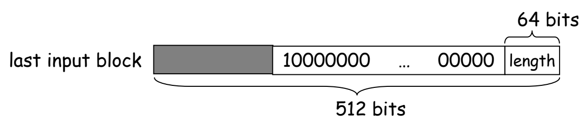




Hash functions / SHA 1

SHA1 – Secure Hash Algorithm

- output size (n): 160 bits
- input block size (b): 512 bits
- padding is always used



- CV_0

  A = `67 45 23 01`

  B = `EF CD AB 89`

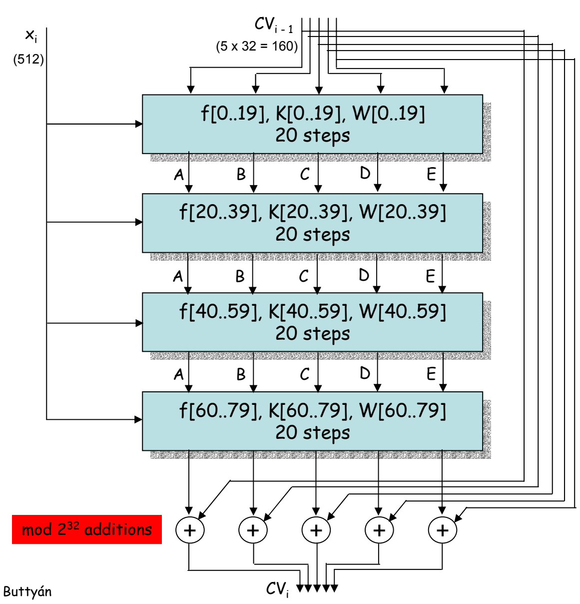
  C = `98 BA DC FE`

  D = `10 32 54 76`

  E = `C3 D2 E1 F0`

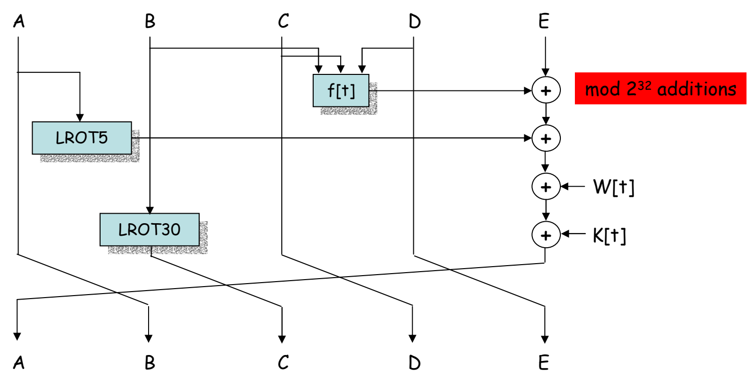
Hash functions / SHA 1

SHA1 compression function f



© Levente Buttyán

SHA1 compression function f cont'd



SHA1 compression function f cont'd

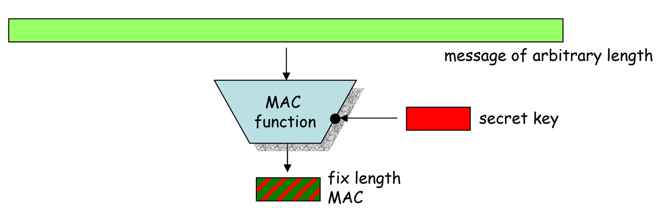
- f[t](B, C, D)
  - t = 0..19 f[t](B, C, D) = $(B \wedge C) \vee (\neg B \wedge D)$
  - t = 20..39 f[t](B, C, D) = $B \oplus C \oplus D$
  - t = 40..59 f[t](B, C, D) = $(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
  - t = 60..79 f[t](B, C, D) = $B \oplus C \oplus D$
- W[t]
  - W[0..15] = x_i
  - t = 16..79 W[t] = LROT1(W[t-16] $\oplus$ W[t-14] $\oplus$ W[t-8] $\oplus$ W[t-3])
- K[t]
  - t = 0..19 K[t] = `5A 82 79 99` [$2^{30} \times 2^{1/2}$]
  - t = 20..39 K[t] = `6E D9 EB A1` [$2^{30} \times 3^{1/2}$]
  - t = 40..59 K[t] = `8F 1B BC DC` [$2^{30} \times 5^{1/2}$]
  - t = 60..79 K[t] = `CA 62 C1 D6` [$2^{30} \times 10^{1/2}$]

