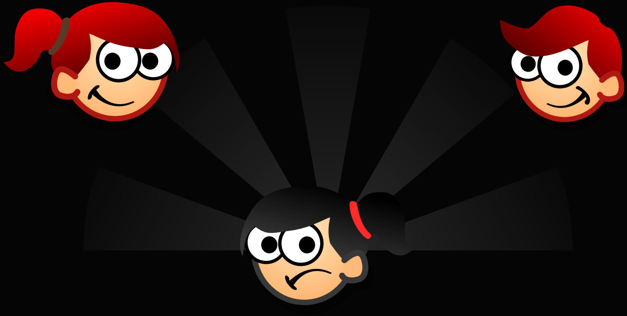
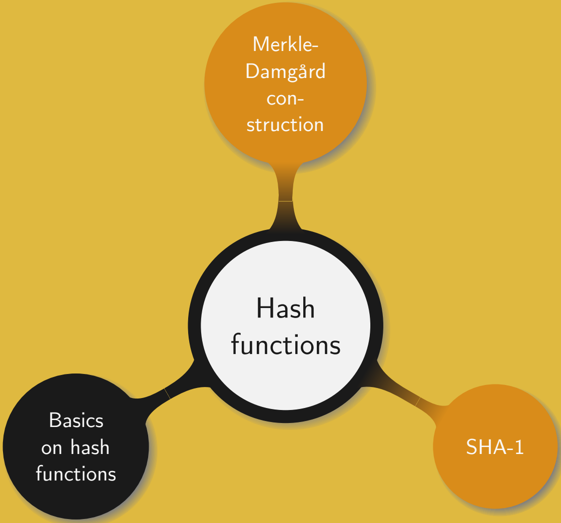


Introduction to Cryptography

4. Hash functions

Manuel – Summer 2021

Chapter organisation



Data integrity

Components of modern cryptography:

- Confidentiality
- Authentication
- Data integrity
- Non-repudiation

Common setup for data integrity:

- Insecure environment
- Unencrypted data
- Data must not be altered

Example.

- Files in an OS
- Software to be downloaded or installed from internet

Fingerprint

Simple high-level idea:

- Construct a short fingerprint of the data
- Store the fingerprint in a secure place
- Recompute and compare the fingerprint on a regular basis
  - Fingerprint is changed: data was altered
  - Fingerprint is unchanged: data was not altered

Construction goals:

- The fingerprint must be a few hundreds of bits long
- A tiny change in that data radically impacts the fingerprint
- It is impossible to alter the data without totally changing the finger-print

Hash function

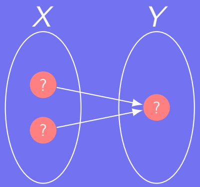
Definition (Hash function)

A *hash function* is a function h that verifies the following properties:

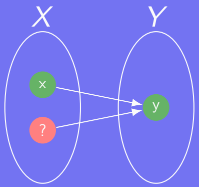
i. Efficiently computed for any input.
ii. *Pre-image resistant*: given y it is computationally infeasible to find x such that $h(x) = y$.
iii. *Second pre-image resistant*: given x, it is computationally infeasible to find $x' \neq x$ with $h(x) = h(x')$.
iv. *Collision resistant*: it is computationally infeasible to find x and x' with $x' \neq x$ and $h(x) = h(x')$.

The output of a hash function is called *message digest* or *digest*.

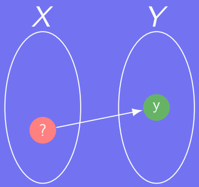
Remarks



Collision Resistance



2nd Pre-image Resistance



Pre-image Resistance

By definition a hash function is a one-way function since the definition of pre-image resistant is the same as the one of a one-way function.

The collision resistance is the most general of the three resistance properties. Roughly speaking collision resistance implies second pre-image resistance, which implies pre-image resistance.

Applications

Remark. A hash function takes inputs of any size and output values of fixed sizes, independently of the input.

Applications requiring the resistance properties:

- Pre-image resistant: password
- Second pre-image resistant: virus
- Collision resistance: signature

The DLP hash function

We present a first example of a hash function due to Chaum, van Heijst and Pfitzmann. Although it satisfies conditions (ii), (iii), and (iv) of a hash function, it is too slow to be used in practice.

