COMP4127/COMP7850

Information Security

Hash function

Learning Outcome

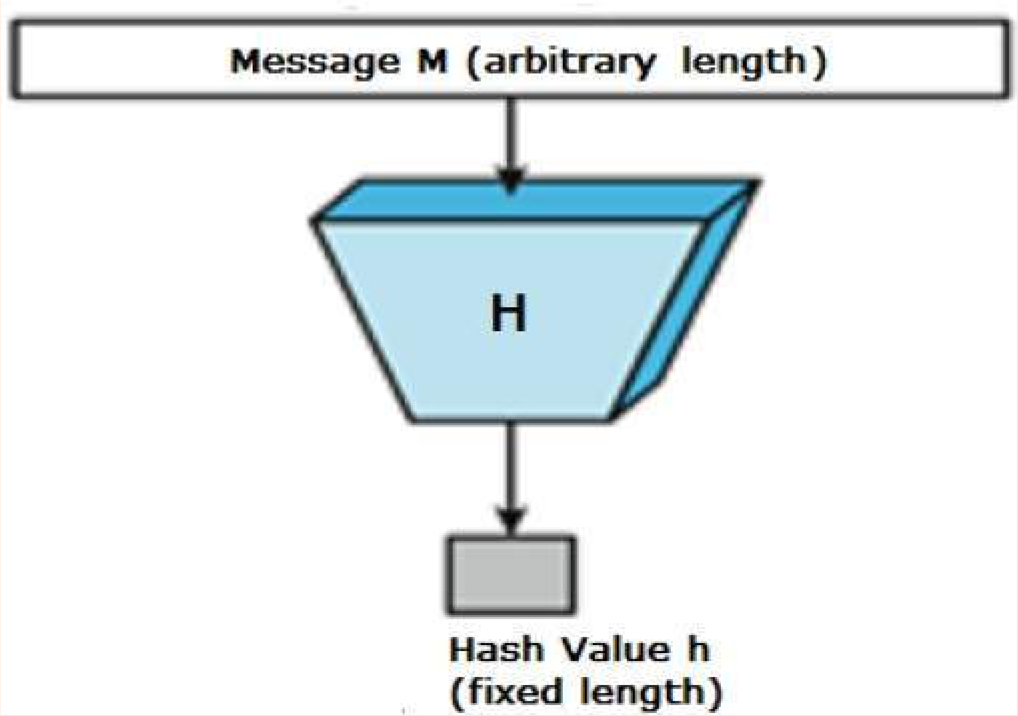
After this lecture you should be able to

1. Name some applications of cryptographic hash functions;
2. Understand the characteristics of crypto hash functions;
3. Differentiate the differences between a MAC and a digital signature;
4. Have some hands-on experiences on hash/MAC;

What is a hash function

- A hash function, sometimes known as a *message digest*, is an important cryptographic primitive. It can be used for verification in general.
- It takes an input of *any length* to a random fixed output.