Message authentication codes

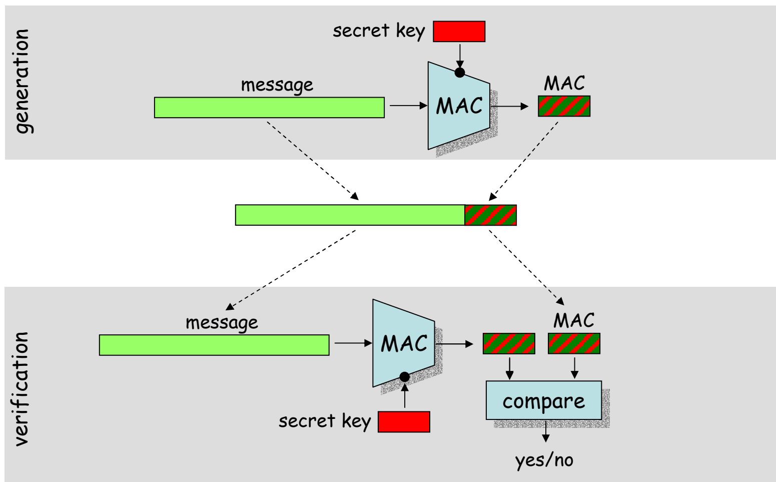
- definition and properties
- constructions based on block ciphers
- constructions based on hash functions

Definition

- MAC functions can be viewed as hash functions with two functionally distinct inputs: a message and a secret key
- they produce a fixed size output (say n bits) called the MAC
- practically it should be infeasible to produce a correct MAC for a message without the knowledge of the secret key
- MAC functions can be used to implement data integrity and message origin authentication services



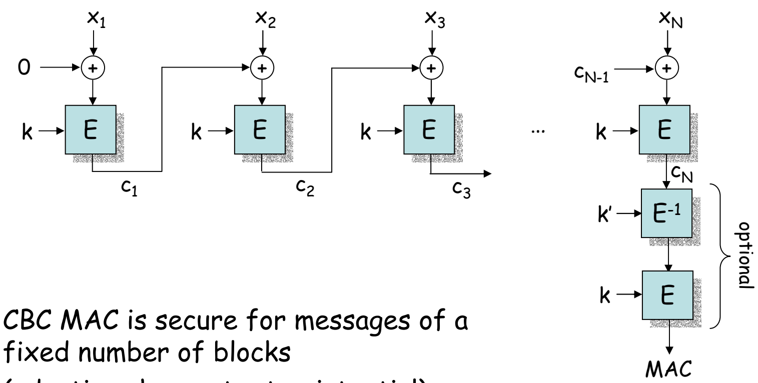
MAC generation and verification



Properties

- ease of computation
  - given an input x and a secret key k, it is easy to compute $MAC_k(x)$
- compression
  - MAC_k maps an input of arbitrary finite length to an output of fixed length (n bits)
- key non-recovery
  - it is computationally infeasible to recover the secret key k, given one or more text-MAC pairs $(x_i, MAC_k(x_i))$ for that k
- computation resistance
  - given zero or more text-MAC pairs $(x_i, MAC_k(x_i))$, it is computationally infeasible to find a text-MAC pair $(x, MAC_k(x))$ for any new input $x \neq x_i$
  - computation resistance implies key non-recovery but the reverse is not true in general

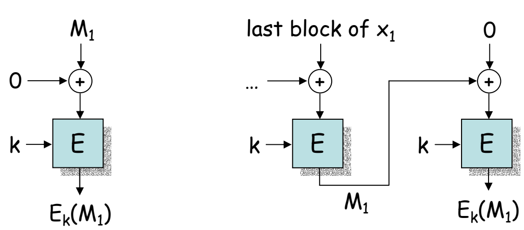
CBC MAC



- CBC MAC is secure for messages of a fixed number of blocks
- (adaptive chosen-text existential) forgery is possible if variable length messages are allowed

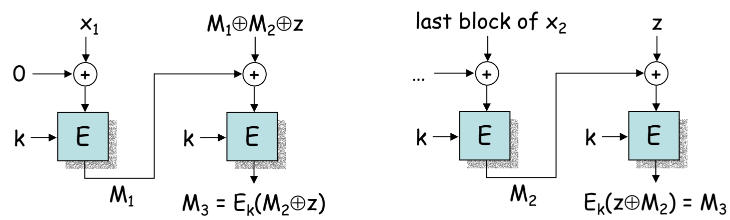
Existential forgery of CBC MAC

- example 1
  - given a known text-MAC pair (x_1, M_1)
  - request MAC for M_1, receive $M_2 = E_k(M_1 \oplus 0) = E_k(M_1)$
  - M_2 is the MAC of the two block message $(x_1|0)$



