technische universiteit eindhoven

Cryptographic Hash Functions
Part I

Cryptography 1

Andreas Hülsing, TU/e
Based on slides by Sebastiaan de Hoogh, TU/e

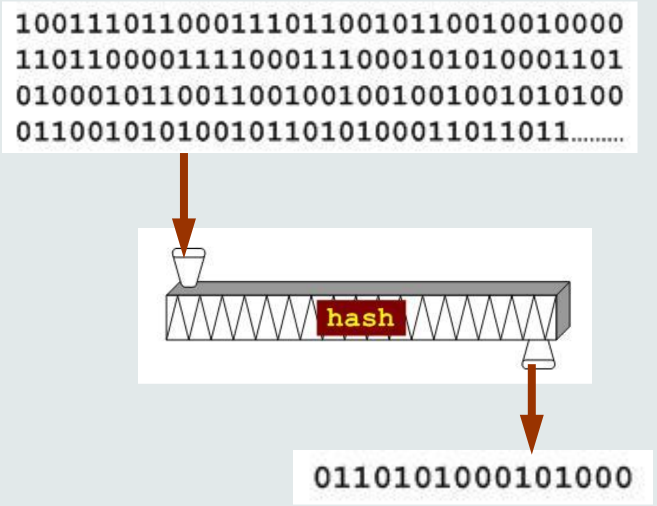
how are hash functions used?

- integrity protection
  - strong checksum
  - for file system integrity (Bit-torrent) or software downloads
- one-way ‘encryption’
  - for password protection
- asymmetric digital signature
- MAC – message authentication code
  - Efficient symmetric ‘digital signature’
- key derivation
- pseudo-random number generation
- …

what is a hash function?

- $h : \{0,1\}^* \rightarrow \{0,1\}^n$

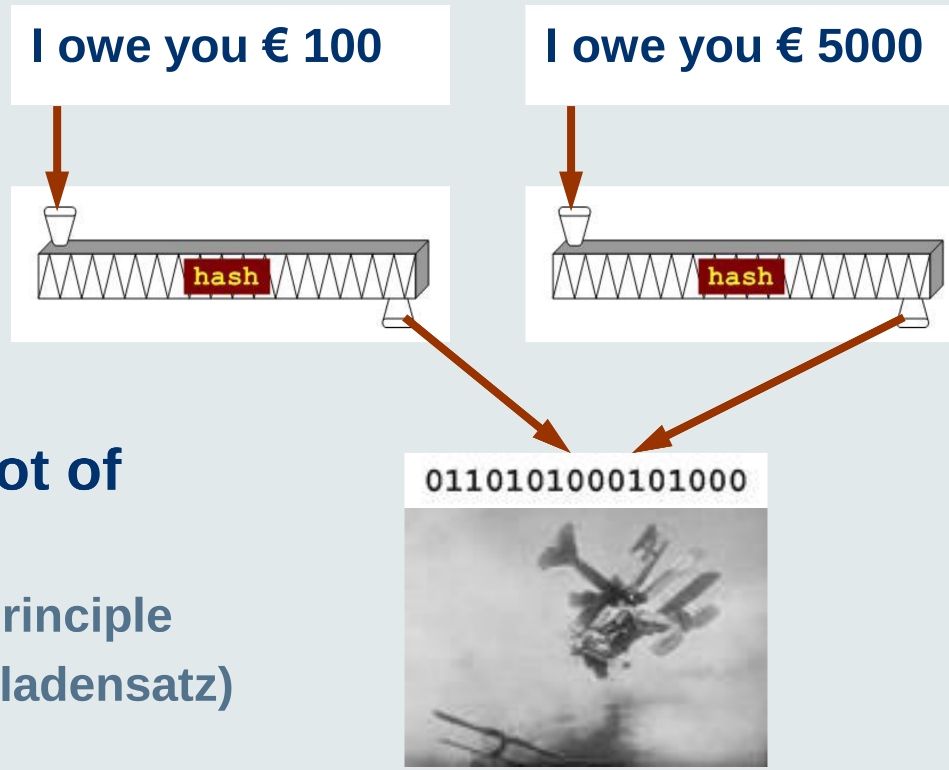
  (general: $h : S \rightarrow \{0,1\}^n$ for some set S)
- **input**: bit string m of arbitrary length
  - length may be 0
  - in practice a very large bound on the length is imposed, such as 2^{64} (≈ 2.1 million TB)
  - input often called the ***message***
- **output**: bit string $h(m)$ of fixed length n
  - e.g. n = 128, 160, 224, 256, 384, 512
  - ***compression***
  - output often called ***hash value***, ***message digest***, ***fingerprint***
- $h(m)$ is **easy to compute** from m
- no secret information, no key



