Algoritmi di Crittografia

Corso di Laurea Magistrale in Informatica

A.A. 2018/2019

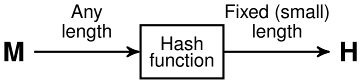
1. Hash Functions
   - General concepts
   - Building hash functions
   - Real hash functions

Algoritmi di Crittografia

1. Hash Functions
   - General concepts
   - Building hash functions
   - Real hash functions

Hash functions

- In general terms, a *hash function* is a function that takes an arbitrary long input **M** and returns a fixed size output



- "Long input" may refer, say, to a long piece of text or to a very large number (depending on the application)
- The output can be generally seen as a sequence of bits, although in some applications it is rather interpreted as an integer number (i.e, an index to a table)
- For all the conceivable applications, though, the main property of a hash function is that it must appear *random*, i.e., returns "unexpected" values

Cryptographic hash functions

- *Cryptographic Hash Functions* (*CHF*s) are often referred to as (message) *fingerprints* or *digests*
- However, there are many more applications of CHFs than those implied by these terms
- With respect to hash functions used in other application settings, a CHF must exhibit additional properties
- The fundamental (and, in some sense "ideal") requirement for crypto is that hash functions must be *one-way*
- Being one-way for a function f means that it "easy" to compute $f(x)$, for any x, but "hard" to invert, i.e., given y compute x such that $f(x) = y$
- This is also known as (*first*) *pre-image resistance*

Some cryptographic applications of hash Functions

- User authentication
- Digital signatures
- Message Authenticity and integrity
- File identifiers
- Program obfuscation

Other important properties of a CHF f

- *Second pre-image resistance*: given x, it is computationally hard to find $x' \neq x$ such that $f(x) = f(x')$
- *Collision resistance*: find a pair $x \neq x'$ such that $f(x) = f(x')$
- (*Lossy*) *compression*: since the size of hash values is fixed, for the majority of inputs f does behave like a lossy compressor

Significance of the fundamental properties

- First pre-image resistance is required in application contexts like *password authentication* and *key derivation*
- We have seen a notable example of key derivation when we discussed the Fortuna generator
- Also second pre-image resistance is crucial to protect user authentication through passwords
- In fact, even is the colliding value is not the “true” password, it is equally good to grant system access
- A collision resistant hash function *f* is suitable for use in all the applications where it is crucial to detect possible modifications in the input (be it unintentional or malicious): digital signatures, intrusion detection systems, forensic analysis, and many others

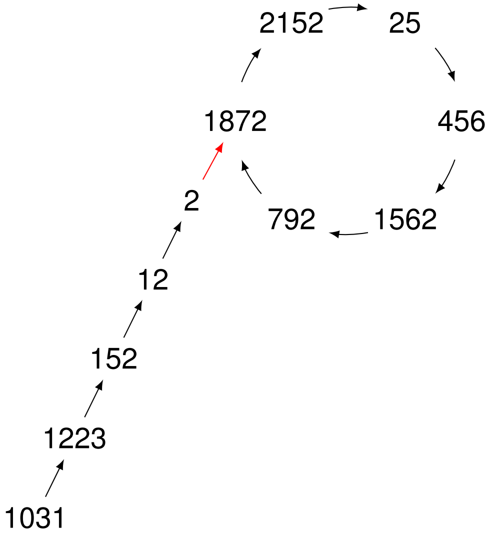
Attacks to find collisions

- Technically, there should be no significantly better attack to find a collision other than the generic *birthday attack*
- Let n be the length of the hash values. Thus there are 2^n possible hash values, i.e., days
- Then, we expect to find a collision (with prob. $> \frac{1}{2}$) provided that we choose $2^{n/2}$ random inputs, i.e., persons
- To avoid to check $O(2^n)$ pairs, we can store a lookup table with $2^{n/2}$ messages indexed by the corresponding hash values (a lot of memory...)
- A different attack, known as *Rho method*, avoid storing such a huge table by paying a little more in terms of time

Finding collisions with the Rho method

- Starting from an arbitrary value x_0, the Rho methods computes a sequence $\{x_i\}_{i \in \mathbf{N}}$ where $x_i = H(x_{i-1})$
- Since the codomain of H is finite (huge, but finite), at some point the values repeat
- That is, there must be an intex j such that, for some $i < j$, $x_j = x_i$
- After that we have $x_{i+k} = x_{j+k}$, for any $k \geq 0$
- The situation is depicted in the next slide, that justifies the name of the method
- In this (toy) example we have $x_0 = 1031$ and $x_{11} = x_5 = 1872$

