P2P Systems and Blockchains

Spring 2021,
instructor: Laura Ricci
laura.ricci@unipi.it

Lesson 7:
CRYPTOGRAPHIC TOOLBOX FOR DHT AND BLOCKCHAINS

10/3/2021
OUTLINE OF THE NEXT LESSONS

- introduction of several tools and data structures for the development of
  - Distributed Hash Tables (DHT)
  - blockchains
  - distributed ledgers
- cryptographic tools
  - cryptographic hash functions
  - digital signatures
- data structures
  - Bloom filters
  - Merkle trees
  - Patricia tries
  - Patricia Merkle tries
CRYPTOGRAPHIC HASH FUNCTIONS

\[ \mathcal{H} : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda \]

- SHA256
- RIPEMD160

Arbitrarily long data \rightarrow Fixed sized hash/digest

- referred also one-way transformations
- properties:
  - take any byte sequence as input
  - fixed size output
  - efficiently computable
  - some security properties (to be seen in the next slides)
- family of hash functions share similar design with different parameters and output length
- related terms: hash, message digest, fingerprint, checksum
CRYPTOGRAPHIC HASH FUNCTIONS

- **input:**
  - input length is counted in bits
  - normally a maximum input length, zero input length is permitted,
- **output:**
  - fixed length output: normally 128/160/256/512-bit

---

Example:

1. (null) → h → a4a6cb8b60695d71 8a902afaba4c2765
2. Hello, world → h → 661dce0da2bcb2d8 2884e0162ac8194
3. This is a clear text that can easily read without using the key. The sentence is longer than the text above. → h → 52f21cf7c7034a20 17a21e17e061a863

*Fixed length Digest:*L
CRYPTOGRAPHIC HASH FUNCTIONS

- property:
  - a small change in the input produces a completely different output

“There was of course no way of knowing whether you were being watched at any given moment. How often, or on what system, the Thought Police plugged in on any individual wire was guesswork. It was even conceivable that they watched everybody all the time. But at any rate they could plug in your wire whenever they wanted to. You had to live – did live, from habit that became instinct – in the assumption that every sound you made was overheard, and, except in darkness, every movement scrutinized.”

“There was of course no way of knowing whether you were being watched at any given moment. How often, or on what system, the Thought Police plugged in on any individual wire was guesswork. It was even conceivable that they watched everybody all the time. But at any rate they could plug in your wire whenever they wanted to. You had to live – did live, from habit that became instinct – in the assumption that every noise you made was overheard, and, except in darkness, every movement scrutinized.”
CRYPTOGRAPHIC HASH FUNCTIONS

- classical security properties:
  - pre-image resistance
  - second pre-image resistance
  - collision-resistance
- also required (useful for cryptocurrencies and blockchains)
  - hiding
  - puzzle-friendliness
HASH FUNCTIONS: SECURITY PROPERTIES

Let $X$ be the domain and $Y$ the codomain of the hash function $h$:

- **preimage resistance**: for any $y \in Y$, it is hard to find $x \in X$ such that $h(x) = y$
- one-way function
- may be also replaced by the hiding property (to be seen later)
THE PIGEONHOLE PRINCIPLE

- if n items are put into m containers, with n > m, then at least one container must contain more than one item
- very simple, but used to demonstrate possibly unexpected results
  - 51 numbers are chosen from the integers between 1 and 100.
  - prove that 2 of the chosen integer are consecutive
  - proof:
    - pigeonholes: pair of consecutive integers {1,2}, {3,4},...{99,100}
    - pigeon: 51 chosen numbers
    - by the pigeonhole principle, there will be at least 1 pigeonhole containing 2 pigeons
**CRYPTOGRAPHIC HASH: COLLISION RESISTANCE**

![Diagram of cryptographic hash function](image)

- since the codomain is smaller than the domain, due to the pigeon principle, collisions exist!
- but it is very hard to find them....
- nobody can find $x$ and $y$ such that $x \neq y$ and $H(x) = H(y)$
HASH FUNCTIONS: SECURITY PROPERTIES

