Hash functions

Gianluca Dini
Dept. of Ingegneria dell'Informazione
University of Pisa
Emai: gianluca.dini@unipi.it

Version: 2021-04-18

An example

The input size is finite but arbitrary

```
Nel mezzo del cammin di nostra vita
mi ritrovai per una selva oscura
che' la diritta via era smarrita.

Ahi quanto a dir qual era e` cosa dura
esta selva selvaggia e aspra e forte
che nel pensier rinova la paura!
```

EASY

NON EASY

MD5

EXADECIMAL

`0xd94f329333386d5abef6475313755e94`

128 bit **The hash size is fixed, generally smaller than the message size**

apr. '21 Hash functions 2

An example with MD5.

Informal properties

- Applicable to messages of any size
- Output of fixed length (digest, hash value, fingerprint)
- No key (!)
- “Easy” to compute
- “Difficult” to invert
- “Unique“ (the hash of a message can be used to "uniquely" represent the message) ➔
  - The output should be highly sensitive to all inputs ➔
  - if we make minor modifications to the input, the output should look like very different

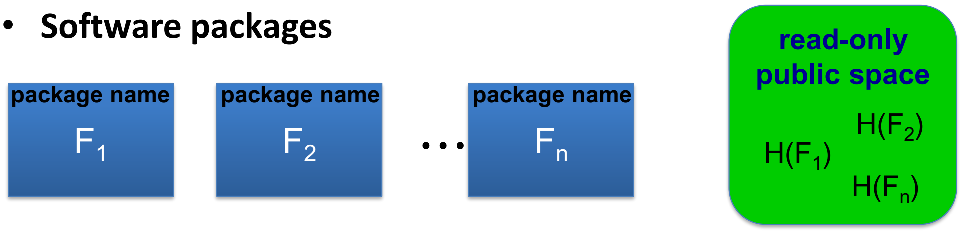
Informally, ah hash function should be efficient to compute, difficult to invert and unique. We shall see that talking about invertibility of hash functions is not appropriate. If the digest is unique, then it can be used to uniquely identify a message. For example, for performance reasons, it would be more efficient to digitally sign the digest instead of the whole message. Informally, in order to be unique, it is necessary that the digest is *highly* sensitive to *all* the input bits: if we make a minor modifications to the input bits, the output bit sequence should look like very different (like a block cipher)

Informal properties

- The fingerprint must be *highly* sensitive to *all* input bits
  - Input «I am not a crook»
    - Hash (MD5): 6d17fcd4ae0e82fa4409f4ea6f4106a6
  - Input «I am not a cook»
    - Hash (MD5): 9ebe3d42d5c01fc59fe3daacbf42f515
- https://www.fileformat.info/tool/hash.htm

Example: protecting files

- **Software packages**



- When user downloads package, can verify that contents are valid
  - H collision resistant ⇒
    attacker cannot modify package without detection
- No key needed (public verifiability), but requires read-only space

Linux distribution uses a solution like this.

Example: protecting files

Prelievo da WinRAR.it

- Se il prelievo non è ancora partito, clicca **qui** per scaricare la versione richiesta.
- **Oppure torna alla pagina dei prelievi file**.

Verifica Integrità del file appena prelevato (checksum)

Nome File: WinRAR-x64-600b1it.exe

Dimensione: 3.442 K

MD5: c11ac9a41e5d178e65417faa6dccf75f

SHA-1: c9a2e9ca312573aaaa7b0c16fd49cb3ce40bf54f

SHA-256: 07a60c7da09679960aa2e9e7335194506cff71caebf0be62b97069d8619221f6

Properties: collisions

QUALSIASI DIMENSIONE

- A hash function H: $\{0,1\}^* \rightarrow \{0,1\}^n$

FINITE NUMBER OF OUTPUT

INFINITO

OUTPUT

2^n

CONFLITTO!