Finding collisions with the Rho method (cont.d)



Finding collisions with the Rho method (cont.d)

- As it can be seen, the first repeated value (i.e., $x_j = x_i$) is the pre-image of two different values, i.e., x_{j-1} and x_{i-1} are the colliding values we want to compute
- First some terminology
  - The subsequence from x_0 to x_i (the "junction" value) is the *tail* of the rho; denote its length by λ
  - The rest of the sequence is the *cycle*, or *head*; denote its length by σ
  - Note that, by convention, the head is traveled clockwise
- Let r be a random value, and consider the two sequences:

$$\begin{aligned} x_0 &= r \\ x_i &= H(x_{i-1}) \quad i = 1, 2, \ldots \end{aligned}$$

and

$$\begin{aligned} y_0 &= r \\ y_i &= H(H(y_{i-1})) = x_{2i} \quad i = 1, 2, \ldots \end{aligned}$$

Finding collisions with the Rho method (cont.d)

- Now, regard the two sequences as they were racers! At the starter fire the race begins with $x_0 = y_0 = r$
- The y-racer suddenly goes at the head (unless the pathological case $x_1 = x_0$ occurs) while x is chasing
- Since y is faster, x cannot reach y while y is running the tail
- At some time, y enters the head and starts cycling while x tries to reach the junction point as well
- When finally x enters the cycle, y is already somewhere in (possibly exactly at the junction and possibly after having traveled the cycle more times)
- At this point we give x one more chance to win the game, namely to complete a full cycle and arrive again at the junction point before being reached by y
- However this also cannot happen since the distance between x and y shrinks by one at each step (try to work out the details)

Finding collisions with the Rho method (cont.d)

- From the above informal discussion, it follows that there exists a value t such that $y_t = x_t$ (i.e., t is where y reaches x in the cycle), and t satisfies:

$$t = \lambda + c, \quad \text{with} \quad c \geq 0 \quad \text{and} \quad c < \sigma$$

- We now re-run the x sequence from the starting block $x_0 = r$ and also the sequence

$$\begin{aligned} z_0 &= x_t \\ z_i &= H(z_{i-1}) = x_{i+t} \quad i = 1, 2, \ldots \end{aligned}$$

- That is, the z-sequence now is as fast as x but starts t steps ahead of x
- The crucial observation is that these two sequences meet precisely at the junction point, which means $x_\lambda = z_\lambda$

Finding collisions with the Rho method (cont.d)

Proof that $x_\lambda = z_\lambda$
We know that

$$t = \lambda + c$$

However, since the y sequence is twice as fast as x and reaches x at t, we can also write

$$2t = \lambda + n\sigma + c$$

for some $n \geq 1$. But then t can also be expressed as $t = n\sigma$ (i.e., an integer multiple of the cycle length) and this implies $z_\lambda = x_{\lambda+n\sigma}$ but the latter is clearly equal to x_λ

The Rho algorithm

- Choose r at random and check that $H(r) \neq r$ (otherwise repeat)
- Set $x_0 = r$, $y_0 = r$ and compute

$$x_i = H(x_{i-1}), \quad y_i = H(H(y_{i-1})), \qquad i = 1, 2, \ldots$$

  until, for some index t, $x_t = y_t$
- Now set $z_0 = x_t$ and $x_0 = r$, and compute

$$x_i = H(x_{i-1}), \quad z_i = H(z_{i-1}), \qquad i = 1, 2, \ldots, \lambda$$

  In doing so, always keep pairs of consecutive iterates, i.e., x_{i-1} and x_i as well as z_{i-1} and z_i.
- When $x_i = z_i$ (that necessarily happens when $i = \lambda$), return x_{i-1} and z_{i-1}

Relative difficulty

- It is not difficult to show that pre-image resistance is at least as hard as second pre-image resistance
- In fact, to find an input that collide with a given input x, we can first compute $H = f(x)$ and the find a pre-image y of H
- In turn, second pre-image resistance is at least as difficult as collision resistance
- In fact, to find a collision you can first create a random input x and then solve the second pre-image problem to find some y that collide with x
- Generic attacks to find a pre-image have average complexity $2^{n/2}$, rather than 2^n

