ΥΣ13 - Computer Security

Hashing

Κώστας Χατζηκοκολάκης

Context

- **Goal**
  - Represent large/sensitive message by a smaller one
  - Numerous applications

Context

- **Goal**
  - Represent large/sensitive message by a smaller one
  - Numerous applications
- **Solution** : hash function
  - $h(x) : \{0,1\}^* \rightarrow \{0,1\}^n$
  - $h(x)$ is the hash/digest of x

Properties

- **One-way**
  - $x \rightarrow h(x)$: easy

Properties

- **One-way**
  - $x \to h(x)$: easy
  - $h(x) \to x$: hard
    - Even to find a single bit of x !
- **No collisions**
  - Do $x \neq x'$ exist such that $h(x) = h(x')$?

Properties

- **One-way**
  - $x \rightarrow h(x)$: easy
  - $h(x) \rightarrow x$: hard
    - Even to find a single bit of x !
- **No collisions**
  - Do $x \neq x'$ exist such that $h(x) = h(x')$? **YES**
  - But the should be hard to find!

Collision-resistance

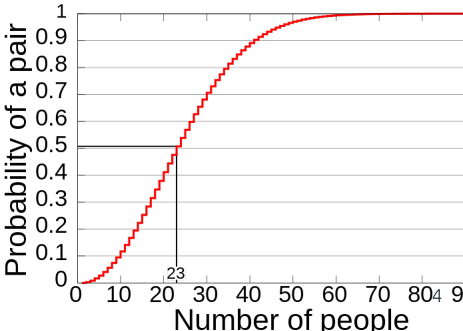
Birthday paradox

- How many people do we need so that any 2 have the same birthday with pb 50%?

Collision-resistance

Birthday paradox

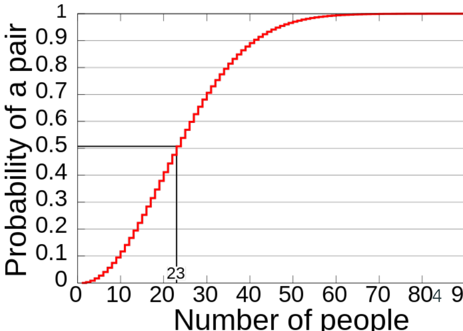
- How many people do we need so that any 2 have the same birthday with pb 50%?
- **Just 23!**
- $pb = 1 - \frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{365-22}{365} \approx 0.507$



Collision-resistance

Birthday paradox

- How many people do we need so that any 2 have the same birthday with pb 50%?
- **Just 23!**
- $pb = 1 - \frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{365-22}{365} \approx 0.507$
- Approximation
  - $e^{-x} \approx 1 - x \, (x \approx 0)$
  - $pb \approx 1 - e^{-\frac{m^2}{2 \cdot 365}}$



Collision-resistance

Birthday paradox

- m people, T possible values each
  - $pb \approx 1 - e^{-m^2/2T}$
  - $m \approx \sqrt{-2T \ln(1-pb)}$

Collision-resistance

Birthday paradox

- m people, T possible values each
  - $pb \approx 1 - e^{-m^2/2T}$
  - $m \approx \sqrt{-2T \ln(1-pb)}$
- 50 bit hash
  - $T \approx 10^{15}$ total values (huge)
  - m: number of messages we hash
  - How many for a 50% collision?

Collision-resistance

Birthday paradox

- m people, T possible values each
  - $pb \approx 1 - e^{-m^2/2T}$
  - $m \approx \sqrt{-2T \ln(1-pb)}$
- 50 bit hash
  - $T \approx 10^{15}$ total values (huge)
  - m: number of messages we hash
  - How many for a 50% collision?
  - 40M (milliseconds to generate!)

One-way encryption

- **Goal**
  - Store *x* in an encrypted form
  - We don’t need to decrypt, only to test equality of encrypted messages

One-way encryption

- **Goal**
  - Store x in an encrypted form
  - We don't need to decrypt, only to test equality of encrypted messages
  - Example: password authentication
    - Protect against data breach
    - Only need to test whether input is correct!