Let $X$ be the domain and $Y$ the codomain of the hash function:

- **collision resistance**: it is hard to find a pair of values $x_1$ and $x_2$, $x_1$ different from $x_2$, such that $H(x_1) = H(x_2)$

- also called **strong collision resistance**
Let $X$ be the domain and $Y$ the codomain of the hash function:

- **second preimage resistance**: given $M$ and thus $h = H(M)$, it is hard to find another value $M'$ that $H(M') = h$

- also called **weak collision resistance**

- may require exhaustive search looking for $M'$
SECOND PREIMAGE RESISTANCE: AN EXAMPLE

- A function which is not second preimage resistant: 8-bit block parity

\[ m = 1101001010001001111001010001010010001000101 \]

\[
\begin{align*}
b_1 &= 11010010 \\
b_2 &= 10001001 \\
b_3 &= 11100101 \\
b_4 &= 00010100 \\
b_5 &= 10100010 \\
b_6 &= 00010100 \\
digest &= 00011100 \ (\text{column-wise } \oplus)
\end{align*}
\]

- A simple way to find another message with the same hash:
  - Invert any even number of bits in \( m \) that are in the same column and the parity will not change

\[
\begin{align*}
m_1 &= 11010010 \\
    &\quad 10001001 \\
    &\quad 11100101 \\
    &\quad 00010100 \\
    &\quad 10100010 \\
    &\quad 00010100 \\
\end{align*}
\quad \begin{align*}
m_2 &= 11110010 \\
    &\quad 10001101 \\
    &\quad 11000101 \\
    &\quad 00110000 \\
    &\quad 10100010 \\
    &\quad 00110100 \\
\end{align*}
\quad \text{digest}(m_2) = 00011100
\]

- Hash which is not second preimage resistant
ATTACKING HASH FUNCTIONS

approaches for attacking an hash function:

• cryptanalysis involves exploiting logical weaknesses in the algorithm

• performing a brute-force attack
  • in cryptography, a brute-force attack, or exhaustive search might be used when it is not possible to take advantage of other weaknesses in the hashing system.
  • it consists of systematically checking all possibilities until the correct one is found.
SECURITY OF HASH FUNCTIONS

- the strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm.
- for a hash code of length n, the level of effort required is proportional to the following.

<table>
<thead>
<tr>
<th>Preimage resistant</th>
<th>$2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second preimage resistant</td>
<td>$2^n$</td>
</tr>
<tr>
<td>Collision resistant</td>
<td>$2^{n/2}$</td>
</tr>
</tbody>
</table>

- 56 bit hash can be brute-forced using $2^{56}$ operations: a security level of 56 bits.
- $2^{128}$ operations is infeasible by current machine, $2^{80}$ becomes feasible.
- note, $2^{81}$ operations take twice the time of $2^{80}$
FINDING COLLISIONS

- what is the maximum number of guesses to certainly find a collision by a brute force attack?

\[
\begin{align*}
\text{512 input bits} & \quad H
\end{align*}
\]

\[
\begin{align*}
H(m) & \quad \text{256 output bits}
\end{align*}
\]

- brute force:
  - pick \(2^{256} + 1\) distinct values in the domain
  - compute the hashes of each of them, and check if any two outputs are equal
  - a collision will be found by the "pigeon principle"
  - the maximum number of guesses required to certainly find a collision is
    \[
    O(2^n) \text{ time complexity}
    \]
    \[
    O(1) \text{ space complexity,}
    \]
    where \(n = \text{len}(H)\)
FINDING COLLISIONS

• what is the maximum number of guesses required to certainly find a collision by brute force?

512 input bits

$H$

256 output bits

$H(m)$

• pick random inputs and compute their hash values

• it is possible to show that you will find a collision with high probability long before examining $2^{256} + 1$ values

• why?
FINDING COLLISIONS

- what is the maximum number of guesses required to certainly find a collision by brute force?