Famous hash: **MD5, SHA1. SHA-256**

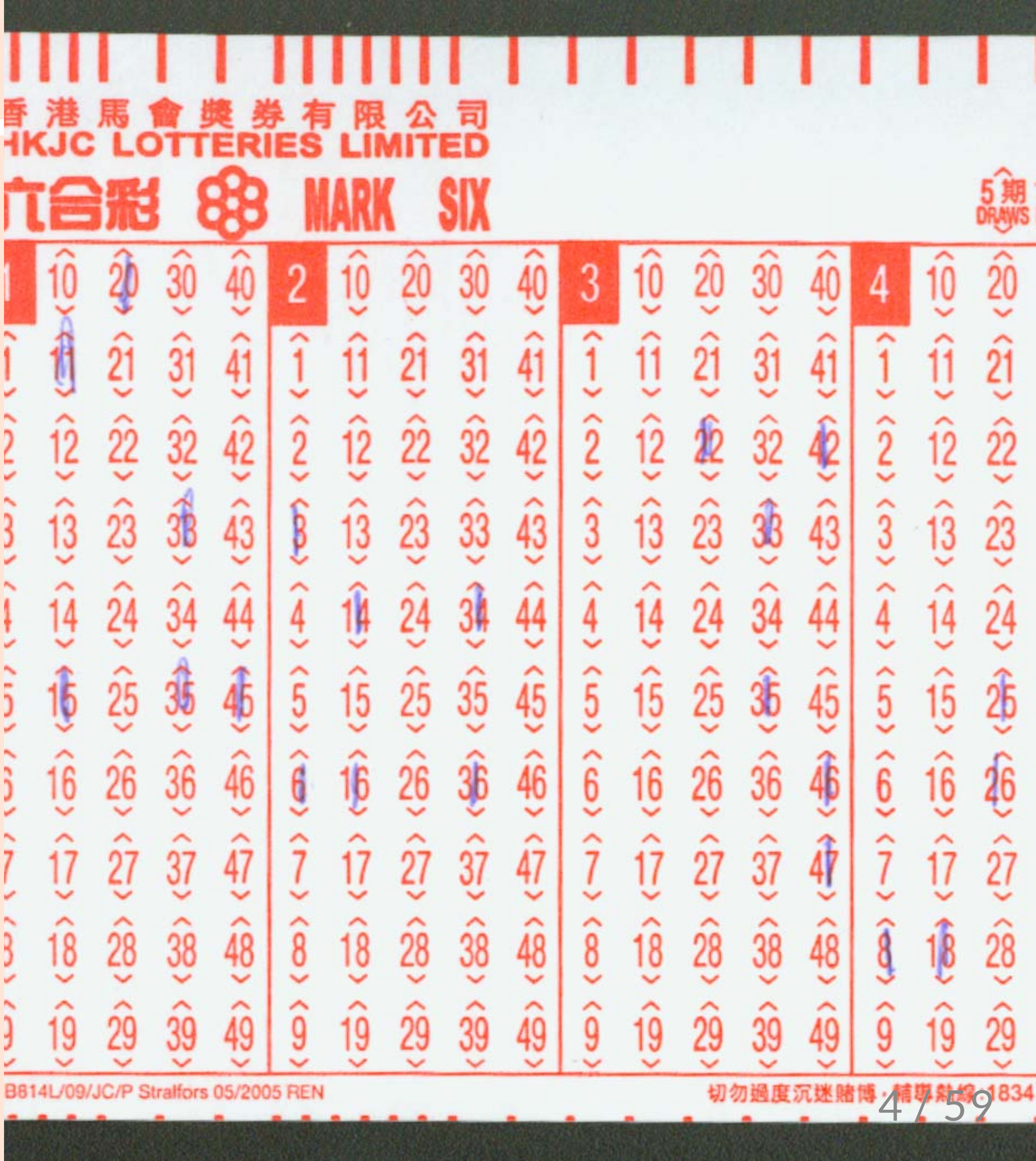


Motivation 1

- How to create a fair mark six online (host cannot cheat)?

or

- In general, how to host a fair game (e.g. paper-scissor-rock) ?



Motivation 2

How to send an *authentic* message?

- Authentic: really sent from someone and the message is not modified.

We assume the attacker has the full control of the network so that he/she can:

- Listen to the traffic
- Drop any package
- Insert any package
- Modify any package

Functions

A function F takes an input x and outputs y where $x \in \mathbf{X}, y \in \mathbf{Y}$. We denote that as $F : \mathbf{X} \to \mathbf{Y}$.

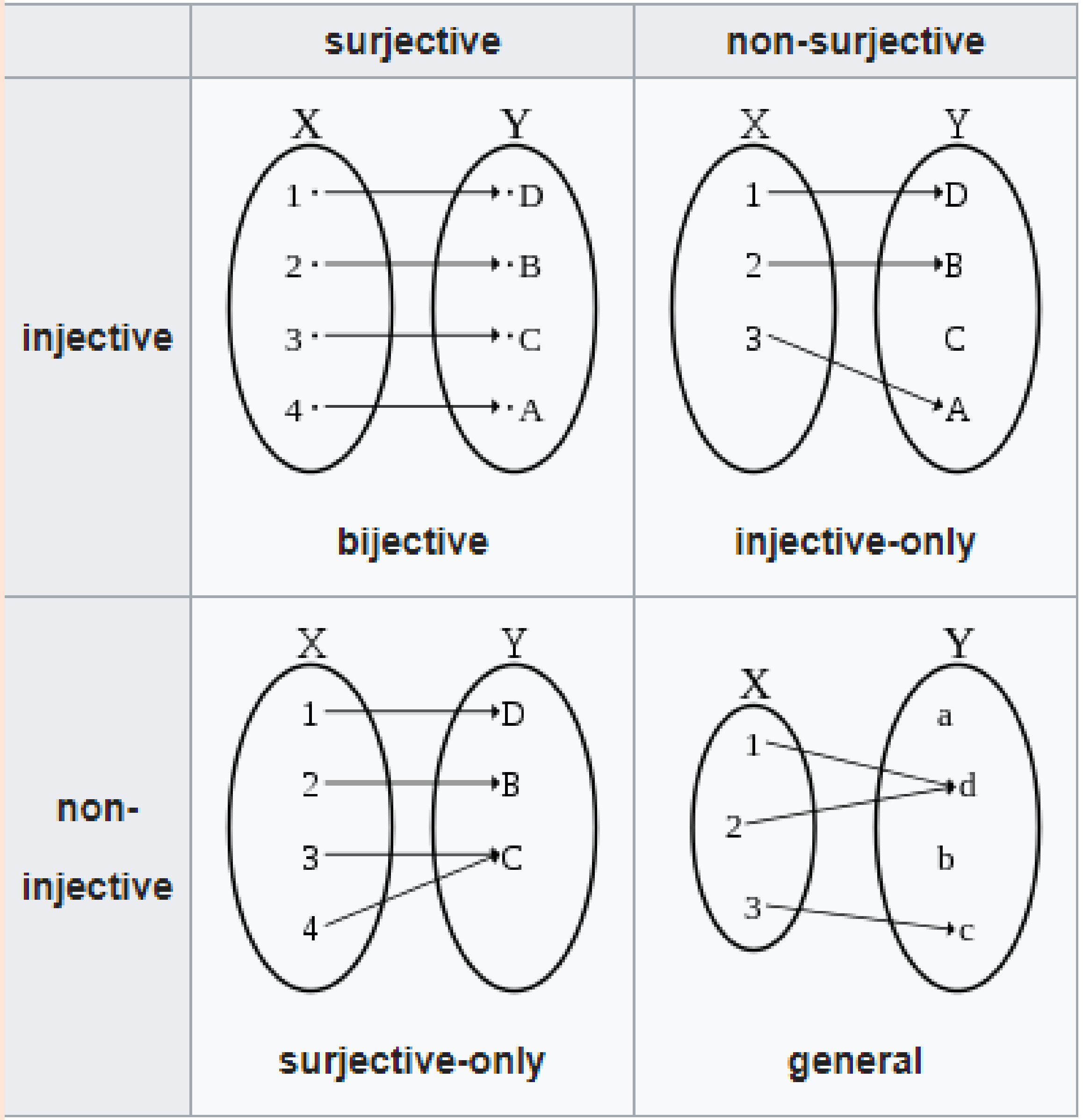
X is called the **domain**, Y is called the **codomain**.