Let p be a prime such that $q = \frac{p-1}{2}$ is also a prime and choose α and β two generators of $\mathrm{U}(\mathbb{Z}/p\mathbb{Z})$. We write any x in $\mathbb{Z}/q^2\mathbb{Z}$ as $x_0 + x_1 q$ with $0 \leq x_0, x_1 \leq q - 1$, and define h from $\mathbb{Z}/q^2\mathbb{Z}$ into $\mathrm{U}(\mathbb{Z}/p\mathbb{Z})$ by

$$h(x) = \alpha^{x_0}\beta^{x_1} \bmod p.$$

We will now show that h is collision resistant, by proving that finding a collision means solving the DLP. To prove this result we will need to recall a result on the number of solutions of a modular equation.

The DLP hash function

Proposition

If two values $x \neq x'$ with $h(x) = h(x')$ are known then the discrete logarithm of β in base α can be efficiently computed.

In order to prove this result we recall the following lemma.

Lemma

Let $a, b \in \mathbb{Z}$ and $m \in \mathbb{N} \setminus \{0\}$ and $d = \gcd(a, m)$. The linear congruence $ax \equiv b \bmod m$ has a solution if and only if $d \mid b$. In that case, it has d solutions that are mutually incongruent mod m.

The DLP hash function

Proof. Suppose $h(x) = h(x')$, with $x = x_0 + x_1 q$ and $x' = x'_0 + x'_1 q$. Since α is a generator of $\mathrm{U}(\mathbb{Z}/p\mathbb{Z})$, $\beta = \alpha^a$ for some integer a and

$$\alpha^{x_0 + a x_1} \equiv \alpha^{x'_0 + a x'_1} \bmod p.$$

From corollary 3.18, $a(x_1 - x'_1) \equiv x'_0 - x_0 \bmod (p-1)$. To solve this equation for a we see that if $x_1 = x'_1$ then $x_0 = x'_0$ and $x = x' \lightning$.

Therefore we assume $x_1 \neq x'_1$, and find $d = \gcd(x_1 - x'_1, p-1)$ incongruent solutions (lemma 4.9). But as $q = \frac{p-1}{2}$ is prime, the only factors of $p-1$ are $1, 2, q$, and $p-1$. Also note that $0 \leq x_1, x'_1 \leq q-1$ implies $-(q-1) \leq x_1 - x'_1 \leq q-1$. And since $x_1 - x'_1 \not\equiv 0 \bmod (p-1)$ it means that d is either 1 or 2.

Thus it suffices to test the two solutions to determine a. Hence finding $x \neq x'$ with $h(x) = h(x')$ implies solving the DLP. $\square$

The birthday paradox

As we have already mentioned the birthdays paradox on several occasions (2.27, 3.75) we now present some more details on this "birthday attack".

The essence of the birthday paradox can be expressed by considering the birthdays of 23 persons. We first assume the birthdays to be independent and equiprobable. If those 23 people all have a different birthday, it means that the second person has 364/365 chance of not sharing a birthday with the first one. Then for the third one the probability is 363/365 and so on until the twenty-third whose probability is 343/365. Therefore the probability of at least two sharing the same birthday is

$$1 - \prod_{i=1}^{22} \frac{365 - i}{365} = 0.507 > \frac{1}{2}.$$

The birthday paradox

Suppose we now have a large number of objects n, that are randomly chosen with replacement by two groups of r persons each. The probability of someone in the first group choosing the same object as someone in the second group can be approximated by $1 - e^{-r^2/n}$.[1] And the probability of i matches is $\left(\frac{r^2}{n}\right)^i \frac{e^{-r^2/n}}{i!}$.

As the probability of a match (i.e. two persons choosing the same object) is expected to be larger than 1/2 we set r^2/n to be $\ln 2$. This yields $r \approx 1.117\sqrt{n}$. Since for $a \ll n$, $a\sqrt{n}$ is of the same order of magnitude as $\sqrt{n}$, it means that a match will be found in average after $\mathcal{O}(\sqrt{n})$ persons have chosen an object.