512 input bits

\[ H \]

256 output bits

\[ H(m) \]

- \( \approx 50\% \) probability of a collision after \( \approx 2^{128} \)
- randomly choose just \( 2^{128} + 1 \) inputs, roughly the square root of the number of possible outputs: there is a high chance that at least two of them are going to collide.

\[ O(2^{n/2}) \] time complexity

\[ O(2^{n/2}) \] space complexity,

where \( n = \text{len}(H) \)

- Why just \( 2^{128} \)? See the birthday paradox in the next slides.
THE BIRTHDAY PARADOX

- how many people (n) should be in a room so that the probability that two of them share a birthday becomes larger than 50%?
- hypothesis:
  - a year is made of 365 days (no leap years)
  - all days are equally probable
- result: if n = 23, then two people will share a birthday with a probability just above 50%.
THE BIRTHDAY PARADOX

- when the second person enters the room, to not match it must avoid the birthday of the first person
- there are 364 possible birthdays to do that

\[
\frac{364}{365} = 0.9973
\]
THE BIRTHDAY PARADOX

- when the third person enters the room, to not match it must avoid two birthdays
- there are 363 possible birthdays to do that
- to get the probability that none of them share a birthday, we multiply it by the results of the previous stage

\[0.9973 \times \frac{363}{365} = 0.9918\]
THE BIRTHDAY PARADOX

- when we have 9 people in the room, the probability that everyone avoids each other birthday has dropped to 90%
- the opposite condition: the probability that at least a pair share a common birthday has nearly risen to 10%
THE BIRTHDAY PARADOX

- when the 23\textsuperscript{th} person enters the room, the chance of each one having a unique birthday has dropped to 0.49 %
- this means that the chance that at least two people share a birthday is now above 50%
THE BIRTHDAY PARADOX

![Graph of the Birthday Paradox](image)
CRYPTOGRAPHIC HASH: COLLISION RESISTANCE

• connection birthday paradox-hashing functions
  • a hash function $H$, with $n$ possible outputs
  • if $H$ is applied to $k$ random inputs, what must be the value of $k$ so that the probability that at least a pair of input $x$, $y$ satisfy $H(y) = H(x)$ is 0.5?

• for a hash function with a 256-bit output
  • about $2^{128}$ times, on average (square radix)
  • for the birthday paradox, brute force attack needs less trials, but luckily, the number of trials is always too high!
this works no matter what H is, but it takes too long to matter

- if a computer calculates 10,000 hashes/sec, it would take $10^{27}$ years to compute $2^{128}$ hashes

- *if every computer ever made by humanity was computing since the beginning of the entire universe, up to now, the probability that they would have found a collision is still infinitesimally small. [narayanan2016bitcoin]*
SECURITY OF HASH FUNCTIONS

- if no design flaws exist, the security of a hash function depends on the bit length of the output hash value.

- given a m-bit hash function, the attacker needs $2^{m/2}$ brute force computation to find a collision.
  - MD5 is $128/2 = 64$ bits security
  - SHA-1 is $160/2 = 80$ bits security
  - SHA-256 is $256/2 = 128$ bits security
  - SHA-512 is $512/2 = 256$ bits security

- at least 80 bits is required, to assure security

- Bitcoin’s blockchain uses SHA-256 (Secure Hash Algorithm).
REAL LIFE HASH FUNCTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Output Length (bits)</th>
<th>Security status</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>128</td>
<td>Collisions found</td>
</tr>
<tr>
<td>SHA1</td>
<td>160</td>
<td>Can be broken in $\sim 2^{61}$ iterations</td>
</tr>
<tr>
<td>SHA2</td>
<td>224-512</td>
<td>No known attacks</td>
</tr>
<tr>
<td>$\rightarrow$ SHA-256</td>
<td>256</td>
<td>No known attacks</td>
</tr>
<tr>
<td>SHA3</td>
<td>224-512</td>
<td>No known attacks</td>
</tr>
</tbody>
</table>