hash collision

- m_1, m_2 are a *collision* for h if

  $h(m_1) = h(m_2)$ while $m_1 \neq m_2$



different documents

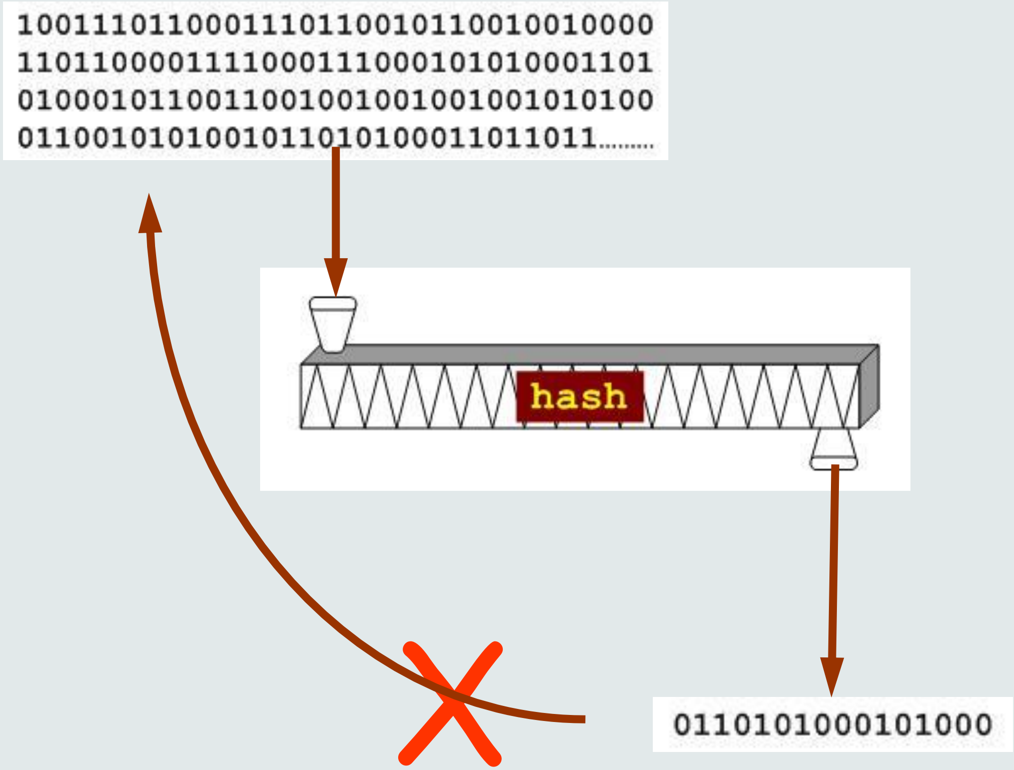
identical hash = collision

- there exist a lot of collisions
  - pigeonhole principle (a.k.a. Schubladensatz)