We now investigate how to transform this observation into an effective cryptographic attack.

[1]This approximation will be derived in the homework.

The birthday attack

Let h be a hash function which digest are of length n. The first obvious strategy is to compute $h(x)$ for about $\mathcal{O}\left(\sqrt{n}\right)$ random x and hope for a collision, as the probability is larger than a half.

Example. Let h be a hash function whose output is 128 bits long. Then the above attack leads to a collision in $\mathcal{O}(2^{64})$ steps.

An important drawback in this attack is the amount of storage required, since all the values must be stored in order to be tested for collisions.

Example. What is the hardest: perform 2^{64} operations or store 2^{64} bytes?

An improved birthday attack

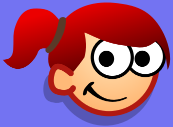
Following the Pollard's rho idea (3.75) it is possible to decrease the amount of storage necessary by computing and comparing two hash sequences.

Select a random initial x_0 and then compute $x_i = h(x_{i-1})$ and $x_{2i} = h(h(x_{2(i-1)}))$. At each step compare x_i and x_{2i}: a collision on x_{i-1} and $h(x_{2(i-1)})$ is found as soon as $x_i = x_{2i}$.

Note that we implicitly assumed h to act as a random oracle (2.21) such that the probability of having x_{i-1} equal to $h(x_{2(i-1)})$ is very low. In such case no information would be gained.

So far we focused on how to find collisions from a theoretical point of view, that is without considering whether or not the generated hash originates from a meaningful message.

A real life birthday attack



Alice is happy: she has received a nice contract for a new job in Eve's company.

Contract

```
Alice will work 16 hours
a week for a base salary
of 50,000 RMB a month.
Alice can take as much
paid holidays as  she
wants.
```

Contract

```
Alice will work 16 hours
a day for a base salary
of 500 RMB a  month;
Alice can take as little
paid holidays as Eve
wants.
```

Eve constructed the two contracts such that they have the same hash. Alice signed one but Eve can pretend it was the other one.

A real life birthday attack

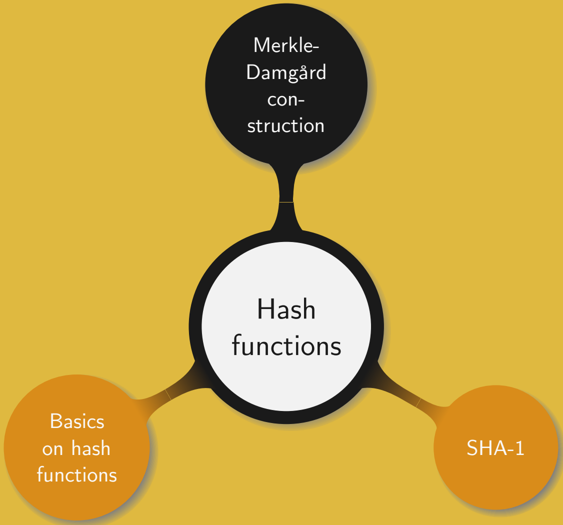
Why is it working?

- Eve generated many good and bad contracts
- She altered each base contract by
    - Changing the punctuation
    - Expressing the same idea using different synonyms
    - Adding extra spaces
- She computed their hash and found a collision

Example. How many ways are there to read this short paragraph?

I {hate|despise|detest} this {terrible|awful|horrible|disastrous} course. It is so {hard|complex|difficult|complicated} and the explanations are always {unclear|confused|doubtful}.

Chapter organisation



Designing hash functions

The goal is to design collision resistant hash functions

Difficulty:

- Number of possible input: infinite
- Number of possible output: finite

Conclusion: any hash function has an infinite number of collisions

Merkle-Damgård construction: methodology to convert a hash function on strings of fixed length into a hash function accepting arbitrary input lengths

We will prove that if the original hash function is collision resistant then so is the constructed one.