Existential forgery of CBC MAC

- example 2
  - given two known text-MAC pairs: (x_1, M_1), (x_2, M_2)
  - request MAC for message $x_1|M_1 \oplus M_2 \oplus z$, where z is an arbitrary block
  - receive $M_3 = E_k(M_1 \oplus M_2 \oplus z \oplus M_1) = E_k(M_2 \oplus z)$
  - M_3 is also the MAC for message $x_2|z$

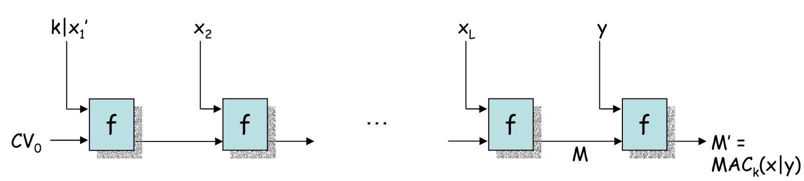


© Levente Buttyán

Secret prefix method

- $MAC_k(x) = h(k|x)$
  - insecure
    - assume an attacker knows the MAC on x: $M = h(k|x)$
    - he can produce the MAC on $x|y$ as $M' = f(M,y)$, where f is the compression function of h

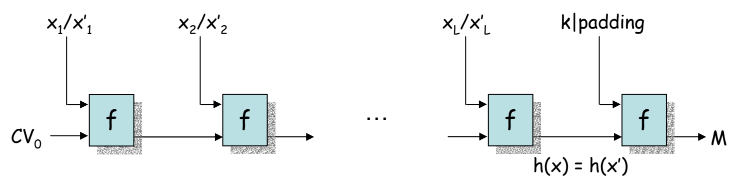
$x = x_1'|x_2|\ldots|x_L$



© Levente Buttyán

Secret suffix method

- $MAC_k(x) = h(x|k)$
  - may be insecure
    - using a birthday attack, the attacker finds two inputs x and x' such that h(x) = h(x') (can be done off-line)
    - then obtaining the MAC M on one of the inputs, say x, allows the attacker to forge a text-MAC pair (x', M)
  - weaknesses
    - key is involved only in the last step
    - MAC depends only on the last chaining variable



HMAC

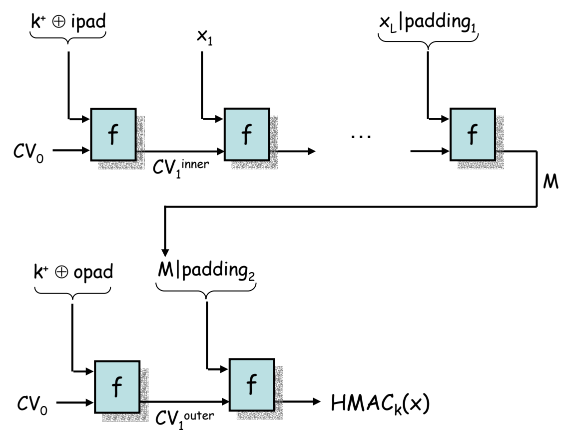
- definition

  $HMAC_k(x) = h(\ (k^+ \oplus opad)\ |\ h(\ (k^+ \oplus ipad)\ |\ x\ )\ )$

  where
  - h is a hash function with input block size b and output size n
  - k^+ is k padded with 0s to obtain a length of b bits
  - ipad is 00110110 repeated b/8 times
  - opad is 01011100 repeated b/8 times

- design objectives
  - to use available hash functions
  - easy replacement of the embedded hash function
  - preserve performance of the original hash function
  - handle keys in a simple way
  - allow mathematical analysis

HMAC illustrated



Digital signatures

- definitions
- types of attacks
- the "hash-and-sign" paradigm
- the RSA signature scheme
- the ElGamal signature scheme

Definition

- similar to MACs but
  - unforgeable by the receiver
  - verifiable by a third party
- used for message authentication and non-repudiation (of message origin)
- based on public-key cryptography
  - private key defines a signing transformation S_A
    - $S_A(m) = \sigma$
  - public key defines a verification transformation V_A
    - $V_A(m, \sigma)$ = true if $S_A(m) = \sigma$
    - $V_A(m, \sigma)$ = false otherwise

Types of attacks on signature schemes

- classification of attacks based on the goal of the attacker
  - total break
    - the attacker is able to compute the private key of the signer or finds an efficient singing algorithm functionally equivalent to the valid signing algorithm
  - selective forgery
    - the attacker is able to compute a valid signature for a particular message or class of messages
    - the legitimate signer is not involved directly
  - existential forgery
    - the attacker is able to forge a signature for at least one message
    - the attacker may not have control over the message for which the signature is obtained
    - the legitimate signer may be involved in the deception