e.g. Function grade $G : \mathbf{Students} \to \{A, B, C, D, F\}$

- Surjective (onto): if each element of the codomain has at least one element mapped from the domain.
- Injective (one-to-one): if each element of the codomain has at most one element mapped from the domain.
- Bijective (one-to-one and onto): both of above

Functions

- Which type of functions should an encryption function be?
- Which type of functions should a compression file (zip/jpeg) be?



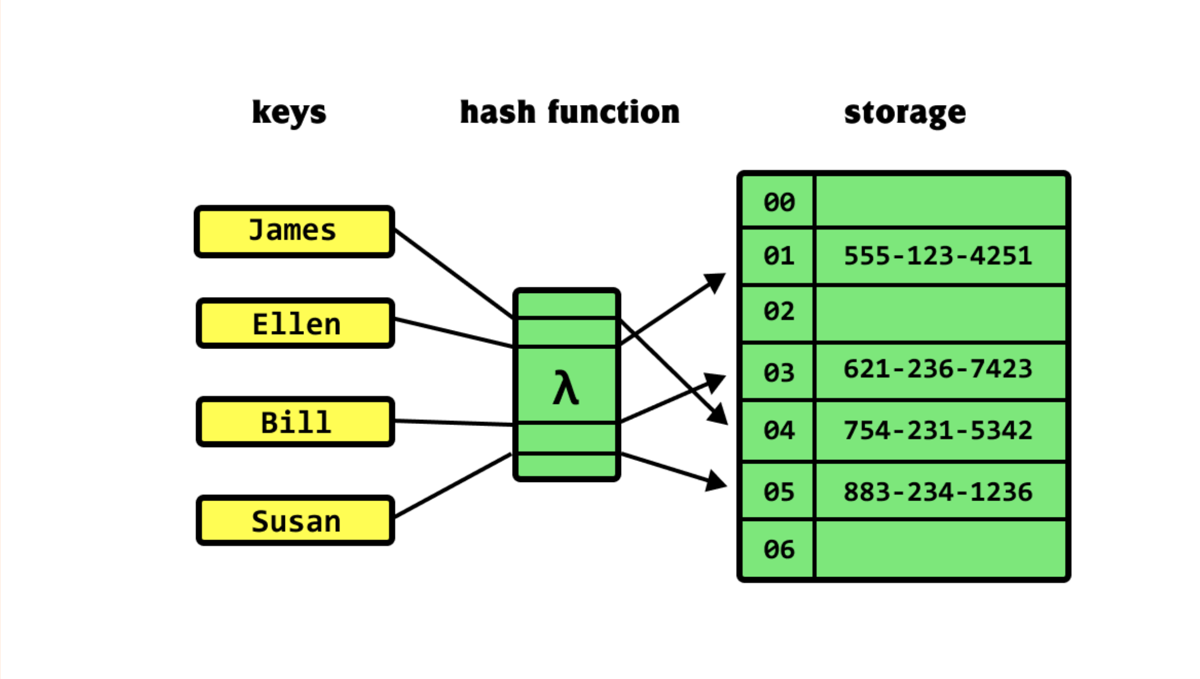
A Hash function

- A hash function is defined as a function that $F : \{0,1\}^m \to \{0,1\}^n$ where $|m| \in (0, \infty)$, n is a fixed number of output independent of the length of m.
  - No matter how long the input is, the output of the hash is always fixed.

🧑‍🏫 💬 Therefore, a hash function must (not) be a (surjective? injective? bijective?) function.

A regular (non-crypto) hash

- Hash functions are widely used in other non-security related area, such as algorithm (Bloom filter, hash table in linked list indexing, checksum, etc.)