preimage

- given h_0, then m is a *preimage* of h_0 if

  $$h(m) = h_0$$

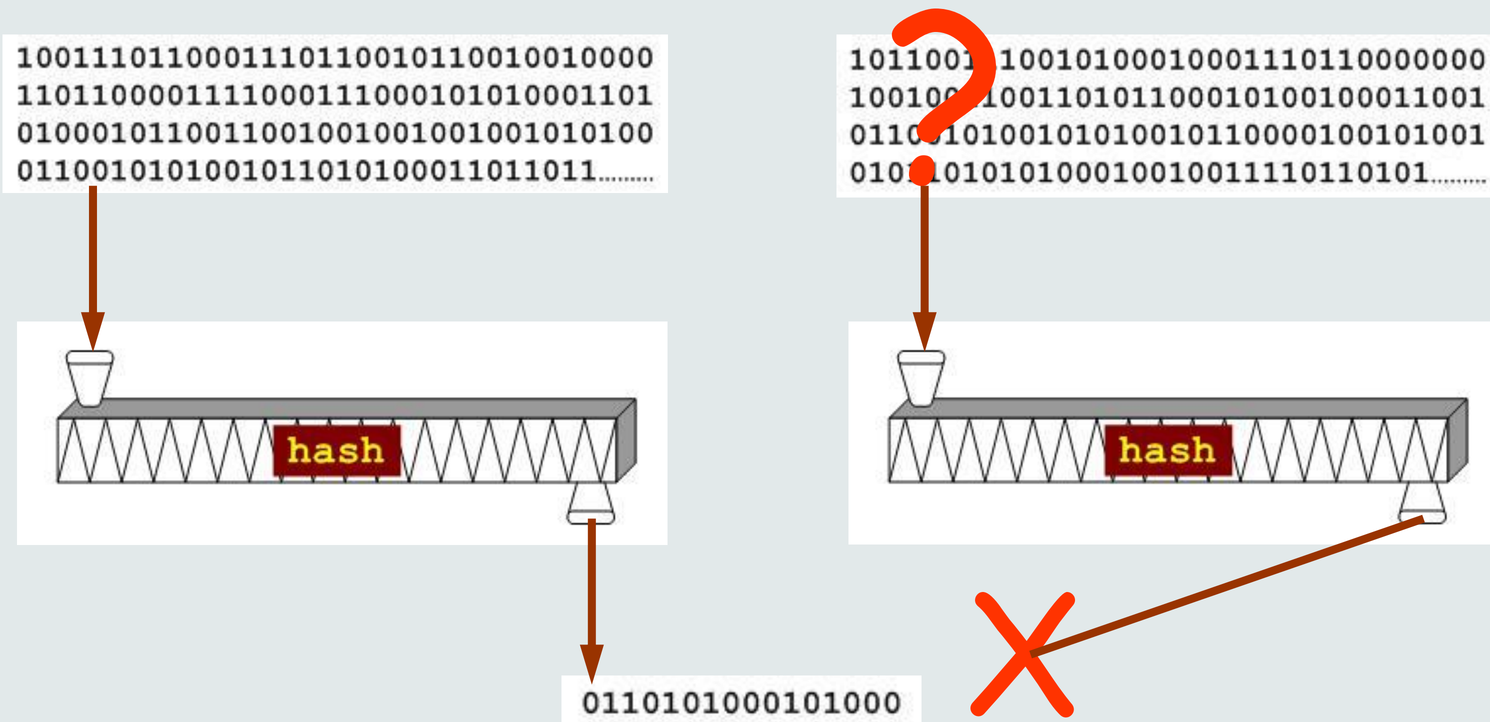


Note:
h_0 might have many preimages!

second preimage

- given m_0, then m is a *second preimage* of m_0 if

  $h(m) = h(m_0)$ while $m \neq m_0$



cryptographic hash function requirements

- ***collision resistance***: it should be computationally infeasible to find a collision m_1, m_2 for h
  - i.e. $h(m_1) = h(m_2)$
- ***preimage resistance***: given h_0 it should be computationally infeasible to find a preimage m for h_0 under h
  - i.e. $h(m) = h_0$
- ***second preimage resistance***: given m_0 it should be computationally infeasible to find a second preimage m for m_0 under h
  - i.e. $h(m) = h(m_0)$

other terminology

- *one-way* function = preimage resistant
  - sometimes preimage + second preimage resistant
- *weak collision resistant* = second preimage resistant
- *strong collison resistant* = collision resistant
- OWHF – one-way hash function
  - preimage and second preimage resistant
- CRHF – collision resistant hash function
  - second preimage resistant and collision resistant