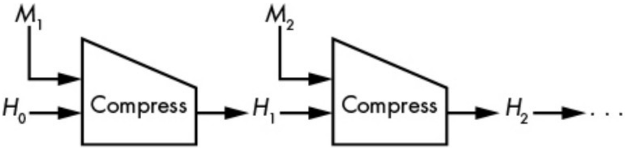
Algoritmi di Crittografia

1. Hash Functions
   - General concepts
   - Building hash functions
   - Real hash functions

An iterative structure

- Almost all hash functions used in real applications embody a technique that reminds the design of block ciphers
- The input M is split in blocks, say $M_1, M_2, \ldots, M_N$, with the last block padded and the blocks are processed in rounds
- There is an internal state that evolves with rounds, starting with a fixed initial state H_0
- The generic round computes $H_{i+1} = c(H_i, M_{i+1})$, $i = 0, \ldots, N-1$, where c is an accurately designed *compression function*, while H_n is the computed hash value
- Clearly, the size of the internal state is the output size (say, 128, 1690, ... bits)
- This design is known as the *Merkle-Damgård* construction
- Actually, there are hash functions families that do not use compression, but we shall not consider them here

The Merkle-Damgård construction



The Merkle–Damgård construction with a generic compression function
Source: J.P. Aumasson, Serious Cryptography, No Starch Press (2018)

Padding

- Padding must ensure that different messages get different hash values
- A padding scheme, specified by ISO/IEC 9797-1 is the following
- First compute the length ℓ of the unpadded message M (length expressed in bits)
- Then prepare the padded message by inserting (in this order):
  1. the value ℓ expressed in big-endian binary in n bits (n is the block length)
  2. the unpadded message
  3. as many (possibly none) bits with value 0 as are required to bring the total length to a multiple of n bits
- For instance, with $n = 128$, a 230 bit message M would be transformed to $\underbrace{\backslash x00 \ldots \backslash x00}_{15} \backslash xe6 M \underbrace{00 \ldots 0}_{26}$

A (perhaps unexpectedly) insecure hash function

- Take and block cipher you trust, say *AES* with 256 bit keys
- Hash functions (at least the ones we are studying here) do not require a key, so take $K = \underbrace{00\ldots0}_{256}$
- Also, take $H_0 = \underbrace{00\ldots0}_{128}$ and define:

$$H_{i+1} = \mathbf{E}(H_i \oplus M_{i+1}), \quad i = 1, \ldots, N$$

  where N is the number of blocks of the (already padded) message and $\mathbf{E}$ is the AES encryption function
- So, if our message (after padding) is made of just two blocks, i.e., $M = M_1 || M_2$, then we have:

$$\begin{aligned} H_1 &= \mathbf{E}(H_0 \oplus M_1) = \mathbf{E}(M_1) \\ H_2 &= \mathbf{E}(H_1 \oplus M_2) \end{aligned}$$

A (perhaps unexpectedly) insecure hash function

- Now assume we can manage to have, after padding, a different message $M' = M'_1 || M'_2$ defined as follows:

$$\begin{aligned} M'_1 &= M_2 \oplus H_1 \\ M'_2 &= H_2 \oplus M_2 \oplus H_1 \end{aligned}$$

- This is not immediate, but can be done (remember we can adapt M as well, since we are looking for arbitrary collisions)

A (perhaps unexpectedly) insecure hash function

- Then we have:

$$\begin{aligned} H_1' &= \mathbf{E}(H_0' \oplus M_1') \\ &= \mathbf{E}(H_0 \oplus M_2 \oplus H_1) \\ &= \mathbf{E}(M_2 \oplus H_1) \end{aligned}$$

and

$$\begin{aligned} H_2' &= \mathbf{E}(H_1' \oplus M_2') \\ &= \mathbf{E}(\mathbf{E}(M_2 \oplus H_1) \oplus H_2 \oplus M_2 \oplus H_1) \\ &= \mathbf{E}(\mathbf{E}(M_2 \oplus H_1) \oplus \mathbf{E}(H_1 \oplus M_2) \oplus M_2 \oplus \mathbf{E}(M_1)) \\ &= \mathbf{E}(M_2 \oplus H_1) \\ &= H_2 \end{aligned}$$

which is also the hash value of the first message

Demo example with ISO/IEC 9797-1 padding scheme

- Under the ISO/IEC 9797-1 padding scheme, the attack outlined in the previous slides involves (before padding) messages of at most 16 bytes (i.e., one block)
- Let m and m' denote the, yet to be determined, unpadded messages that we want to produce a collision
- m and m' can be determined by reasoning "backwards" from the padded counterparts, say, M and M'
- We can then write:

$$\begin{aligned} M &= M_1||M_2 = \underbrace{\backslash x00 \ldots \backslash x00}_{15} \backslash xXY||M_2 \\ M' &= M'_1||M'_2 = \underbrace{\backslash x00 \ldots \backslash x00}_{15} \backslash xWZ||M'_2 \end{aligned}$$

where XY and WZ are the lengths (in bits and base 16) of m and m'