Examples of a regular (non-crypto) hash

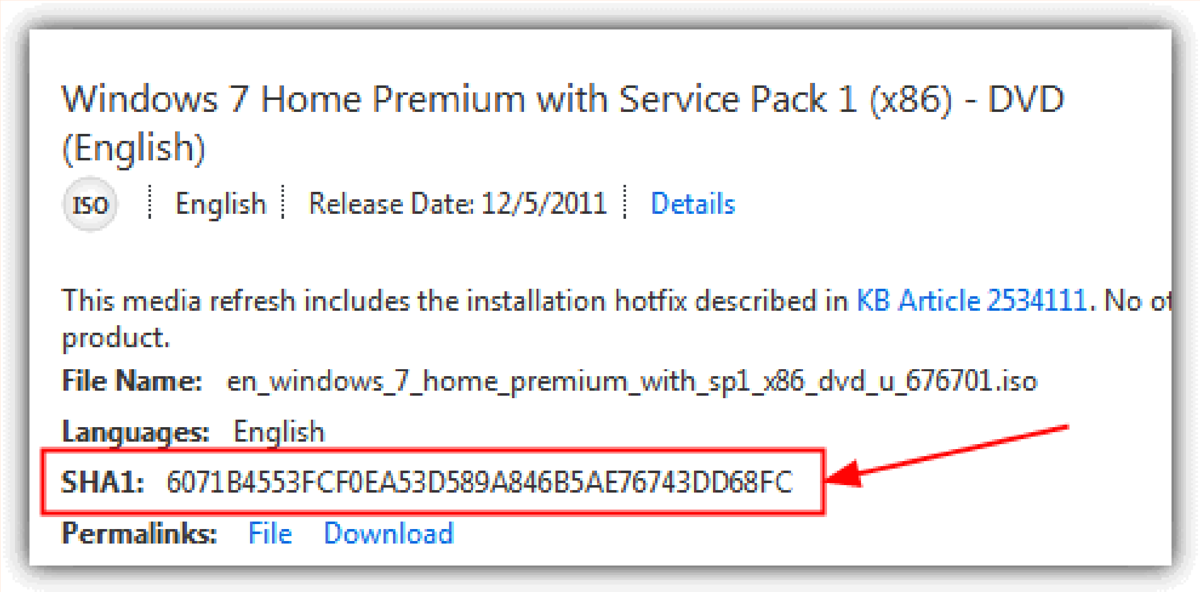
- $H(x) = x \mod n$
- $H(x) = x^3 + 1 \mod n$
- FNV-1

image credit: https://khalilstemmler.com/blogs/data-structures-algorithms/hash-tables/

A Cryptographic Hash function

A cryptographic hash function has much stronger requirement than a regular hash function. We want it to be impossibly hard to:

1. modify a message without changing the hash.
2. invert a given hash.
3. find two different messages with the same hash.



img: microsoft.com

❌ Modify a message without changing the hash

🧑‍🏫 💬 $H(m) \neq H(m')$ even if m and m' are almost identical

- It implies a cryptographic hash function will exhibit some avalanche effect.
- Changing even a single bit in the input will produce an avalanche of change through the entire hashed value.
- With a good probability that approximately 50% chance of bit will be flipped if a single bit of message is changed.
- More importantly it is impossibly hard to find such collisions or near-collisions.
- This is also known as *second-pre-image resistance*.

❌ Invert a given hash

🧑‍🏫 💬 Unable to compute m from $H(m)$

- It is also called *pre-image resistance*.
- The hash function is also said to be a **one-way function**: it is very easy to compute a hash for a given message, but it is very hard to compute a message for a given hash.

❌ Find two messages with the same hash

HONG KONG BAPTIST UNIVERSITY

🧑‍🏫 💬 Never find any pair m and m' such that $H(m) = H(m')$

- This is also known as *collision resistance*.
- Not to confuse with second pre-image resistance, collision resistance focus on the word *any*.

Can this be a cryptographic hash function?

$f(x) = x \mod p$, where p is a constant say 100.

- Property 1: Busted - if we have a message $m = 100$, $f(m) = f(m + 100) = f(m + 200)$.
- Property 2: Busted - given the hash value $y = 50$, we can invert the function as $y = 50 = f(50)$, thus $m = 50$.
- Property 3 : Busted - for example, $m = 1, m' = 101$ we have $f(m) = f(m')$.

Can this be a cryptographic hash function?

$f(x) = E_k(x)$ where k is a constant, E is an encryption function say AES.

- Property 2: Busted - given the hash value y, we can invert the function as $D_k(y) = m$.

Besides, x has an upper limit (support only n-bits).

Can this be a cryptographic hash function?

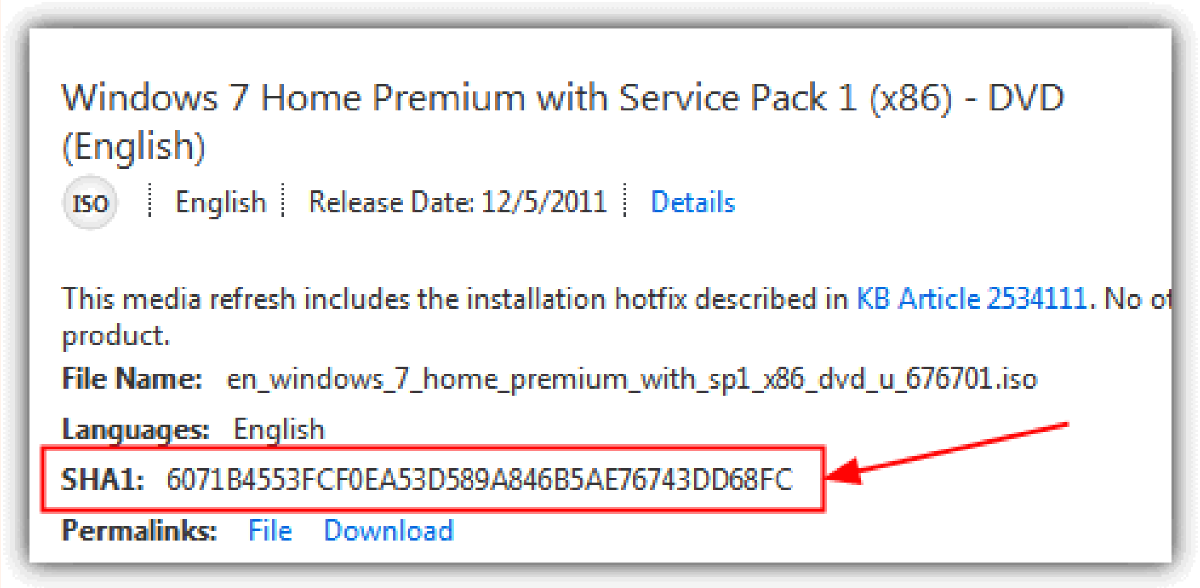
$f(x) = E_k(x_1) \oplus E_k(x_2) \oplus \cdots E_k(x_m)$ where $x = x_1||x_2||\cdots||x_m$ (assume proper padding applies on last block), k is a constant.