(COLLISION)

- Properties
  - Compression: H maps an input *x* of arbitrary finite length into an output H(x) of fixed length *n*
  - Ease of computation: given *x*, H(x) must be "easy" to compute
  - Many-to-one: a hash function is many-to-one and thus implies collisions (pigeonhole principle)
  - (Def) A collision for H is a pair x_0, x_1 s.t. $H(x_0) = H(x_1)$ and $x_0 \neq x_1$

CONFIGURAZIONE DI n BIT

UNIQUE PROPERTY

NON POSSIBILE PER DEFINIZIONE.

BASTA CHE LE COLLISIONI SIANO DIFFICILI DA TROVARE.

apr. '21 Hash functions 7

Hash functions suffers from collisions by definition.

Security properties

- Preimage resistance (one-wayness)
  - For essentially all pre-specified outputs, it is *computationally infeasible* to find any input which hashes to that output
    - i.e., given an output y, to find x such that y = h(x) for which x is not known
- 2nd-preimage resistance (weak collision resistance)
  - it is computationally infeasible to find any second input which has the same output as any specified input
    - i.e., given *x*, to find x' ≠ x such that h(x) = h(x')
- Collision resistance (strong collision resistance)
  - it is computationally infeasible to find any two distinct inputs which hash to the same output,
    - i.e., find *x*, *x*' such that *h*(*x*) = *h*(*x*')

apr. '21 Hash functions 8

An hash function produces collisions. However, a hash function is secure if collisions are difficult to find. Notice that 2nd-preimage resistance and collision resistance are very different properties.

y: SEQUENZA DI m BIT

PROPRIETÀ:

1. DEVE ESSERE DIFFICILE TROVARE x TALE CHE $y = H(x)$
2. DATO x_1 e $y_1 = H(x_1)$, DEVE ESSERE DIFFICILE TROVARE $x_2 \neq x_1$ TALE CHE $H(x_2) = H(x_1)$
3. DEVE ESSERE DIFFICILE TROVARE x_1 E x_2 TALE CHE $H(x_1) = H(x_2)$

DIVERSI! NELLA 2 HO UN SOLO GRADO DI LIBERTÀ PERCHÉ x_1 È FISSATO.

PIÙ FACILE DEL 2

Classification

- One-way hash function (OWHF)
  - Provides preimage resistance, 2-nd preimage resistance
  - OWHF is also called weak one-way hash function
- Collision resistant hash function (CRHF)
  - Provides 2-nd preimage resistance, collision resistance
  - CRHF is also called strong one-way hash function

Relationship between security properties

- FACT 1 - Collision resistance implies 2nd preimage resistance
- FACT 2 - Collision resistance does not imply preimage resistance
  - However, in practice, CRHF almost always has the additional property of preimage resistance

Attacking Hash Functions

- An attack is successful if it produces a collision (forgery)
- Types of forgery
  - Selective forgery: the adversary has complete, or partial, control over x
  - Existential forgery: the adversary has no control over x

Black box attacks

- Consider H as a black box
- Only consider the output bit length n
- Assume H approximates a random variable
  - Each output is equally likely for a random input (so weak collisions exist for all output values)

ATTACCHI :
- ANALYTICAL
- BLACK BOX → SI APPOGGIANO SOLO ALLA DIMENSIONE DELL'OUTPUT

Specific Black box Attacks

↓

- Guessing attack
  - find a 2nd pre-image
  - Running time: $O(2^n)$ hash ops

$x \to \boxed{H} \xrightarrow{n} y$

- Birthday attack:
  - find a collision
  - Running time: $O(2^{n/2})$ hash ops $O(\sqrt{2^n})$
- These attacks constitute a security upper bound
  - More efficient analytical attacks may exist (e.g., against MD5, SHA-1) → DEVO CONOSCERE L'ALGORITMO E LE SUE VULNERABILITÀ

Guessing attack

→

- Objective: to find a 2^{nd} pre-image
  - Given x_0, find $x_1 \neq x_0$ s.t. $H(x_0) = H(x_1)$
- The attack

```
int GuessingAttack(x0) {
    repeat
        x1 ← random(); // guessing
    until h(x0) == h(x1)
    return x1;
}
```