One-way encryption

- **Goal**
  - Store x in an encrypted form
  - We don't need to decrypt, only to test equality of encrypted messages
  - Example: password authentication
    - Protect against data breach
    - Only need to test whether input is correct!

- **Solution**
  - Store $h(x)$

One-way encryption

- **Goal**
  - Store x in an encrypted form
  - We don't need to decrypt, only to test equality of encrypted messages
  - Example: password authentication
    - Protect against data breach
    - Only need to test whether input is correct!
- **Solution**
  - Store $h(x)$
  - Better: generate random r (salt), store $r, h(x, r)$ why?

One-way encryption

- **Goal**
  - Store x in an encrypted form
  - We don't need to decrypt, only to test equality of encrypted messages
  - Example: password authentication
    - Protect against data breach
    - Only need to test whether input is correct!
- **Solution**
  - Store $h(x)$
  - Better: generate random r (salt), store $r, h(x, r)$ why?
- Which properties of h does this rely on?

One-way encryption

- **Goal**
  - Store x in an encrypted form
  - We don't need to decrypt, only to test equality of encrypted messages
  - Example: password authentication
    - Protect against data breach
    - Only need to test whether input is correct!
- **Solution**
  - Store $h(x)$
  - Better: generate random r (salt), store $r, h(x, r)$ why?
- Which properties of h does this rely on?
  - One-wayness: should not learn the password
  - Collision-resistance: should not login with different password

One-way encryption

Can we break it?

- Preimage attack : find x' such that $h(x')$ matches the given $h(x)$

One-way encryption

Can we break it?

- Preimage attack : find x' such that $h(x')$ matches the given $h(x)$
- Assume 365 outputs. How many x's to generate for 50% success pb?

One-way encryption

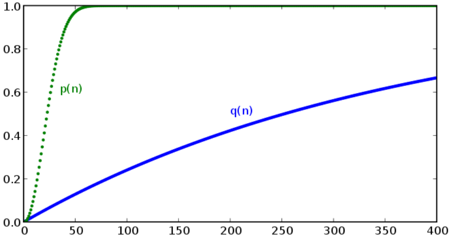
Can we break it?

- Preimage attack : find x' such that $h(x')$ matches the given $h(x)$
- Assume 365 outputs. How many x's to generate for 50% success pb?
- **253!** huh? but we said 23…

One-way encryption

Can we break it?

- Preimage attack : find x' such that $h(x')$ matches the given $h(x)$
- Assume 365 outputs. How many x's to generate for 50% success pb?
- **253!** huh? but we said 23…
- Different problem: pb that someone has the same birthday as you!
- $pb = 1 - \frac{364}{365}^n$
  (only 6% for $n = 23$)



Signatures

- Assume: sign(x, Alice) is a message that can only be constructed by Alice
    - We will see how to do this using asymmetric encryption!

Signatures

- Assume: sign(x, Alice) is a message that can only be constructed by Alice
  - We will see how to do this using asymmetric encryption!
- Can be used to show approval of x
  - Eg: x is a contract signed by Alice
  - But it is expensive for large x
- Solution: provide sign($h(x)$, Alice)
- Alice needs to know x to construct $h(x)$!

Signatures

- Assume: sign(x, Alice) is a message that can only be constructed by Alice
  - We will see how to do this using asymmetric encryption!
- Can be used to show approval of x
  - Eg: x is a contract signed by Alice
  - But it is expensive for large x
- Solution: provide sign($h(x)$, Alice)
- Alice needs to know x to construct $h(x)$!
  - Does this show approval of x?

Signatures

- Assume: sign(x, Alice) is a message that can only be constructed by Alice
  - We will see how to do this using asymmetric encryption!
- Can be used to show approval of x
  - Eg: x is a contract signed by Alice
  - But it is expensive for large x
- Solution: provide sign($h(x)$, Alice)
- Alice needs to know x to construct $h(x)$!
  - Does this show approval of x? Yes if collision-free

Signatures

- Assume: sign(x, Alice) is a message that can only be constructed by Alice
  - We will see how to do this using asymmetric encryption!
- Can be used to show approval of x
  - Eg: x is a contract signed by Alice
  - But it is expensive for large x
- Solution: provide sign($h(x)$, Alice)
- Alice needs to know x to construct $h(x)$!
  - Does this show approval of x? Yes if collision-free
  - One-wayness can be useful if we want to reveal x in the future!