Formal treatment

- **Efficient Algorithm**
  - Runs in **polynomial time**, i.e. for input of length **n**, $t_A \leq n^k = poly(n)$ for some **constant k**
- **Probabilistic Polynomial Time (PPT) Algorithm:**
  - **Randomized** Algorithm
  - Runs in **polynomial time**
  - Outputs the right solution with some **probability**
- **Negligible:**
  We call $\varepsilon(n)$ negligible if

$$(\exists n_c > 0)(\forall n > n_c)\colon \varepsilon(n) < \frac{1}{poly(n)}$$

technische universiteit eindhoven

Formal treatment

For security parameter n, key space K, message space M and range R, a family of hash functions $F_n=(I,H)$ is a pair of efficient algorithms:

- $I(1^n)$: The key generation algorithm that outputs a (public) function key $k \in K$
- $H(k,m)$: Takes a key $k \in K$ and a message m $\in M$ and outputs outputs the hash value $H(k,m) \in R$

TU/e technische universiteit eindhoven

Formal security properties: CR

Collision resistance: For any PPT adversary A, the following probability is negligible in n:

$$Pr[k \leftarrow I(1^n), (m_1, m_2) \leftarrow A(1^n, k): H(k, m_1) = H(k, m_2) \wedge (m_1 \neq m_2)]$$

Formal security properties: CR

$\mathcal{C}$

$k \leftarrow I(1^n)$

$\xrightarrow{\quad k \quad}$

$\xleftarrow{\quad (m_1, m_2) \quad}$

$H(k, m_1) = H(k, m_2)$
$\wedge (m_1 \neq m_2)?$

Formal security properties: PRE

Preimage resistance: **For any PPT adversary A, the following probability is negligible in n:**

$$Pr[k \leftarrow I(1^n), h \leftarrow R, m \leftarrow A(1^n, k, h): H(k, m) = h]$$

Formal security properties: PRE

$\mathcal{C}$

$$k \leftarrow I(1^n)$$
$$h \leftarrow R \xrightarrow{\quad k,h \quad}$$

$$\xleftarrow{\quad m \quad}$$

$$H(k,m) = h?$$

Formal security properties: SPR

Second-preimage resistance: **For any PPT adversary A, the following probability is negligible in n:**

$$\boldsymbol{Pr[k \leftarrow I(1^n), m \leftarrow M, m' \leftarrow A(1^n, k, m): H(k, m) = H(k, m') \wedge (m_1 \neq m_2)]}$$

Formal security properties: SPR

$\mathcal{C}$

$\mathcal{A}$ 

$k \leftarrow I(1^n)$

$m \leftarrow M$ $\quad \xrightarrow{\quad m, k \quad}$

$\xleftarrow{\quad m' \quad}$

$$H(k, m) = H(k, m') \wedge (m \neq m')?$$

Reductions

- **Transform an algorithm for problem 1 into an algorithm for problem 2.**
- **„Reduces problem 2 to problem 1"**
- **Allows to relate the hardness of problems:**

  If there exists an efficient reduction that reduces problem 2 to problem 1 then an efficient algorithm solving problem 1 can be used to efficiently solve problem 2.

Reductions II

Use in cryptography:

- Relate security properties
- „Provable Security“: Reduce an assumed to be hard problem to breaking the security of your scheme.
- Actually this does not proof security! Only shows that scheme is secure IF the problem is hard.

Relations between hash function security properties

Easy start: CR -> SPR

Theorem (informal): If F is collision resistant then it is second preimage resistant.

Proof:

- By contradiction: Assume A breaks SPR of F then we can build an oracle machine M^A that breaks CR.
- Given key k, M^A first samples random $m \leftarrow M$
- M^A runs $m' \leftarrow A(1^n, k, m)$ and outputs (m', m)
- M^A runs in approx. same time as A and has same success probability. -> Tight reduction

Formal security properties: CR

$\mathcal{C}$ $\qquad$ M^{A}

$k \leftarrow I(1^n)$

$\xrightarrow{\quad k \quad}$

$m_1 \leftarrow M$

$\xrightarrow{\quad m_1, k \quad}$

$\xleftarrow{\quad m_2 \quad}$

$\xleftarrow{\quad (m_1, m_2) \quad}$

$$H(k, m_1) = H(k, m_2)$$
$$\wedge\ (m_1 \neq m_2)?$$

Easy start: CR -> SPR

Theorem (informal): If F is collision resistant then it is second preimage resistant.