Bitcoin typically uses SHA-256(SHA-256(transaction))

- Retired:
  - SHA1, MD2 (output: 128 bit, Rivest), and MD4 (output: 128 bit, Rivest), vulnerable
  - MD5 (output: 128 bit, Rivest)
    - vulnerable, but ok for a large set of applications

- Current:
  - SHA2, SHA3, available for 224,256,384,512 bits fingerprints.
  - ripemd160 (output: 160 bit) also used in Bitcoin

Play with hash functions: https://www.pelock.com/products/hash-calculator
HASH FUNCTIONS LIFE CYCLE

New function proposed

Security evaluated

Attacks improved and are practical

Theoretical attacks proposed

Function standardized
 HASH FUNCTIONS LIFE CYCLE

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Snefru</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MD2 (128-bit)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MD5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RIPEMD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HAVAL-128[1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA-2 family</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA-3 (Keccak)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Key**
- Didn't exist/not public
- Under peer review
- Considered strong
- Minor weakness
- Weakened
- Broken
- Collision found

- historically, popular cryptographic hash functions have a useful lifetime of around 10 years
- the last one SHA-3 (Keccak)
Consider the outcome of a coin toss:
- Encode it as $HEADS=0$ or $TAIL=1$
- Then hash the output: $H_{toss}(outcome) = (1 + outcome) \mod 2$
  - $H_{toss}(HEADS)=1$, $H_{toss}(TAIL)=0$. 
COIN TOSS: AN ATTACK

- a possible attack: the attacker is able to find the domain \{HEADS, TAILS\}
  - when it sees a hash value of 1, it might try hashing each value in the set
  - it may easily find out that the outcome of the toss is HEADS
- the main problem:
  - the input domain is easily enumerable
  - hiding is related to the pre-image property
CHRYPTOGRAPHIC HASH: HIDING

a solution:

- pick a random integer $R$ for instance of 256 bits, from a distribution with a high min entropy, i.e. all the values in the distribution are negligibly likely, and no particular value is more likely than others
- append $R$ to the original input
- the input space becomes extremely hard to enumerate ($2^{257}$ possibilities).
- given a hash value, it is extremely hard for an attacker without the knowledge of the random key $R$ to deduce the outcome of the toss.

Hiding property

- a hash function $H$ is said to be hiding if when a secret value $R$ is chosen from a probability distribution that has high min-entropy, then, given $H(R || x)$, it is infeasible to find $x$. 
COMMITMENT: MOTIVATION

- Alice wishes to play with Bob paper-scissors-stone over the telephone (or internet)
  - no trusted third party
- rules of the game in the figure
- fairness: no one decides his/her choice after acquiring the other's choice
- if they can express their choices together, it may be a fair game.
  - difficult to obtain in a distributed system
- whoever goes first is going to lose the game!

if both choose the same item the game is declared a draw.
COMMITMENT SCHEME

- commit to a value and reveal it later
  - seal a value in an envelope and put that envelope out on the table where everyone can see it.
  - cannot change the value inside it, it remains secret now
  - open the envelope later and reveal the value

Commitments \{commit, open, verify\}

A prover $P$ hides a secret in the commit phase
and opens it to a verifier $V$ in the open phase.