Signatures

Can we break it?

- Alice wants to force bob into signing a fraudulent contract x' !

Signatures

Can we break it?

- Alice wants to force bob into signing a fraudulent contract x' !
- Collision attack : find
  - honest contract x and fraudulent contract x'
  - such that $h(x) = h(x')$
  - So Bob will provide $\text{sign}(h(x), \text{Bob}) = \text{sign}(h(x'), \text{Bob})$

Signatures

Can we break it?

- Alice wants to force bob into signing a fraudulent contract x' !
- Collision attack : find
  - honest contract x and fraudulent contract x'
  - such that $h(x) = h(x')$
  - So Bob will provide $\text{sign}(h(x), \text{Bob}) = \text{sign}(h(x'), \text{Bob})$
- Assume 365 outputs. How many x, x's to generate for 50% success pb?

Signatures

Can we break it?

- Alice wants to force bob into signing a fraudulent contract x' !
- Collision attack : find
  - honest contract x and fraudulent contract x'
  - such that $h(x) = h(x')$
  - So Bob will provide $\text{sign}(h(x), \text{Bob}) = \text{sign}(h(x'), \text{Bob})$
- Assume 365 outputs. How many x, x's to generate for 50% success pb?
  - 23, but...

Signatures

Can we break it?

- Alice wants to force bob into signing a fraudulent contract x' !
- Collision attack : find
  - honest contract x and fraudulent contract x'
  - such that $h(x) = h(x')$
  - So Bob will provide $\text{sign}(h(x), \text{Bob}) = \text{sign}(h(x'), \text{Bob})$
- Assume 365 outputs. How many x, x's to generate for 50% success pb?
  - 23, but...
  - useless if x, x' are both honest/fraudulent.
  - So we need double the attempts (but still a big problem)

Ideal hash function

- **Random Oracle**
  - Given $x \in \{0,1\}^*$, generate random $h(x) \in \{0,1\}^n$
  - Remember it for future calls!

Ideal hash function

- **Random Oracle**
  - Given $x \in \{0,1\}^*$, generate random $h(x) \in \{0,1\}^n$
  - Remember it for future calls!
- Is this one-way?

Ideal hash function

- **Random Oracle**
  - Given $x \in \{0,1\}^*$, generate random $h(x) \in \{0,1\}^n$
  - Remember it for future calls!
- Is this one-way?
  - $\Pr[h(x) = y] = \Pr[h(x') = y]$ for any x, x'
  - So x and $h(x)$ are independent (the oracle does not use x!)

Ideal hash function

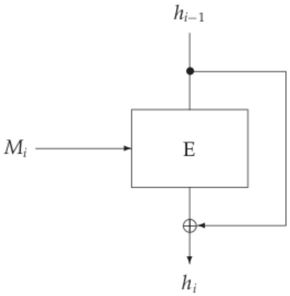
- **Random Oracle**
  - Given $x \in \{0,1\}^*$, generate random $h(x) \in \{0,1\}^n$
  - Remember it for future calls!
- Is this one-way?
  - $\text{Pb}[h(x) = y] = \text{Pb}[h(x') = y]$ for any x, x'
  - So x and $h(x)$ are independent (the oracle does not use x!)
- Is this collision-resistant?

Ideal hash function

- **Random Oracle**
  - Given $x \in \{0,1\}^*$, generate random $h(x) \in \{0,1\}^n$
  - Remember it for future calls!
- Is this one-way?
  - $\mathrm{Pb}[h(x) = y] = \mathrm{Pb}[h(x') = y]$ for any x, x'
  - So x and $h(x)$ are independent (the oracle does not use x!)
- Is this collision-resistant?
  - As much as the birthday paradox allows!