Digital signatures / Definitions

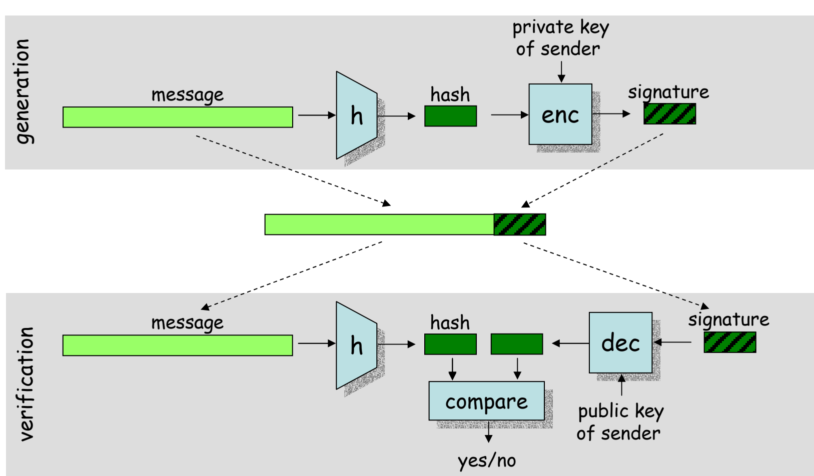
Types of attacks on signature schemes

- classification of attacks based on the means of the attacker
  - key-only attack
    - only the public key is available to the attacker
  - known-message attack
    - the attacker has signatures for a set of messages known to the attacker but not chosen by him
  - chosen-message attack
    - the attacker obtains signatures for messages chosen by him before attempting to break the signature scheme
  - adaptive chosen-message attack
    - the attacker is allowed to use the signer as an oracle
    - he may request signatures for messages which depend on previously obtained signatures

Digital signatures / Hash-and-sign paradigm

"Hash-and-sign" paradigm

- motivation: public/private key operations are slow
- approach: hash the message first and apply public/private key operations to the hash value only



Yuval's birthday attack

- input: legitimate message m_1, fraudulent message m_2
- output: messages m_1', m_2' such that
  - m_1' and m_2' are minor modifications of m_1 and m_2, respectively
  - $h(m_1') = h(m_2')$
- generate $t = 2^{n/2}$ minor modifications of m_1
- hash each modifications and store the hash values
- generate a minor modification m_2' of m_2, compute its hash value $h(m_2')$, and look for matches among the stored hash values
- repeat the above step until a match is found (this is expected after t steps)
- complexity: $2^{n/2}$ storage and $\sim 2^{n/2}$ processing
- consequences: a signature on m_1' is also a valid signature on m_2'

RSA signature scheme

- signature generation (input: m)
  - compute $\mu = h(m)$
  - (PKCS #1 formatting)
  - compute $\sigma = \mu^d \bmod n$
- signature verification (input: m, σ)
  - obtain the authentic public key (n, e)
  - compute $\mu' = \sigma^e \bmod n$
  - (PKCS #1 processing, reject if μ' is not well formatted)
  - compute $\mu = h(m)$
  - compare μ and μ'
    - if they match, then output true
    - otherwise, output false

ElGamal signature scheme

- basis of the Digital Signature Standard (DSS)
- ElGamal is a randomized signature scheme
- key generation
  - generate a large random prime p and select a generator g of Z_p^*
  - select a random integer $1 \leq a \leq p-2$
  - compute $A = g^a \bmod p$
  - public key: (p, g, A)   private key: a
- signature generation for message m
  - select a random secret integer $1 \leq r \leq p-2$ such that gcd(r, p – 1) = 1
  - compute $r^{-1} \bmod (p - 1)$
  - compute $R = g^r \bmod p$
  - compute $S = r^{-1}(h(m) - aR) \bmod (p - 1)$
  - signature on m is (R, S)

Digital signatures / ElGamal

ElGamal signature scheme

- signature verification
  - obtain the public key (p, g, A) of the signer
  - verify that 0 < R < p; if not then reject the signature
  - compute $v_1 = A^R R^S \bmod p$
  - compute $v_2 = g^{h(m)} \bmod p$
  - accept the signature iff $v_1 = v_2$
- proof that signature verification works

  $S \equiv r^{-1}(h(m) - aR) \pmod{p - 1}$

  $rS \equiv h(m) - aR \pmod{p - 1}$

  $h(m) \equiv rS + aR \pmod{p - 1}$

  $g^{h(m)} \equiv g^{aR+rS} \equiv (g^a)^R (g^r)^S \equiv A^R R^S \pmod{p}$

  thus, $v_1 = v_2$ is required

Digital signatures / ElGAmal