2^n NUMERO DI COLPI PREVISTI

$\boxed{P = \frac{1}{2^n}}$

apr. '21 Hash functions 14

Guessing attack

- Running time
  - Every step requires
    - 1 random number generation: efficient!
    - 1 hash function computation: efficient!
  - Constant and negligible data/storage complexity
  - Running time in the order of 2^n operations

The data/storage complexity is negligible because the guessing attack requires just one parameter (x0) and has two store two values (x0 and x1) for each loop.
As the output of the hash function can be assumed as a uniform random variable, the probability to obtain x0 for a given input (x1) is equal to $1/2^n$. So, in order to guess x1, we expect to run 2^n instances of the loop.

Birthday attack

- Intuition
  - Start with
    - x_1 = «Transfer $10 into Oscar's account»
    - x_2 = «Transfer $10.000 into Oscar's account»

  DUE DOCUMENTI "SIMILI"

  - Alter x_1 and x_2 at nonvisible locations so that semantics is unchanged
    - Spaces, tabs, return,…
  - Continue until $H(x_1) == H(x_2)$

Birthday attack

- Attack Algorithm
  1. Choose $N = 2^{n/2}$ random input messages $x_1, x_2, \ldots, x_N$ (distinct w.h.p.)
  2. For i := 1 to N compute $t_i = H(x_i)$ COMPUTE THE HASH $\boxed{2^{n/2} \text{ HASH}}$ DA CONSERVARE
  3. Look for a collision ($t_i = t_j$), $i \neq j$. If not found, go to step 1.
- Attack complexity
  - Running Time: $2^{n/2}$
  - Space: $2^{n/2}$

QUANTE ISTANZE DEL LOOP 1-3 DOBBIAMO FARE PRIMA DI TROVARE UNA COLLISIONE?

The data/storage complexity if given by $N = 2^{n/2}$. The time complexity is given by N hash computations times the expected number of times the algorithm loop 1-3 is performed. The expected number of times the algorithm loop is performed depends on the probability of finding a collision at step 3. This probability is about ½ and thus the loop is expected to be carried out two times. It follows that the running time is in the order of $N = 2^{n/2}$. The probability of finding a collision at step 3 is ½ by virtue of the *Birthday Paradox*.

How well will this algorithm work?
- I shall show that the algorithm requires just a few iterations, namely 2.
- Let's have a look to the Birthday paradox, first. See next slides.

Analysis
- Notice that here $B = 2^m$ and thus $B^{1/2} = 2^{m/2}$.
- All these tags T_1 to T_N are independent of one another.
- If we choose $2^{m/2}$ or 1.2 $2^{m/2}$ tags, the probability that the collision will exist is roughly one half. Each iteration is going to find a collision with probability one half, so we have to iterate about two times in expectation. And as a result the *running time* of this algorithm is basically $2^{m/2}$ evaluations of the hash function.
- Notice also this algorithm takes a lot of space but we're going to ignore the space issue and we're just going to focus on the running time.
- This says that if your hash function outputs *m*-bits outputs there will always be an

attack algorithm that runs in time $2^{m/2}$.

- So for example if we output 128-bit outputs Then a collision could be found in time 2^{64}, *which is not considered sufficiently secure*. This is why collision resistant hash functions generally are not going to output 128 bits but more.

Birthday paradox: intuition

UNIVERSITÀ DI PISA

- Problem #1.
  - In a room of t = 23 people, what is the probability that at least a person is born on 25 December?
    - Answer: 23/365 = 0.063
- Problem #2.
  - In a room of t = 23 people, what is the probability that at least 2 people have the same birthdate?
    - Answer: 0.507 $\sim \frac{1}{2}$

AT LEAST 2 PEOPLE HAVE THE SAME BIRTHDATE = P

Q = 1 − P = PROBABILITY OF THE COMPLEMENTARY EVENT

$$Q = \frac{\overset{365-1}{364}}{\underset{2\ \text{PERSONE}}{365}} \cdot \frac{\overset{365-2}{363}}{\underset{3\ \text{PERSONE}}{365}} \cdot \ldots \cdot \frac{365-(23-1)}{365}$$

apr. '21 Hash functions 18

Let's have first an intuition of the Birthday Paradox.
Consider the solution of Problem 1.