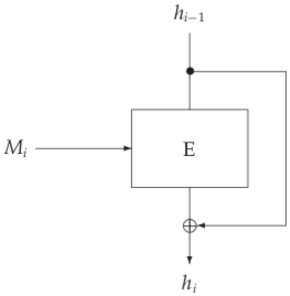
Constructing a hash function

- Recall: we can create a block cipher from a random function (Feistel)
  - in other words: from an ideal hash function



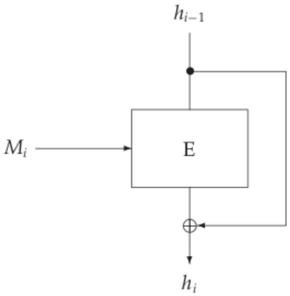
Constructing a hash function

- Recall: we can create a block cipher from a random function (Feistel)
  - in other words: from an ideal hash function
- We can also do the opposite!
  - Given a block cipher, construct a hash
  - Use the input *x* as the key
  - Start *h* from 0, update each time
  - XOR with the output of the previous round



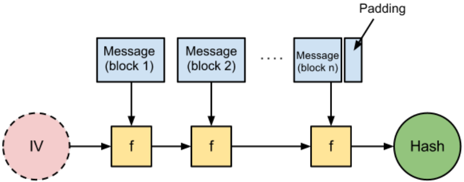
Constructing a hash function

- Recall: we can create a block cipher from a random function (Feistel)
  - in other words: from an ideal hash function
- We can also do the opposite!
  - Given a block cipher, construct a hash
  - Use the input *x* as the key
  - Start *h* from 0, update each time
  - XOR with the output of the previous round
- Needs at least 128 bits block size!
  - How many messages for 0.0001% collision? Do the math…
  - Used in practice with AES



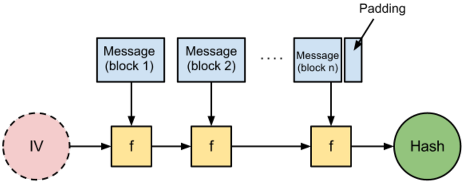
Merkle-Damgård

- Compression function $f : \{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$
- If f is collision-resistant, so is h
- Padding if the last block is smaller. How?



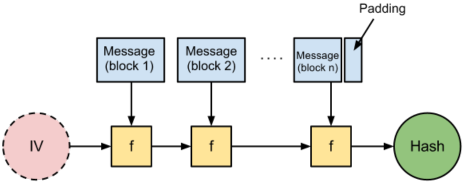
Merkle-Damgård

- Compression function $f : \{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$
- If f is collision-resistant, so is h
- Padding if the last block is smaller. How?
  - Is it safe to add zeroes?



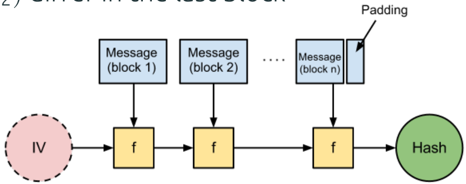
Merkle-Damgård

- Compression function $f : \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$
- If f is collision-resistant, so is h
- Padding if the last block is smaller. How?
  - Is it safe to add zeroes?
  - No! $h(\texttt{HashInpu t}) = h(\texttt{HashInpu t000000})$



Merkle-Damgård

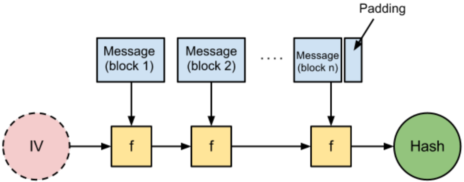
- Compression function $f: \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$
- If f is collision-resistant, so is h
- Padding if the last block is smaller. How?
  - Is it safe to add zeroes?
  - No! $h(\texttt{HashInpu t}) = h(\texttt{HashInpu t000000})$
- Safe conditions
  - $|m_1| = |m_2|$: $|\mathrm{Pad}(m_1)| = |\mathrm{Pad}(m_2)|$
  - $|m_1| \neq |m_2|$: $\mathrm{Pad}(m_1), \mathrm{Pad}(m_2)$ differ in the last block
- Common:
  - `HashInpu t1000000 <size>`



Merkle-Damgård

Length extension

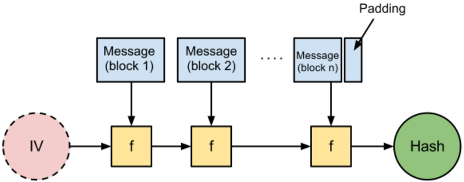
- Can we construct $h(m_1 \| m_2)$ from $h(m_1)$?