Demo example with ISO/IEC 9797-1 padding scheme

- Does $M_2 = m$ (and/or $M_2' = m'$)? Well, it depends on the presence of possible padding zeros.
- Clearly, if (say) $\backslash xXY = \backslash x80$, i.e., we want the unpadded message exactly 16 byte long, then $M_2 = m$
- On the other hand, if $\backslash xXY = \backslash x78$, i.e., we want the unpadded message 15 byte long, then $M_2 = m||\backslash x00$
- ... and so on

Demo example with ISO/IEC 9797-1 padding scheme

- Since $H_0 = \underbrace{\backslash x00 \ldots \backslash x00}_{16}$, we know that, for the first message,

$$H_1 = \mathbf{E}(M_1)$$

- Now, according to the attack, we must have $M_1' = M_2 \oplus H_1$, and hence set

$$M_2 = M_1' \oplus H_1 = M_1' \oplus \mathbf{E}(M_1)$$

- Note that M_2 depends on both M_1 and M_1', i.e., on $\backslash xXY$ and $\backslash xWZ$

Demo example with ISO/IEC 9797-1 padding scheme

- Now, recall the discussion about the length of m
- Setting $\backslash xXY = \backslash x80$ (i.e., $|m| = 128$), we do not pose any other constraints on M_2 and can safely assume $m = M_2$
- On the contrary, if we set $\backslash xXY = \backslash x78$ (i.e., assume $|m| = 120$) then we require $M_2 = m||\backslash x00$
- This in turn requires that $M_1' \oplus H_1$ ends with 8 zeros
- To satisfy this constraint, we can only use $\backslash xWZ$ as our "free variable", but $\backslash xWZ$ has only 16 possible values ($8i$ for $i = 1, 2, ..., 16$), with no guarantee that at least one works
- The situation becomes worse if we assume $|m| = 112$, since this would require $M_2 = M_1' \oplus H_1 = m||\backslash x00\backslash x00$
- Then we set $\backslash xXY = \backslash x80$

Demo example with ISO/IEC 9797-1 padding scheme

- Summing up, we have $M = \underbrace{\backslash x00 \ldots \backslash x00}_{15} \backslash x80 || M_2$ and

$$\begin{aligned} m &= M_2 = M_1' \oplus H_1 \\ &= \underbrace{\backslash x00 \ldots \backslash x00}_{15} \backslash xWZ \oplus \mathbf{E}(\underbrace{\backslash x00 \ldots \backslash x00}_{15} \backslash x80) \end{aligned}$$

- Now we proceed to determine H_2 and, hence, M_2'
- By definition

$$H_2 = \mathbf{E}(H_1 \oplus M_2) = \mathbf{E}(M_1')$$

and

$$\begin{aligned} M_2' &= H_2 \oplus M_2 \oplus H_1 \\ &= \mathbf{E}(M_1') \oplus M_1' \oplus H_1 \oplus H_i \\ &= \mathbf{E}(M_1') \oplus M_1' \end{aligned}$$

Demo example with ISO/IEC 9797-1 padding scheme

- Now the difficulty we have overcome before pops up again, but now with no apparent solution
- In fact, if we set $\backslash xWZ = \backslash x80$, then $M_1' = M_1$ and also

$$\begin{aligned} M_2' &= \mathbf{E}(M_1') \oplus M_1' \\ &= \mathbf{E}(M_1') \oplus M_1 \\ &= M_2 \end{aligned}$$

  so that we obtain a “collision” simply because $m = m'$
- On the other hand, setting $\backslash xWZ = \backslash x78$ we require $M_2' = m' || \backslash x00$
- This time, however, we do not have any free variable left and must rely only on good luck (with only one chance over 256 possibilities)
- Different settings for $\backslash xWZ$ lead clearly to much smaller probabilities

Demo example with ISO/IEC 9797-1 padding scheme

- Hence, provided that we choose the key with some care, with the ISO/IEC 9797-1 padding scheme this attack does not produce a collision
- In particular, it does not works with the zero key
- However, the analysis clearly shows a potential vulnerability, as a badly chosen key (and there are many) would make an attacker's day
- With $m = 128$ and $m' = 120$ the first bad keys are 38, 83, 580, and 615
- With $m = 128$ and $m' = 112$, the first four keys are 28082, 28716, 70606, and 114903