<table>
<thead>
<tr>
<th>Commit phase</th>
<th>Open phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$ computes and sends $com$ to $V$:\n  $r = \text{random}()$\n  $com = \text{commit}(\text{secret}, r)$</td>
<td>$P$ opens secret and $r$ to $V$:\n  $V \text{ verify}(com, \text{secret}, r) = T \lor F$</td>
</tr>
<tr>
<td>Hiding $V$ cannot find clues of $(\text{secret}, r)$ from $com$ at the commit phase.</td>
<td>Binding $P$ cannot open $(\text{secret}', r') \neq (\text{secret}, r)$ such that \n  $T = \text{verify}(com, \text{secret}', r')$</td>
</tr>
</tbody>
</table>
PAPER-SCISSOR-STONE: COMMITMENT

• a commitment scheme may solve the problem of paper-scissor-stone

• the party who goes first, ‘commits’ to his/her choice
  • the other party cannot determine what was committed to
  • the other party can then verify that the revealing party has not altered its choice between the commitment and the revealing stage

• implementation through hash functions (A:Alice, B:Bob)

  \[ A \rightarrow B : h_A = H(RA || \text{paper}) \]

  \[ B \rightarrow A : \text{scissors} \]

  \[ A \rightarrow B : RA, \text{paper} \]

• at the end of the protocol Bob needs to verify that the \( h_A \) sent by Alice is equal to \( H(RA || \text{paper}) \).
  • if the values agree Bob knows that Alice has not cheated
  • the result is that Alice loses the game since scissors cut paper
• Bob is not able to determine that Alice has committed to the value “paper”, since:
  • he does not know the random value of $RA$ used (hiding)
  • he is unable to invert the hash function, (pre-image resistance)
• as soon as Bob sends the value scissors to Alice, she knows she has lost but is unable to cheat
  • she would need to come up with a different value of $RA$, say $R0A$, which satisfies
    \[ H(RA \ || \ \text{paper}) = H(R0A \ || \ \text{stone}). \]
  • but this would mean that Alice could find collisions in the hash function
  • this does not happen if the hash function is second-preimage resistant
• this property of the commitment scheme is called binding
  • Alice cannot change her mind after the commitment procedure
Commitment example

How do we play ‘Rock Paper Scissor’ fairly without third party?

Provers believe the verifiers did not cheat if $T = \text{verify}(\text{com, secret, } r)$

At the commit phase, both verifiers get no information by the **Hiding** property.

At the open phase, both provers cannot change their minds by the **Binding** property.
COMMITMENT SCHEME

• sealing the envelope
  • com ← commit (secret, nonce)
  • publish com

• opening the envelope
  • publish (nonce, value); com has been previously published
  • anyone can use verify(.... ) to check validity

• API
  • com ← commit (value, nonce)
  • match ← verify(com, nonce, value)

• implementation
  • H(msg | nonce) ← commit(msg, nonce)
  • H(msg | nonce) = com ← verify(com, key, msg)
an hash/search puzzle consists of:

- a cryptographic hash function, \( H \)
- a random value, \( r \)
- a target set, \( S \)
- a solution of the puzzle is a value \( x \), such that:

\[
m = r || x
\]

\[
H(m) \in S
\]

based on partial pre-image attack:

- you have to find a part of the input
- such that the output belongs to a set (not a single value like in the pre-image attack

- Bitcoin Proof of Work (PoW) is based on a hash/search puzzle
Cryptographic Hash: Search Puzzles

512 input bits

256 output bits

\( m \)

\( S \)
512 input bits

\[ m \]

256 output bits

\[ H(m) \in S \]

- m is a valid puzzle solution
512 input bits

\[ m \]

\[ H \]

256 output bits

\[ S \]

\[ H(m) \notin S \]

- m is a no valid puzzle solution
the difficulty may be tuned by defining the size of $S$:

- if $S$ is large, the puzzle is less difficult
- in Bitcoin is defined by the number of leading zeros of SHA-256

512 input bits

\[ m \]

256 output bits

\[ S \]

\[ H(m) \in S \]
Puzzle-friendliness property: a hash function $H$ is said to be puzzle-friendly if

- for every possible $n$-bit output value $y$
- if $k$ is chosen from a distribution with high min-entropy
- then it is infeasible to find $x$ such that $H(k \parallel x) = y$ in time significantly less than $2^n$.