- Property 2: Busted, same reason.

Why these properties are important?

Let say f is used as crypto hash but not satisfies Property 3 - collision resistance.

- m : DVD ISO.
- m' : Virus.
- $f(m) = f(m')$



A vigilant user will download and run a virus even if he checks the message digest!

Why these properties are important?

Assume H is an ideal hash function. We can create a lottery scheme as follows: $y = H(m||s)$, where m is the lottery result, s is a long secret string.

- The lottery host published y and hide m and s.
- Customer cannot invert y so don't know m. (Property 2)
- To publish the result, the host released both m and s, everyone can verify. (Property 3)

Publish before the draw:

```
Hash: 3e5485a7b38a16be1642bffb867e10c2f5899a530aed586c21ca26b1d11f055c
```

After the draw, publish:

```
1 3 7 11 22 34 #23     ee145092452298fdc9102a
```

History of Hash functions

- 1992: **Ronald Rivest** proposed MD5 (Message Digest 5) and was specified in RFC1321.
- 1993: NSA published SHA-1
- 2001: SHA-2 was designed by NSA and adopted by NIST as a standard.
- 2004: **Wang Xiaoyun** et al. breaks MD5 with collision attacks.
- 2005: **Wang Xiaoyun** et al. breaks SHA-1 with only 2^{69} operations as compared with 2^{80}.
- 2007: NIST called for SHA-3 by a competition.
- 2011: NIST deprecated use of SHA-1.
- 2015: NIST published SHA-3.

[More story about Prof Wang, in Chinese]

Hash functions

- The internal structure of MD5/SHA-1/SHA-2 are all based on Merkle-Damgård construction while SHA-3 is totally different.
- SHA-2 contains a list of variants namely SHA-224, SHA-256, SHA-384, SHA-512. The naming is according to the size of hashed value.

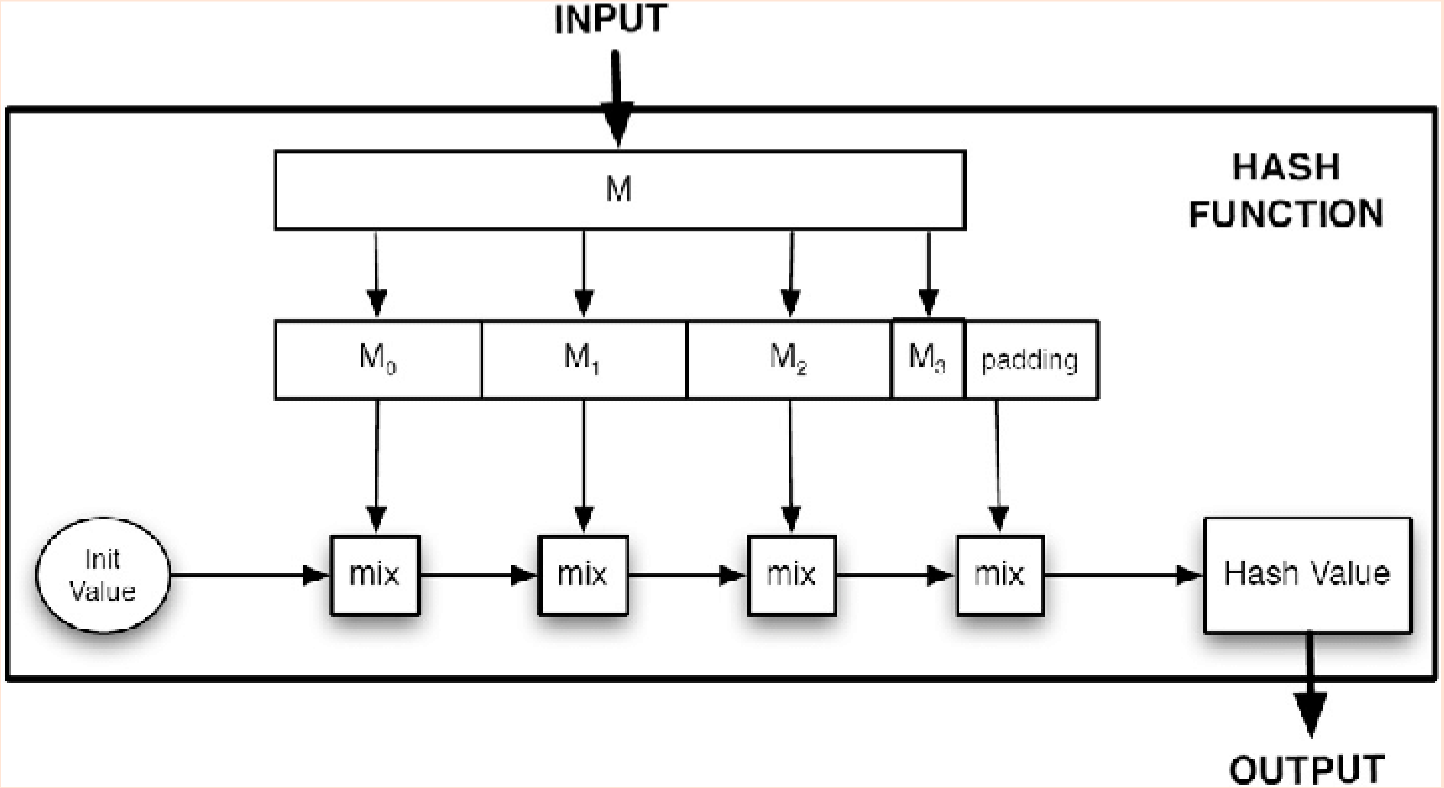
> 🧑‍🏫 💬 NIST has not yet decided to retire SHA-2 which is still widely used in many systems. May be this day will never come, or may be tomorrow.