- P = 1/365 + … + 1/365 (23 times) = 0.063.

Consider now the solution of Problem 2.

- Let P be the probability we want to calculate.
- Let Q be the probability of the complementary event, Q = Pr[no two people have the same birth date]. Then, Q = 1 – P.
- Let's compute Q. Q = (364/365) × (363/365) ×… × (343/365) = 0.493.
- Then, P = 0.507

The probability of finding two people who were born in the same day is much greater than finding a person who was born in a specific date.
Problem 1 can be connected to finding a 2nd-preimage whereas Problem 2 can be connected to find a collision. The intuition is that finding a collision is simpler than finding a 2nd-preimage.

Birthday attack

- Apply the birthday paradox to hash function
  - We have 2^n elements (not 365)
  - t inputs: $x_1, x_2, \dots, x_t$

Q = Probability of no collision: $\left(1-\frac{1}{2^n}\right)\left(1-\frac{2}{2^n}\right)\cdots\left(1-\frac{t-1}{2^n}\right) = \prod_{i=1}^{t-1}\left(1-\frac{i}{2^n}\right) \approx \prod_{i=1}^{t-1} e^{-\frac{i}{2^n}} = e^{-\frac{1+2+\cdots+t-1}{2^n}} \approx e^{-\frac{t(t-1)}{2^{n+1}}} \cong e^{-\frac{t^2}{2^{n+1}}}$

  - Probability of collision λ = 1 – P(no collision)
  - Solve in t, $t \approx 2^{(n+1)/2}\sqrt{\ln\left(\frac{1}{1-\lambda}\right)}$

$l = 0.5$ $t \approx 1.2 \times 2^{n/2}$

apr. '21 Hash functions 19

Let us now apply the Birthday Paradox to the Birthday Attack to compute the number of iterations of the loop. In practice we wish to compute the probability of collision for a given number t of inputs. Then we solve in t.
In the computation we employ the following simplifications:

1. $e^{-x} \approx 1 - x$
2. $1+2+\ldots+t-1 = t\cdot(t-1)/2$
3. $t(t-1) \approx t^2$ for $t >> 1$

Birthday attack

↓

UNIVERSITÀ DI PISA

- In practice,
  - The number of messages we need to hash to find a collision is in the order of the square root of the number of possible output values, i.e., $\sqrt{2^n} = 2^{n/2}$
- For example
  - n = 80 bit
  - $\lambda = 0.5$
  - $t \approx 2^{40.2}$ (doable with current laptops)
- Notice
  - The probability of collision λ does not influence the attack complexity very much

Hash functions

HOW TO BUILD HASH FUNCTIONS

Types of hash functions

- Dedicated hash functions
- Block cipher-based hash functions

BLOCK CIPHERS SONO MOLTO VERSATILI

1. ENCRYPTION
2. STREAM CIPHERS
3. PRNG
4. HASH FUNCTIONS

POSSO RIUSARE I COMPONENTI.

apr. '21 Hash functions 22

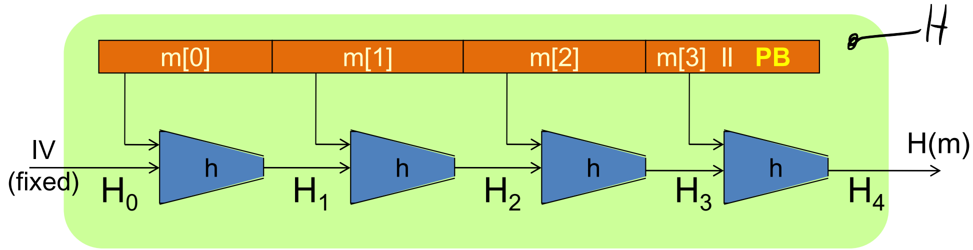
How to build a hash function

UNIVERSITÀ DI PISA

- Approach
  - Given a CRHF for short messages, construct a CRHF for long messages
- Solution:
  - The $\underbrace{\text{Merkle-Damgard iterated construction}}_{\text{funzione iterativa}}$