- Puzzle-friendly property implies that no solving strategy to solve a search puzzle is much better than trying exhaustively all the values of $x$. 
The Merkle-Damgård transform

- used to convert a fixed-length hash function to a hash function taking inputs of arbitrary length
- preserves collision resistance.
- design collision resistant compression functions
  - operating on short, fixed-length inputs
  - convert such compression functions into full-fledged hash functions
- adopted by most popular hashing functions
CRYPTOGRAPHIC HASH APPLICATIONS

Digital signatures

Bitcoin transaction ID

Deduplication

Password storage

Sign in

Stay signed in

Forgot password?
CRYPTOGRAPHIC HASH APPLICATIONS

- generate data fingerprinting
- digest: if we know $H(x) = H(y)$
  - then it’s safe to assume that $x = y$.
  - useful because the hash is small: need not to compare entire files
- e-Mule, for instance, exploited MD-5 to verify that two files are the same, even if they are described by different keywords
- file or message integrity: antitampering
- use the hash value as the checksum to check if the data is changed or modified.
- to recognize if a content $C$ is the same of a content $C_1$ that we saw before,
  - just remember the hash of $C_1$, hash is a proxy of $C_1$!
  - compute the hash of $C$ and compare with that of $C_1$
  - if the two hashes are equal, the content has not be tampered
CRYPTOGRAPHIC HASH APPLICATIONS

- a distributed hash table (DHT) is a class of a decentralized distributed system that provides a lookup service similar to a hash table: (key, value)
- pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key.
CRYPTOGRAPHIC HASH APPLICATIONS

- Bitcoin use block chain (hash chain) to store transaction ledger in a P2P (Peer-to-Peer) network
- tamper freeness property

![Diagram of hash chain](image)
CRYPTOGRAPHIC FUNCTIONS: RECAP

hash function:

• arbitrary size input
• fixed-size output
• efficiently computable

cryptographic hash function must have also some security properties:

• hiding:
• collision resistance
• puzzle friendliness

collision freedom and hiding can be violated trivially through brute force

• compute the hash of all possible values for pre-digest until you find one that produces the desired digest
• have to be rendered computationally infeasible by making sure that domain $X$ is very large
• what are the assumptions when you verify them?
• when you produce them?
DIGITAL SIGNATURES

- the second cryptographic primitive needed as building blocks for blockchains
- based on public-key algorithms

- public-key algorithms are asymmetric algorithms
  - based on the use of two different keys, instead of just one.
  - the two keys are called the **private** and **public key**
  - private key
    - must be know only by its owner.
  - public key:
    - known to everyone (it is public)
- relation between the keys:
  - what one key encrypts, the other one decrypts, and viceversa.
ASYMMETRIC ENCRYPTION: CONFIDENTIALITY

- send a confidential message protected with a public key

- “you” encrypt something with my public key, that you know, because it is public
- “I” need my private key to decrypt the message.
- what is encrypted with one key is decrypted with the other key using the same algorithm.
PUBLIC-KEY SYSTEMS ADVANTAGES

Advantage of public-key systems over symmetric algorithms:

• no need to agree on a common key for both the sender and the receiver.

• if someone wants to receive an encrypted message, the sender only needs to know the receiver's public key

• as long as the receiver keeps the private key secret, no one but the receiver will be able to decrypt the messages encrypted with the corresponding public key.
ASYMMETRIC ENCRYPTION: PROPERTIES

- is relatively easy to compute the public key from the private key
- it is very hard to compute the private key from the public key (which is the one everyone knows)
Digital signature: a piece of data which attached to a message

- can be used to find out if the message was tampered with during the conversation
- for instance, through the intervention of a malicious user
DIGITAL SIGNATURES: INTEGRITY

- using public-key cryptography as in the previous slide
  - ensures integrity, because we have a way of knowing if the message we received is exactly what was sent by the sender.
  - the message itself is sent unencrypted.