Iterated hash functions

Definitions

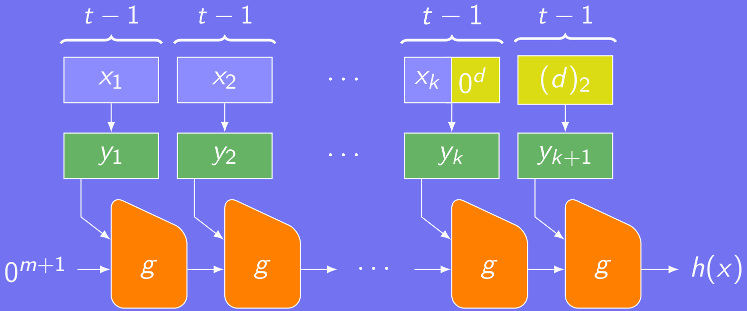
1. A function g defined by
$$g : \{0,1\}^{m+t} \longrightarrow \{0,1\}^{m}, \quad t \geq 1,$$
is called a *compression function*.
2. A hash function constructed by iteratively applying a compression function is called an *iterated hash function*.

Let x be a bitstring, $|x|$ stands for the length of x and for any bitstring y the concatenation of x and y is denoted $x\|y$. In particular for a bit x, $\underbrace{x\|x\|\cdots\|x}_{k \text{ times}}$ is denoted x^k.

Let a be an integer, then $(a)_2$ designates its binary representation.

Merkle-Damgård construction

Let $g : \{0,1\}^{m+t} \to \{0,1\}^m$, $t \geq 2$, be a compression function and x be a bitstring of length $n > 0$.



1. Split x into $k = \lceil \frac{n}{t-1} \rceil$ blocks
2. For $i = 1$ to k set y_i to x_i
3. Set y_k to $x_k \| 0^d$
4. Set y_{k+1} to $(d)_2$
5. Compute $z_1 = g(0^{m+1} \| y_1)$
6. For $i = 1$ to k
   compute $z_{i+1} = g(z_i \| 1 \| y_{i+1})$
7. Define $h(x)$ as z_{k+1}

Merkle-Damgård theorem

Theorem (Merkle-Damgård)

Let g be a collision resistant compression function defined from $\{0,1\}^{m+t}$ into $\{0,1\}^m$, with $t \geq 2$. Then the Merkle-Damgård construction is a collision resistant hash function.

Remark. Before proving this theorem we first note that the map $x \mapsto y$ must be injective. In fact if it is not injective then it is possible to find $x \neq x'$ such that $y = y'$. As a result we have $h(x) = h(x')$, that is h is not collision resistant.

Merkle-Damgård theorem

Proof. Assuming we have a collision on h, i.e. $x \neq x'$ and $h(x) = h(x')$, we will prove that a collision on the compression function g can be efficiently found.

First note that if $|x| \neq |x'|$, then they are padded with two different values d and d', respectively. Similarly $k+1$ and $k'+1$ denote the number of blocks for x and x'.

Case 1: consider $x \neq x'$ with $|x| \not\equiv |x'| \bmod (t-1)$. Then $d \neq d'$ and $y_{k+1} \neq y'_{k'+1}$. We then have

$$\begin{aligned} g(z_k \| 1 \| y_{k+1}) &= z_{k+1} = h(x) \\ &= h(x') = z'_{k'+1} \\ &= g(z'_{k'} \| 1 \| y'_{k'+1}) \end{aligned}$$

which is a collision on g since $y_{k+1} \neq y'_{k'+1}$.

Merkle-Damgård theorem

Proof (continued). Case 2a: consider $|x| \equiv |x'| \bmod (t-1)$ with $k = k'$. This implies $y_{k+1} = y_{k'+1}$, and we have

$$\begin{aligned} g(z_k \| 1 \| y_{k+1}) &= z_{k+1} = h(x) \\ &= h(x') = z'_{k+1} \\ &= g(z'_k \| 1 \| y'_{k+1}). \end{aligned}$$

If $z_k \neq z'_k$ then a collision is found. Otherwise we repeat the process and get

$$\begin{aligned} g(z_{k-1} \| 1 \| y_k) &= z_k \\ &= z'_k & &= g(z'_{k-1} \| 1 \| y'_k). \end{aligned}$$

Then either we have found a collision or we continue backward until one is obtained. If none is found then we get

$$z_1 = z'_1, \cdots, z_{k+1} = z'_{k+1} \lightning.$$

Merkle-Damgård theorem

Proof (continued). Case 2b: consider $|x| \equiv |x'| \bmod (t-1)$ with $k \neq k'$. Without loss of generality assume $k' > k$ and proceed as in case 2a. If no collision is found before $k = 1$ then we have

$$\begin{aligned} g(0^{m+1} \| y_1) &= z_1 \\ &= z'_{k'-k+1} \\ &= g(z'_{k'-k} \| 1 \| y'_{k'-k+1}). \end{aligned}$$

By construction the $m+1$st bit on the left is 0 while on the right it is 1. Hence we have found a collision.