Proof:

- By contradiction: Assume A breaks SPR of F then we can build an oracle machine M^A that breaks CR.
- Given key k, M^A first samples random $m \leftarrow M$
- M^A runs $m' \leftarrow A(1^n, k, m)$ and outputs (m', m)
- M^A runs in approx. same time as A and has same success probability. -> Tight reduction

SPR -> PRE ?

Theorem (informal): If F is second-preimage resistant then it is also preimage resistant.

Proof:

- By contradiction: Assume A breaks PRE of F then we can build an oracle machine M^A that breaks SPR.
- Given key k, m, M^A runs $m' \leftarrow A(1^n, k, H(k, m))$ and outputs (m', m)
- M^A runs in same time as A and has same success probability.

Do you find the mistake?

SPR -> PRE ?

Theorem (informal): If F is second-preimage resistant then it is also preimage resistant.

Counter example:

- **the *identity function* $id : \{0,1\}^n \rightarrow \{0,1\}^n$ is second-preimage resistant but not preimage resistant**

SPR -> PRE ?

Theorem (informal): If F is second-preimage resistant then it is also preimage resistant.

Proof:

- By contradiction: Assume A breaks PRE of F then we can build an oracle machine M^A that breaks SPR.
- Given key k, m, M … and outputs (m',m)
- M^A runs in same time as A and has same success probability.

We are not guaranteed that $m \neq m'$!

Do you find the mistake?

SPR -> PRE ?

Theorem (informal, corrected): If F is second-preimage resistant, $|M| \geq 2|R|$, and $H(k,m)$ is regular for every k, then it is also preimage resistant.

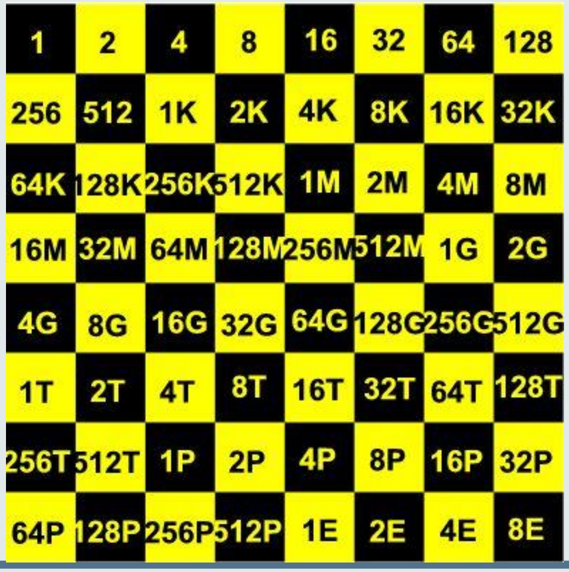
Proof:

- **By contradiction: Assume A breaks PRE of F then we can build an oracle machine M^A that breaks SPR.**
- **Given key k, m, M^A runs $m' \leftarrow A(1^n, k, H(k,m))$ and outputs (m',m)**
- **M^A runs in same time as A and has at least half the success probability.**

Same corrections have to be applied for CR -> PRE

generic (brute force) attacks

- assume: hash function behaves like random function
- **preimages** and **second preimages** can be found by **random guessing search**
  - search space: ≈ n bits, ≈ 2^n hash function calls
- **collisions** can be found by **birthdaying**
  - search space: ≈ ½n bits,
    ≈ $2^{½n}$ hash function calls
- this is a big difference
  - MD5 is a 128 bit hash function
  - (second) preimage random search:
    ≈ 2^{128} ≈ $3\text{x}10^{38}$ MD5 calls
  - collision birthday search: only
    ≈ 2^{64} ≈ $2\text{x}10^{19}$ MD5 calls



birthday paradox

- **birthday paradox**

  given a set of t (≥ 10) elements

  take a sample of size k (drawn with repetition)

  in order to get a probability $\geq$ ½ on a collision

  (i.e. an element drawn at least twice)