- does not guarantee data confidentiality, the message is sent in clear

- to guarantee confidentiality + data integrity apply both the previous schemes
  - the sender signs the document with its private key and encrypts the document with the public key of the receiver
  - the receiver decrypts the document with its private key and it applies to the resulting document the public key of the sender
  - if the result “makes sense” then return “ok”
PUBLIC-KEY SYSTEMS ADVANTAGES

• the solution in previous slides only guarantees 'weak authentication'
  • only the sender's public key can decrypt the digital signature, encrypted with the sender's private key.
  • this only guarantees
    • whoever sent the message has the private key corresponding to the public key we used to decrypt the digital signature.
    • but maybe the sender isn't really who he claims to be
      • he/she just someone impersonating the sender.

• 'weak authentication' is not sufficient?
  • you need digital certificates
  • certification authorities
API FOR DIGITAL SIGNATURES

\[(sk, pk) := \text{generateKeys}(\text{keysize})\]

- \(sk\): secret signing key
- \(pk\): public verification key

\[\text{sig} := \text{sign}(sk, \text{message}) \quad /*\text{cipher the message through the secret key and obtain the signature.}\]

\[\text{isValid} := \text{verify}(pk, \text{message}, \text{sig}) \quad /*\text{decipher the signature through the public key and compare the result with the message}\]

and the following property must hold:

\[\text{verify}(pk, \text{message}, \text{sign}(sk, \text{message})) = \text{true}\]
DIGITAL SIGNATURES

• Three algorithms (KeyGen, Sign, Verify)
  • KeyGen: takes as input the security parameter. returns the signing-key and verification-key.
  • Sign: takes as input the signing-key and the message to be signed and returns a signature.
  • Verify: takes as input the verification-key, a message and a signature on the message and returns either True or False.

• Major challenge:
  • what prevents the adversary from learning how to sign messages by analysing the verification-key?
DIGITAL SIGNATURES CONSTRUCTION

- Based on the RSA (Rivest Shamir Adleman), one way trapdoor function (with hardness that relates to the factoring problem).

- The RSA algorithm
  - based on the discrete-logarithm problem.
  - the DSA algorithm

- Bitcoin.
  - uses ECDSA, a DSA variant over elliptic curve groups.
  - typical Bitcoin transaction
    - input: contains a signature and public-key
    - output: contains the code (smart contract) for the verification procedure
PLAYING WITH JAVA AND SIGNATURES

```java
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import javax.crypto.Cipher;

public class SignatureTest {
    public static void main(String[] args) throws Exception {
        // generate public and private keys
        KeyPair keyPair = buildKeyPair();
        PublicKey pubKey = keyPair.getPublic();
        PrivateKey privateKey = keyPair.getPrivate();

        // encrypt the message
        byte[] encrypted = encrypt(privateKey, "This is a secret message");
        System.out.println(new String(encrypted)); // <<encrypted message>>

        // decrypt the message
        byte[] secret = decrypt(pubKey, encrypted);
        System.out.println(new String(secret)); // This is a secret message
    }
}
```
public static KeyPair buildKeyPair() throws NoSuchAlgorithmException {
    final int keySize = 2048;
    KeyPairGenerator keyPairGenerator =
        KeyPairGenerator.getInstance("RSA");
    keyPairGenerator.initialize(keySize);
    return keyPairGenerator.genKeyPair();
}

public static byte[] encrypt(PrivateKey privateKey, String message) throws Exception {
    Cipher cipher = Cipher.getInstance("RSA");
    cipher.init(Cipher.ENCRYPT_MODE, privateKey);
    return cipher.doFinal(message.getBytes());
}

public static byte[] decrypt(PublicKey publicKey, byte[] encrypted) throws Exception {
    Cipher cipher = Cipher.getInstance("RSA");
    cipher.init(Cipher.DECRYPT_MODE, publicKey);
    return cipher.doFinal(encrypted);
}
This is a secret message
- Encryption is two-way, and requires a key to encrypt/decrypt.

- Hashing is one-way. There is no 'de-hashing'.