All the cases being covered this completes the proof. $\square$

Constructing hash functions

Remark. We have proven the Merkle-Damgård theorem in the case $t \geq 2$. In fact this is true for any $t \geq 1$ but the case $t = 1$ requires a different approach as $|x| \bmod (t-1)$ cannot be considered[2].

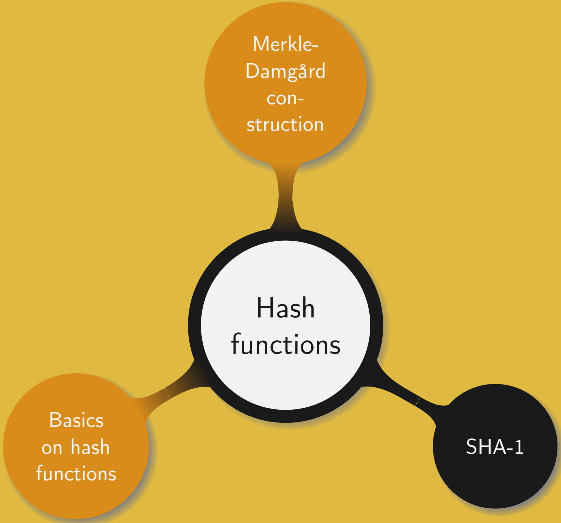
Merkle-Damgård construction served as a basis for the design of various hash functions. Two common examples are MD5 and SHA-1.

MD5 hash function:

- Designed in 1991
- First flaw discovered in 1996
- Collisions found in 2004
- Practical collisions demonstrated in 2005
- Flame malware used collisions on Windows certificates to infect computers... in 2012!

[2] This case will be proven in the homework.

Chapter organisation



Secure Hash Algorithm

Secure Hash Algorithm (SHA):

- 1993: SHA-0, 160 bits hash function, never widely adopted
- 1995: SHA-1, similar to SHA-0, with several weaknesses fixed
- 2001: SHA-2, significantly different from SHA-1
- 2005: first attacks against SHA-1
- 2008: SHA-0 totally broken (< 1 hour to find collisions)
- 2012: best theoretical attack on SHA-1 in 2^{61} operations
- 2012: Keccak hash function is selected to become SHA-3
- 2017: major companies have stopped accepting SHA-1 certificates

Padding

Given x of length $|x|$:

1. Append 1 to the message
2. Append 0s until the length is $-64 \bmod 512$
3. Append $|x|$ written in base 2 over 64 bits

Let y be the padded value of x. By construction $|y| \equiv 0 \bmod 512$. Break y into

$$k = \left\lfloor \frac{|x|}{512} \right\rfloor + 1$$

blocks of 512 bits each.

Example. Assume $|x| = 2800$ bits. Since $2800 \equiv 240 \bmod 512$ append a 1 followed by 207 0s, and the bit representation of 2800 over 64 bits. Thus y is composed of $k = 6$, 512-bit blocks.

SHA-1 algorithm

As SHA-1 follows the Merkle-Damgård construction it is simply described as an algorithm, while most of the work if performed by the compression function.