The Merkle-Damgard iterated construction

$m \to \boxed{H} \to H(m)$



- Compression function h: T × X ⟶ T
  - H_i - chaining variables
- Padding block PB: 1000… || msg len
  - msg len on 64 bits
  - If no space for PB add another block

h = COLLISION - RESISTANT HASH FUNCTION PER MESSAGGI PICCOLI

Merkle-Damgard collision resistance

- **Theorem**. if compression function h is collision resistant then so is H.
- **Proof**
  - By contradiction
  - Collision on H => collision on h Q.E.D.
- Comment
  - To construct a CRHF, it suffices to construct a collision resistant compression function

The MD4 family

UNIVERSITÀ DI PISA

Algorithm		Output [bit]	Input [bit]	No. of rounds	Collisions found
MD5		128	512	64	yes
SHA-1		160	512	80	yes
SHA-2	**SHA-224**	224	512	64	no
	SHA-256	256	512	64	no
	SHA-384	384	1024	80	no
	SHA-512	512	1024	80	no

First Collision on SHA-1 (2017)

- CWI – Google team
- Forged PDF documents
- Running time
  - Over 9,223,372,036,854,775,808 SHA1 computations that took 6,500 years of CPU computation and 100 years of GPU computations
    - 10^5 times faster than black box attack

https://www.cwi.nl/news/2017/cwi-and-google-announce-first-collision-for-industry-security-standard-sha-1

Sample hash functions

UNIVERSITÀ DI PISA

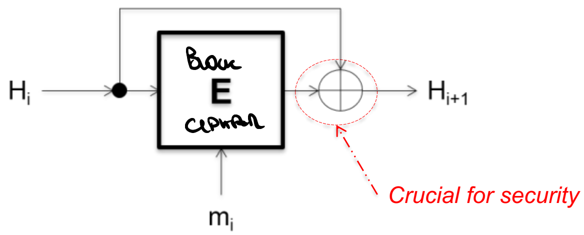
Hash Function	*m*	Preimage	Collision
MD5	128	2^{128}	2^{64}
RIPEMD-128	128	2^{128}	2^{64}
SHA-1	160	2^{160}	2^{80}
RIPEMD-160	160	2^{160}	2^{80}
SHA-256	256	2^{256}	2^{128}
SHA-512	512	2^{512}	2^{256}

Hash functions from block ciphers

- Use block cipher chaining techniques
  - Matyas-Meyer-Oseas
  - Davies-Meyer
  - Miyaguchi-Preneel
  - Use block ciphers with 192/256 bit blocks
    - E.g. AES
- Cons
  - (digest size = block size) may be not enough for collision resistance
  - Possible solutions
    - Use block cipher with larger blocks (AES-192, AES-256)
    - Hirose scheme: use several instances of the block cipher

Davies-Meyer (compression function)

- Finding a collision h(H, m) = h(H’,m’) requires $2^{m/2}$ evaluations of (E, D) ⇒ best possible!



apr. ’21 Hash functions 30

The message is used as the key of the cipher. Notice that the XOR operation is crucial for security. Actually, if we remove the XOR, the compression function is not collision resistant anymore. This can be proven as follows. Let us remove the XOR and thus let h(H, m) = E(m, H). It follows that constructing a collision becomes simple. In order to construct a collision we have to determine two pairs (H, m) and (H’, m’) that produce the same output. We may proceed as follows:

1. We choose a random triple (H, m, m’) and construct H’ such that E(m, H) = E(m’, H’).
2. Now, H’ can be easily computed by decrypting both sides using m’ as a key: H’ = D(m’, E(m, H))

All SHA-* use the D-M compression function. In particular, SHA-256 uses the SHACAL-2 block cipher.