Summary of SHA-2 Family

Hash function	Message size	Block size	Word size	Digest size
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

MD construction

It is proven that if mix function is collision resistant, then so is the hash function. MD5/SHA1/SHA2/Whirlpool are different by the function mix.



Security of Hash functions

Attacks on (Ideal) Hash functions

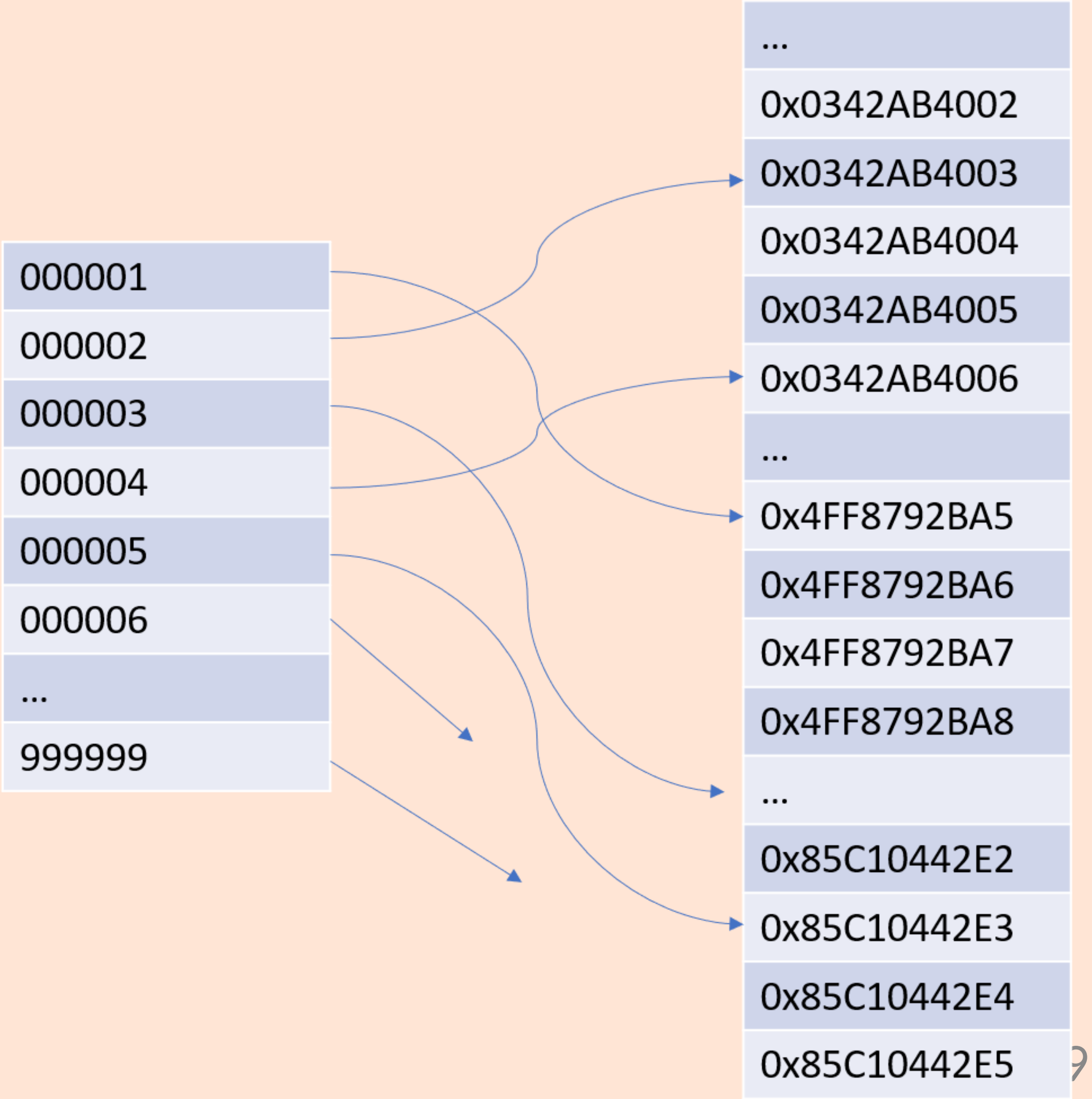
Three common way to attack a hash function:

- **Brute-force**
- **Second Pre-image**
- **Birthday attack**

Brute-force attack

To invert a hash value.