  k has to be $> 1.2\sqrt{t}$
- **consequence**

  if $F : A \rightarrow B$ is a surjective random function

  and $|A| \gg |B|$

  then one can expect a collision after about $\sqrt{(|B|)}$ random function calls

meaningful birthdaying

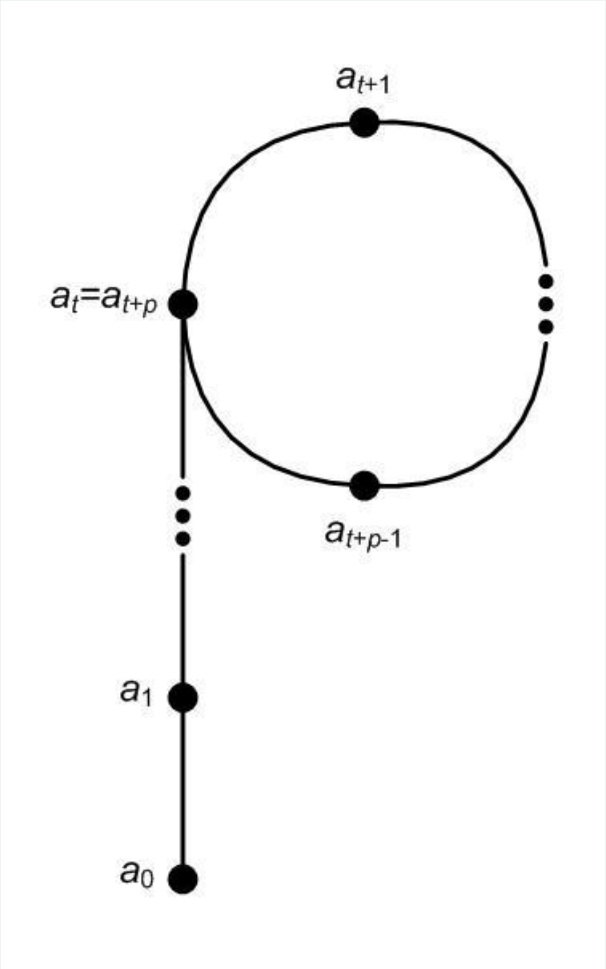
- random birthdaying
  - do exhaustive search on ½n bits
  - messages will be ‘random’
  - messages will not be ‘meaningful’
- Yuval (1979)
  - start with two meaningful messages m_1, m_2 for which you want to find a collision
  - identify ½n independent positions where the messages can be changed at bitlevel without changing the meaning
    - e.g. tab ←→ space, space ←→ newline, etc.
  - do random search on those positions

implementing birthdaying

- **naïve**
  - store $2^{½n}$ possible messages for m_1 and $2^{½n}$ possible messages for m_2 and check all 2^n pairs
- **less naïve**
  - store $2^{½n}$ possible messages for m_1 and for each possible m_2 check whether its hash is in the list
- **smart: Pollard-ρ with Floyd's cycle finding algorithm**
  - computational complexity still $O(2^{½n})$
  - but only constant small storage required

Pollard-ρ and Floyd cycle finding

- **Pollard-ρ**
  - iterate the hash function:
    $a_0, a_1 = h(a_0), a_2 = h(a_1), a_3 = h(a_2), \ldots$
  - this is ultimately periodic:
    - there are minimal t, p such that $a_{t+p} = a_t$
    - theory of random functions: both t, p are of size $2^{½n}$
- **Floyd's cycle finding algorithm**
  - Floyd: start with (a_1,a_2) and compute $(a_2,a_4), (a_3,a_6), (a_4,a_8), \ldots, (a_q,a_{2q})$ until $a_{2q} = a_q$; this happens for some $q < t + p$



security parameter

- *security parameter* n: resistant against (brute force / random guessing) attack with search space of size 2^n
  - complexity of an n-bit exhaustive search
  - n-bit *security level*
- nowadays 2^{80} computations deemed impractical
  - security parameter 80 seen as sufficient in most cases
- but 2^{64} computations should be about possible
  - though a.f.a.i.k. nobody has done it yet
  - security parameter 64 now seen as insufficient in most cases
- in the near future: security parameter 128 will be required

- for collision resistance hash length should be $2n$ to reach security with parameter n