Exercise

- Problem
  - If we remove the xor, the compression function is not collision resistant anymore.
  - Proof (by contradiction)
    - Remove the xor ➔ h(H, m) = E(m, H)
    - To construct a collision (H, m) and (H’, m’) is easy
      - Choose a random triple (H, m, m’)
      - Determine H’ such that E(m, H) = E(m’, H’) ➔ H’ = D(m’, E(m, H)

Q.E.D.

Hash functions

USES OF HASH FUNCTIONS

Uses of hash functions

UNIVERSITÀ DI PISA

- Digital signatures
  - Requires strong collision resistance
- Password storage
  - Requires weak collision resistance
- Authentication
  - Requires weak collision resistance

Hash Functions

AUTHENTICATION

Integrity vs authentication

- Message integrity
  - The property whereby data has not been altered in an unauthorized manner since the time it was created, transmitted, or stored by an authorized source
- Message origin authentication
  - A type of authentication whereby a party is corroborated as the (original) source of specified data created at some time in the past
- Data origin authentication $\overset{\text{VERA!}}{\Longrightarrow}$ data integrity

  CORRELATE

apr. '21 Hash functions 35

INTEGRITY = ABILITÀ DI INDIVIDUARE CAMBIAMENTI NEI MESSAGGI

AUTHENTICATION = PROVA CHE IL MESSAGGIO PROVIENE EFFETTIVAMENTE DAL SENDER.

Use of hash functions for authentication

- The purpose of a hash functions, *in conjunction with other mechanisms* (authentic channel, encryption, digital signature), is to provide message integrity and authentication

Example #1
Alice takes the hash of a file.
Alice takes the digest of the file
Alice sends the bundle file + digest to Bob by email.
Mr Lou Cipher intercept the email, changes the file, changes the hash and forwards the bundle file' + digest' to Bob

Example #2
Alice takes the hash of a file.
Alice sends the file by email.
Alice reads the hash to Bob over the phone (*physically authentic channel*)

Authentic channel

- Alice → Bob
  - Let t = H(x)

x, t

- MIM attack

MIM

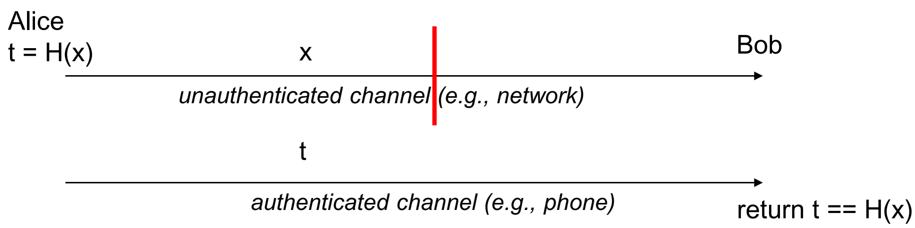
x, t | x', t'

t' = H(x')

Authentic channel

HASH FUNCTIONS NON POSSONO ESSERE USATE DA SOLE

- Alice
  - Computes t = H(x)
  - Sends x to Bob through the network
  - Reads t to Bob over the phone
    - An additional channel considered authenticated by assumption



Hash functions with block ciphers

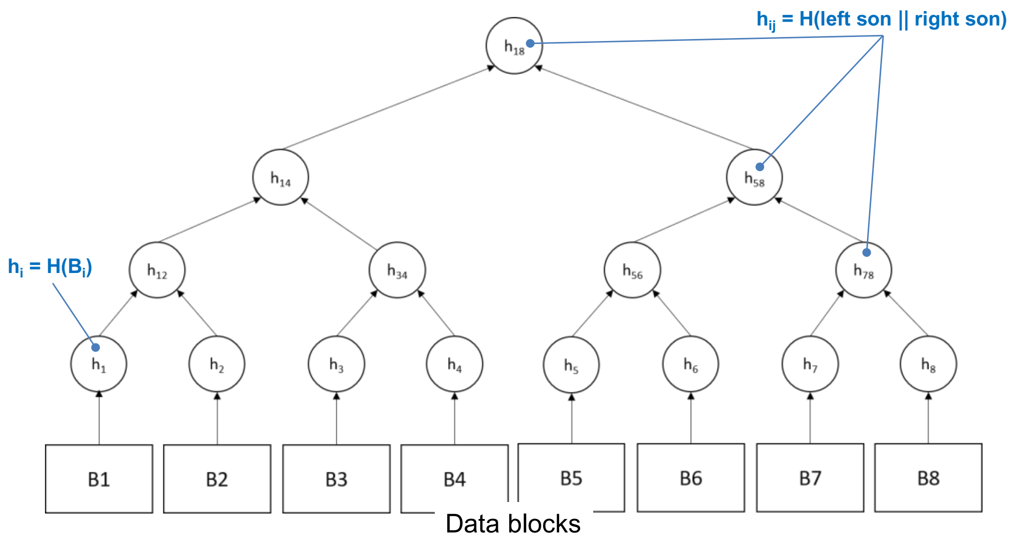
SICUREZZA $\underbrace{E_k}$ DIPENDE DAL CIPHERS