Algorithm. (*SHA-1*)

```
**Input**  : $x$ a bit string
**Output:** $h(x)$, where $h$ is SHA-1
$H_0 \leftarrow 67452301$; $H_1 \leftarrow EFCDAB89$; $H_2 \leftarrow 98BADCFE$;
$H_3 \leftarrow 10325476$; $H_4 \leftarrow C3D2E1F0$;
$d \leftarrow (447 - |x|) \bmod 512$;
$y \leftarrow x \| 1 \| 0^d \| (|x|)_2$;                /* $|x|$ expressed over 64 bits */
for $i \leftarrow 1$ to $k$ do
    $H_0, H_1, H_2, H_3, H_4 \leftarrow$ compress($H_0, H_1, H_2, H_3, H_4, y_i$)
end for
return $H_0 \| H_1 \| H_2 \| H_3 \| H_4$
```

SHA-1 compression function

The compression function onto which SHA-1 relies uses:

- The functions $f_0, \ldots, f_{79}$ defined by

$$f_i(B, C, D) = \begin{cases} (B \wedge C) \vee (\neg B \wedge D) & \text{if } 0 \leq i \leq 19 \\ B \oplus C \oplus D & \text{if } 20 \leq i \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{if } 40 \leq i \leq 59 \\ B \oplus C \oplus D & \text{if } 60 \leq i \leq 79 \end{cases}$$

- The constants $K_0, \ldots, K_{79}$ defined by

$$K_i = \begin{cases} \text{5A827999} & \text{if } 0 \leq i \leq 19 \\ \text{6ED9EBA1} & \text{if } 20 \leq i \leq 39 \\ \text{8F1BBCDC} & \text{if } 40 \leq i \leq 59 \\ \text{CA62C1D6} & \text{if } 60 \leq i \leq 79 \end{cases}$$

SHA-1 compression function

Algorithm. (*SHA-1 compression function*)

Input : Five 32-bit values H_0, H_1, H_2, H_3, H_4 and a 512-bit block y

Output : Five 32-bit values H_0, H_1, H_2, H_3, H_4

```
Function compress($H_0, H_1, H_2, H_3, H_4, y$):
    split $y$ into 16 words $W_0, \dots, W_{15}$;
    for $i \leftarrow 16$ to $79$ do
        $W_i \leftarrow ROTL(W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16})$
    end for
    $A \leftarrow H_0$; $B \leftarrow H_1$; $C \leftarrow H_2$; $D \leftarrow H_3$; $E \leftarrow H_4$;
    for $i \leftarrow 0$ to $79$ do
        $T \leftarrow ROTL^5(A) + f_i(B, C, D) + E + W_i + K_i$;
        $E \leftarrow D$; $D \leftarrow C$;
        $C \leftarrow ROTL^{30}(B)$;
        $B \leftarrow A$; $A \leftarrow T$;
    end for
    $H_0 \leftarrow H_0 + A$; $H_1 \leftarrow H_1 + B$; $H_2 = H_2 + C$; $H_3 = H_3 + D$; $H_4 = H_4 + E$;
    return $H_0, H_1, H_2, H_3, H_4$
end
```

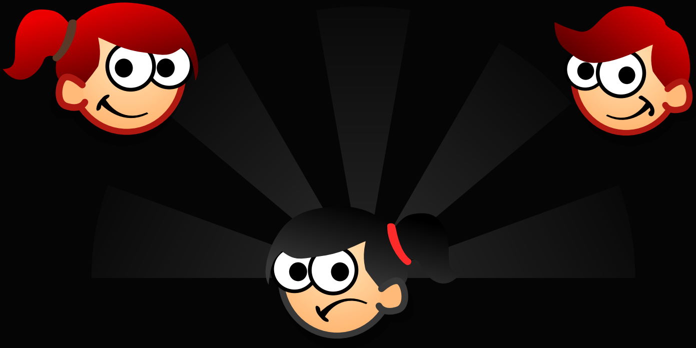
Comments on SHA-1

Remark.

- Compared to SHA-0, SHA-1 only adds *ROTL* in the construction of W_{16} to W_{79}
- All the constant in SHA-1 are constructed such as to be above any suspicion
- Weaknesses on SHA-1 lead to the SHA-3 competition
- SHA-2 is a family of six hash functions
- SHA-2 digests can have values 224, 256, 384 and 512
- To date there is no known security issue on any of the SHA-2 hash functions

Key points

- What is a hash function?
- Why are hash function important in cryptography?
- Explain the birthday paradox
- Describe the Merkle-Damgård construction
- What secure hash function should be used in 2021?



Thank you!