- **Brute-force attacks** simply enumerate the input space and find the possible value.
- If the input space is limited, brute-force attacks are feasible.
    - For example, if we have a hash value of a passphrase of 6 digit numbers, we can try 1000000 times with brute-force attack, regardless the length of hash output.
- If the input space is infinite or very very big it does not work.



Second Pre-image attack

- However if the hash output is not long enough, e.g. n-bits where $n = 30$ for example....
- It is possible that after trying for 2^n times, we find another input that has the same hash value as the original input.
  - This is called **Second Pre-image attack**.

Birthday attack (a.k.a Birthday Paradox)

> 🧑‍🏫 💬 How many has a birthday on 4th Jan?

- Need about 365/2 students in my class in order to find one (with 50% chance)

> 🧑‍🏫 💬 However, if I only want **any** two students have the same birthday?

$$
\begin{aligned}
\Pr(\text{No same birthday for n people}) &= 1 \times \frac{364}{365} \times \frac{363}{365} \cdots \times \frac{365-n}{365} \\
&= \frac{P_n^{365}}{365^n}
\end{aligned}
$$

Birthday attack (a.k.a Birthday Paradox)

香港浸會大學 HONG KONG BAPTIST UNIVERSITY

- With $n = 21$, probability is around 50%, $n = 30$, probability < 30%.
- Using this principle, performing $2^{n/2}$ operations will produce a good probability outputting two messages of the same hash (assume the output of the hash is n-bits long).

e.g. A 160-bit hash algorithm requires only 2^{80} operations to produce two messages of the same hash (with a good probability).

Hash function Application

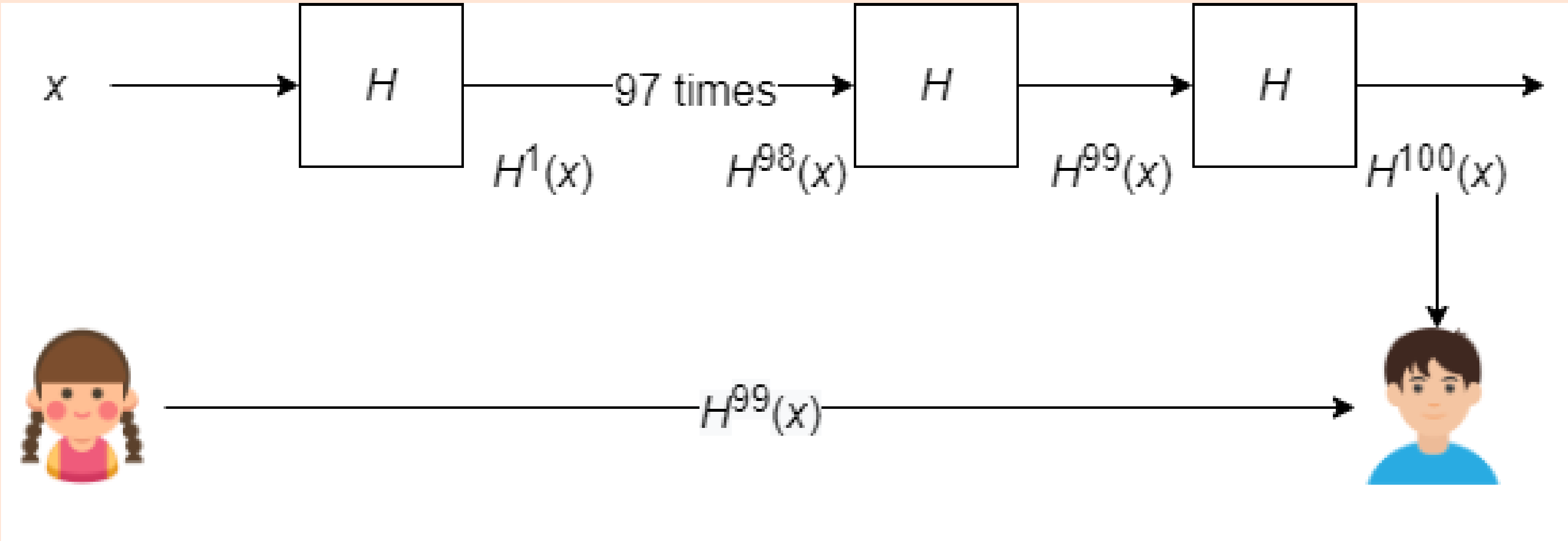
Password Storage:

- Storing plaintext: when you pass the "gateman", all passwords are yours!
- Storing $H(P)$: cannot invert the password directly, however, passwords usually has a low entropy ➡️ prone to password guessing attack.
- Storing $H(P||s)$: much better, as long as the attacker does not has the salt - but easily to see which users have the same passwords.
- 💡 Any better idea?