- $E_k(x||H(x))$ recommended
  - Confidentiality and integrity
  - As secure as E
  - H has weaker properties than digital signatures
- x, $E_k(H(x))$ SOSTITUISCO $x_1 : H(x_1) = H(x)$ not recommended
  - Prove that sender has seen H(x)
  - H must be collision resistant
  - Key k must be used only for this integrity function
- $E_k(x)$, $H(x)$ not recommended
  - H(x) can be used to check guesses on x
  - H must be collision resistant

Hash functions

MERKLE TREE (no)

Merkle Tree (1979)



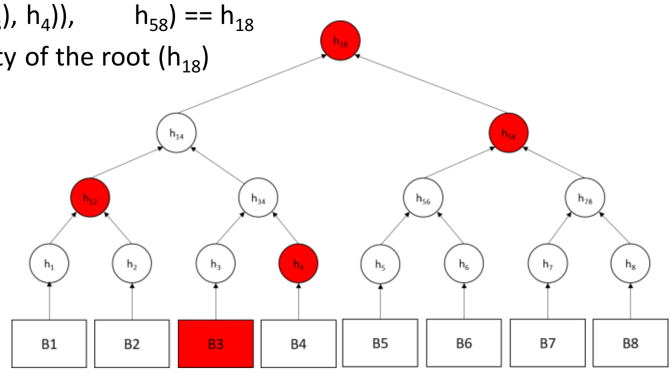
Merkle tree - properties

UNIVERSITÀ DI PISA

- MT (or hash tree) allows efficient and secure verification of the contents of large data structures
- The root is digitally signed or securely store
- Verifying whether a leaf node is part of the MT requires computing a #hashes proportional to the logarithm of the #leaves

Merkle Tree - verification

- Proof that B3 belongs to the data set
  - List of hashes:
    $<h_4, h_{12}, h_{58}, h_{18}>$
  - Check whether
    $H(H(h_{12}, H(H(B_3), h_4)), \quad h_{58}) == h_{18}$
  - Verify authenticity of the root (h_{18})



Merkle Tree - applications

- File systems
  - IPFS, Btrfs, ZFS
- Content distribution protocols
  - Dat, Apache Wave
- Distributed revision control system
  - Git, Mercurial
- Backup Systems
  - Zeronet
- P2P networks
  - Bitcoin, Ethereum
- NoSQL systems
  - Apache Cassandra, Riak, Dynamo
- Certificate Transparency framework

Hash functions

ONE-TIME PASSWORD

One-Time Password

- One-Time Password (OTP)
  - A password that is valid for only one login session or transaction
  - A.k.a. dynamic password, dynamic pin
- Pros
  - Not vulnerable to replay attack
  - Not vulnerable to password-reuse attack
- Cons
  - Hard to remember, so you need additional technology

One-Time Password

- Methods of generating OTP
  - Based on time-synchronization
  - Based on the previous password
  - Based on a challenge

One-Time Passwords

- Time Synchronization
  - Prover
    - Token, $clock_p$
  - Verifier:
    - Authentication server, $clock_v$
  - Problems
    - Clocks of prover and verifier are roughly synchronysed
    - Network latency, user delay, clock skews