Merkle-Damgård

Length extension

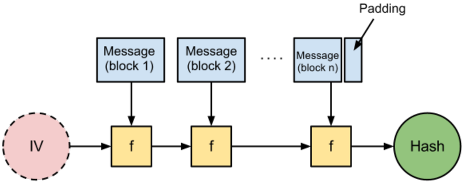
- Can we construct $h(m_1 \| m_2)$ from $h(m_1)$?
- What if padding is used?



Merkle-Damgård

Length extension

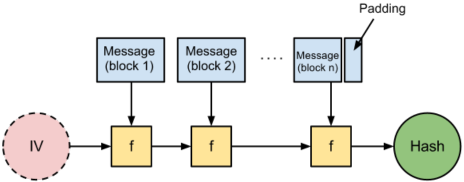
- Can we construct $h(m_1 \| m_2)$ from $h(m_1)$?
- What if padding is used?
- Does this violate
  - one-wayness?
  - collision-resistance?



Merkle-Damgård

Length extension

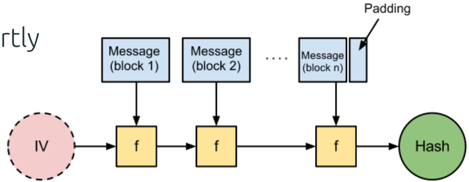
- Can we construct $h(m_1 \| m_2)$ from $h(m_1)$?
- What if padding is used?
- Does this violate
  - one-wayness?
  - collision-resistance?
- Is it a problem?



Merkle-Damgård

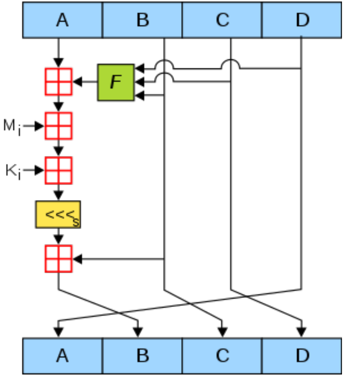
Length extension

- Can we construct $h(m_1 \| m_2)$ from $h(m_1)$?
- What if padding is used?
- Does this violate
  - one-wayness?
  - collision-resistance?
- Is it a problem?
  - Maybe…we'll come back shortly



MD5

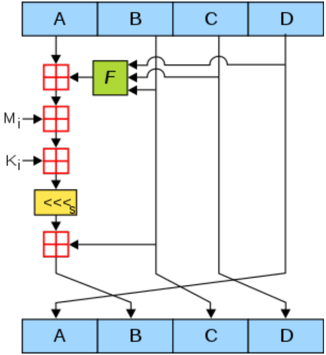
- 128 bits output
- 512 bit blocks (with padding)
- Merkle-Damgård design
- Compression function:
  - 4 rounds of 16 operations
  - 4 simple non-linear functions *F*



MD5

Attacks

- 1996: collisions in the compression function
- 2004: collision attacks
- 2008: fraudulent certificate
- Common suffix can be added
  - $h(m_1) = h(m2) \Rightarrow h(m_1 \| m) = h(m_2 \| m)$
  - Similar to length extension
- Preimage attack still hard



SHA family

SHA-0

- NIST, 1993
- 160 bits
- Merkle-Damgård design
- **Attacks**
  - 1998: theoretical collision in 2^{61} steps
  - 2004: real collision (2^{51} steps)
  - 2008: collision in 2^{31} steps (1 hour on average PC)