Security aspects

- Merkle and Damgård proved that the security (i.e., resistance to pre-image and collision attacks) of the hash function is guaranteed if the compression function is secure
- The converse is not necessarily true as the hash can be secure even in cases where the compression function is not secure
- In spite of the latter circumstance, a discovered weakness of the compression function is a serious "alarm bell" ringing about the security of the hash function
- The main concern about hash functions based on the Merkle-Damgård construction if the so-called *length-extension bug*
- In fact, for this construction, if $M' = M||E$, then it holds that:

$$f(M') = c(f(M), E)$$

where (recall) c is the compression function

The Davis-Meyer compression “strategy”

- There is a simple solution to the problem of defining a secure compression function, namely to use a “secure” block cipher
- Well, actually, a block cipher does not “shrink” the size of the input...
- However, a block cipher takes 2 inputs, namely a block B and the key K, and returns the transformed block
- Hence, if we regard the input as $B||K$, then we can think of a block cipher as of a compressor
- Here clearly, we use the fact that the hash function does not require a key
- The Davis-Meyer compression strategy is thus the following:

$$c(H_i, M_{i+1}) = \mathbf{E}(M_{i+1}, H_i) \oplus H_i$$

where, in $\mathbf{E}(M_{i+1}, H_i)$, the message block M_{i+1} as the role of the key

The Davis-Meyer compression "strategy" (cnt.d)

- The reason for the final $\oplus H_i$ should be clear
- Without it, the chain could be easily inverted using the decryption function and (again) the message blocks as keys
- Many real hash functions use the Davis-Meyer compression

Algoritmi di Crittografia

MD5

- Developed by Ron Rivest!
- Hash values are 128 bit long
- Splits the message in blocks of 512 bits and pads the last block
- The internal state (128 bits) is regarded as four 32-bit words
- The compression function is based on four rounds that mix the state and the message blocks
- The “mixing” is quite complex, as it is composed of a number of basic operations (additions, boolean bitwise operations, rotations)

Insecurity of MD5

- 128 bits lead immediately to an attack based on the birthday paradox that only uses 2^{64} MD5 computations
- However, the compression function of MD5 admits easier to find collisions
- Although a collision in the compression function dows not imply necessarily hash vulnerability, it has been recently shown that attacks can be made which use much less than 2^{64} MD5 computations
- Summing up: MD5 should not be used

SHA-1

- 160 bit internal state (hence hash values)
- Combines Merkle and Damgård construction with a Davis-Meyer compression function
- The block cipher used to compress is a specially designed one
- Used the cipher, the internal state is modified as follows:

$$H = \mathbf{E}(M, H) + H$$

- Observe that the encryption does work on the whole message (not on single blocks), and that integer addition “replaces” the exclusive or
- Actually, the 160 bit internal state is regarded as five 32 bit words

Security issues

- Considered insecure
- Web servers using SHA-1 are marked as insecure by chrome (and chromium alike) web browsers
- Theoretical 80 bit security against birthday attacks recently reduced to 63 bits
- Real (and successful) attacks are known, though (two different pdf files with the same SHA-1 digest)
- Summing up: like MD5, SHA-1 should not be used (attacks cannot but improve...)

SHA-2 family

- Includes 2 pairs of algorithms: SHA-256 and SHA-224 from the one hand, and SHA-512 and SHA-384 from the other
- SHA-256 and SHA-224 works with 256 bit internal states
- SHA-224 simply truncates the last state (i.e., the hash value to be returned) to 224 bits but is otherwise identical to SHA-256
- Similarly, SHA-384 truncates the last state to 384 bits but is otherwise identical to SHA-512
- The reason for all these different hash lengths has to do with the fact that the SHA-2 family has been designed to be used with triple-DES (112 bit keys) as well as with AES (128, 192, or 256 bit keys)

Security issues

- As far as we know, the SHA-2 family members guarantee the security tied to the respective bit lengths: for instance, SHA-512 seems to guarantee 256 bit security against collision attacks
- Some concerns have been raised that have to with the fact that the structure of the SHA-2 family is really close to that of SHA-1 (which is deemed insecure)
- However, no theoretical nor practical attacks to any SHA-2 member has been reported yet
- The relative inefficiency of SHA-2 computations, is a limiting factor affecting (negatively) the efficiency of the attacks