One-Time Passwords

- Time Synchronization
  - Times
    - T0 = initial time
    - T = current time
    - X = time steps in a second
    - C = no. off time-steps between T0 and T
    - C = (T – T0)/X
    - W = acceptance window
  - Key
    - Key k shared between prover and verifier

OTP – time synchronization

- The protocol
  - Prover Authenticator
    - T_p <- $clock_p()$
    - $C_p = (Ta - T_0)/X$
    - HOTP = $HMAC_k(C_p)$

------------------------HOTP------------------------------------>

```
Tv <- clockv()
for all t in [Tv – W/2, Tv + W/2] {
    Cv = (t – T0)/X;
    if (HOTP == Hk(Cv)
        return TRUE;
}
return FALSE
```

< ----------------------------TRUE|FALSE---------------------------

One Time Password

- For more details
  - D. M'Raihi, S. Machani, M. Pei, J. Rydell. TOTP: Time-Based One-Time Password Algorithm, RFC 6238, IETF, May 2011

One Time Password

- Hash List (Lamport's scheme)
  - Setup
    - Seed p_0 <- random
    - $p_i = H(p_{i-1})$, i = 1, …, n
    - p_n is stored at the verifier
  - Password verification
    - Prover sends p_{n-1} to Verifier
    - Verifier returns ($p_n == H(p_{n-1})$)
    - *More in general*
    - Verifier returns ($p_i == H(p_{i-1})$) or ($p_i == H^i(p_0)$)
      - 2nd form in case p_i are not verified sequentially

One-Time Password

- Challenge-Response
  - Prover and Verifier share a key K
  - Verifier Prover

```
ch <- random()
send(Prover, ch)
      ------------------------------------------------------->
                                                   res = H_k(ch)
                                                   send(Verifier, res)
      < -------------------------------------------------------
return (res == H_k(ch))
```

Hash function

PASSWORD STORAGE

Storage of password

- Passwords are stored in hashed form
  - <username, hash>
- Example
  - alice 4420d1918bbcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b
  - jason 695ddccd984217fe8d79858dc485b67d66489145afa78e8b27c1451b27cc7a2b
  - mario cd5cb49b8b62fb8dca38ff2503798eae71bfb87b0ce3210cf0acac43a3f2883c
  - teresa 73fb51a0c9be7d988355706b18374e775b18707a8a03f7a61198eefc64b409e8
  - bob 4420d1918bbcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b
  - mike 4b529ac375b4217be17fef1a4a6f1624185cc99909e92278c0759e12ab3d61fa

apr. '21 Hash functions 55

Hash computed by means of SHA-256.

Storage of password

UNIVERSITÀ DI PISA

- Criticalities
  - If different users choose the same password, they have the same hash
    - Example: Alice and Bob
  - Dictionary attack (brute force attack)
    - E.g.: https://www.onlinehashcrack.com/
  - Rainbow table attack
    - Pre-computed database of hashes for fast access
      - Trade storage for computation
    - E.g. https://crackstation.net/
      - E.g.: Mike / “friendship”

Notice that CrackStation is able to also spot heuristics such as Fr13ndsh1p

Salting password →

- Salt
  - A fixed-length cryptographically-strong random value that is added to the input of hash functions to create unique hashes for every input, regardless of the input not being unique.
  - A salt makes a hash function look non-deterministic, which is good as we don't want to reveal password duplications through our hashing.

Salting password

- Salting a password
  - Upon creation of a new password pwd
  - Define salt = random()
  - Compute hash = H(salt|pwd)
  - Store <username, salt, hash>
- Advantages
  - Salting makes a Rainbow Table Attack infeasible
  - If stored elsewhere than hash, salt also makes a Dictionary attack infeasible

Salting password

→

- Example
  - Alice
    - Password: admin
    - salt: 317029;
    - hash: f9ea5ab02d83138e4f0f1f87ffd2c62a
  - Bob
    - Password: admin
    - salt: 450982
    - hash: 8c13e26985d3972bff4063861194c98c