SHA family

SHA-1

- SHA-0 + a bitwise rotation in the compression function
  - 160 bits, Merkle-Damgård design
- **Attacks**
  - 2005: theoretical collision in 2^{69} steps
  - 2017: real collision
    - `http://shattered.io/`
    - Still expensive: 2^{63} steps (6500 CPU + 100 GPU years)
  - Many applications affected (git, svn, …)
    - but no reason to panic

SHA family

- SHA-2
  - 2001
  - 224/256/384/512 bits, Merkle-Damgård design
  - Attacks are still hard

SHA family

- SHA-2
  - 2001
  - 224/256/384/512 bits, Merkle-Damgård design
  - Attacks are still hard
- SHA-3
  - 2012
  - 224/256/384/512 bits
  - The first one not using the Merkle-Damgård design
  - Protection against length extension

Protecting integrity

- **Problem**
  - Downloaded 1GB file, how to know it is correct?

Protecting integrity

- **Problem**
  - Downloaded 1GB file, how to know it is correct?
- **Solution**
  - send h(file) together with the file
  - Protects against errors

Protecting integrity

- **Problem**
  - Downloaded 1GB file, how to know it is correct?
- **Solution**
  - send h(file) together with the file
  - Protects against errors
- Does it protect against a malicious adversary?

Protecting integrity

- **Problem**
  - Downloaded 1GB file, how to know it is correct?
- **Solution**
  - send h(file) together with the file
  - Protects against errors
- Does it protect against a malicious adversary?
  - No! The adversary can alter both the file and its digest

Protecting integrity

MAC

- Keyed function
  - $\mathrm{MAC}_k : \{0,1\}^* \to \{0,1\}^n$
- Unforgeable
  - cannot produce $\mathrm{MAC}_k(m)$ without k
  - even if $(m_1, \mathrm{MAC}_k(m_1)), \ldots, (m_k, \mathrm{MAC}_k(m_k))$ are known!
- Alice and Bob need a shared key k

Protecting integrity

HMAC

- construct MAC_k from a hash h **how?**

Protecting integrity

HMAC

- construct MAC_k from a hash h **how?**
- $\text{HMAC}_k(m) = h(k \| m)$?

Protecting integrity

HMAC

- construct MAC_k from a hash h **how?**
- $HMAC_k(m) = h(k \| m)$?
  - Length extension attack!
  - url: `bank.com/transfer?from=Alice`, digest: $h(k \| \text{url})$

Protecting integrity

HMAC

- construct MAC_k from a hash h **how?**
- $\text{HMAC}_k(m) = h(k \| m)$?
  - Length extension attack!
  - url: `bank.com/transfer?from=Alice`, digest: $h(k \| \text{url})$
- $\text{HMAC}_k(m) = h(m \| k)$?

Protecting integrity

HMAC

- construct MAC_k from a hash h **how?**
- $\text{HMAC}_k(m) = h(k \| m)$?
  - Length extension attack!
  - url: `bank.com/transfer?from=Alice`, digest: $h(k \| \text{url})$
- $\text{HMAC}_k(m) = h(m \| k)$?
  - Better, but collisions are easily exploitable

Protecting integrity

HMAC

- construct MAC_k from a hash h **how?**
- $\text{HMAC}_k(m) = h(k \| m)$?
  - Length extension attack!
  - url: `bank.com/transfer?from=Alice`, digest: $h(k \| \text{url})$
- $\text{HMAC}_k(m) = h(m \| k)$?
  - Better, but collisions are easily exploitable
- $\text{HMAC}_k(m) = h(m \| k \| m)$?
  - Better, with some vulnerabilities

Protecting integrity

HMAC

- construct MAC_k from a hash h **how?**
- $HMAC_k(m) = h(k \| m)$?
  - Length extension attack!
  - url: `bank.com/transfer?from=Alice`, digest: $h(k \| \text{url})$
- $HMAC_k(m) = h(m \| k)$?
  - Better, but collisions are easily exploitable
- $HMAC_k(m) = h(m \| k \| m)$?
  - Better, with some vulnerabilities
- $HMAC_k(m) = h(m \| h(k \| m))$
  - standard approach

References

- Mironov, Hash functions: Theory attacks and applications.
- Ross Anderson, Security Engineering, Sections 5.3.1, 5.6