Hash chain

We define $H^n(x) = H^{n-1}(H(x))$, $H^1(x) = H(x)$, like $H^3(x) = H(H(H(x)))$

- Assume Alice has a secret x and securely register $H^{100}(x)$ to a server Bob.
- To login, Alice sends $H^{99}(x)$ to Bob.
- Next time, Alice sends $H^{98}(x)$ to Bob, and so on.



Hash chain - Example with numbers

x	$H^1(x)$	...	$H^{98}(x)$	$H^{99}(x)$	$H^{100}(x)$
0xF335	0x0EF2	...	0xDD05	0x9CB4	0xAF24

- Bank stores $H^{100}(x) = \texttt{0xAF24}$ in the server and store $s = \texttt{0xF335}$ and a counter $c = 100$ in trusted device (token). The bank then securely issue the token to a user.
- When the user wishes to login his online ebanking account, he supplies his username, password, and $H^{c-1}(x) = H^{99}(x) = \texttt{0x9CB4}$. The bank will verify the username, password, and check if $H(\texttt{0x9CB4}) = \texttt{0xAF24}$. If they match, the user will decrease the counter c by 1. The bank will replace the hash 0xAF24 by 0x9CB4
- Next time the user will login using $H^{98}(x) = \texttt{0xDD05}$.

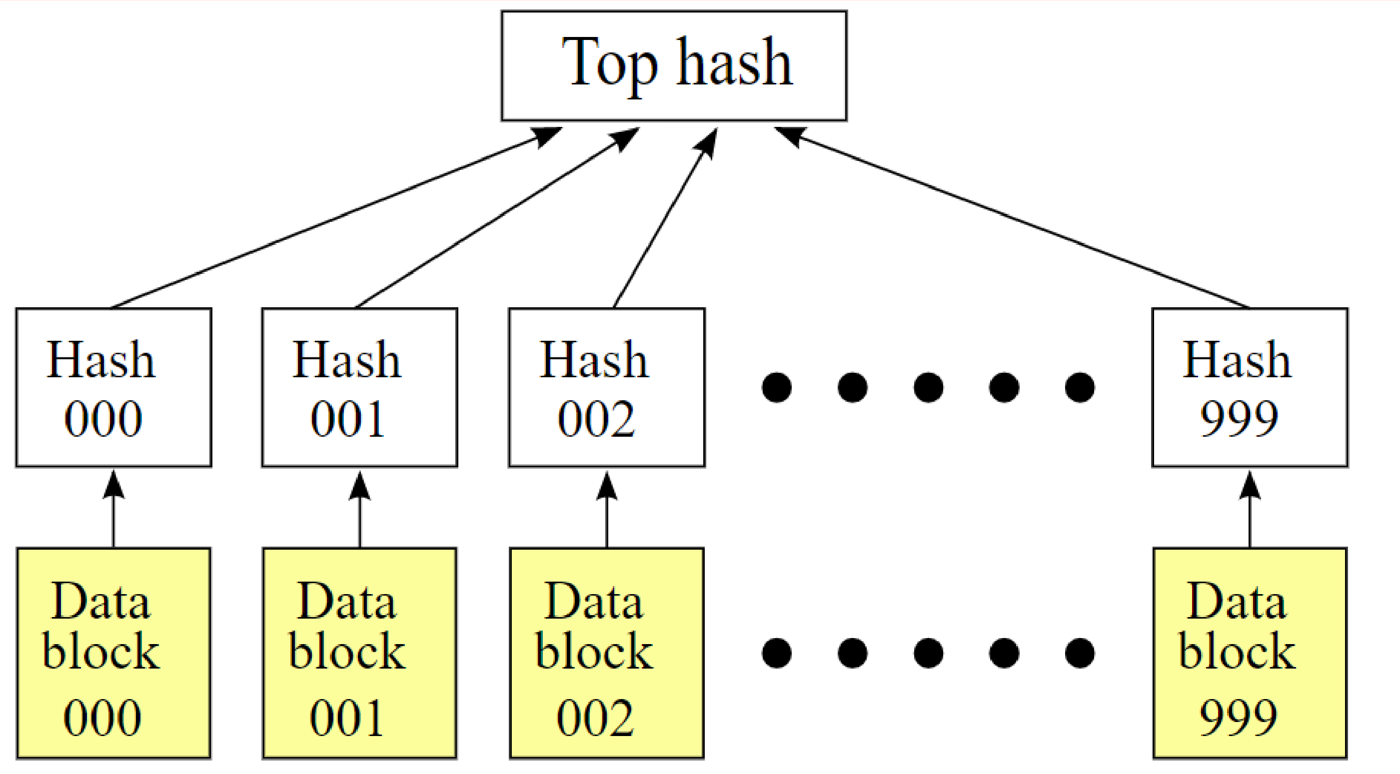
Hash chain

A hash chain can be used in applications that requires only one-way authentication, e.g., a security token.
Problems:

- Synchronization: how to keep track the number of hashes from the prover?
- Computation effort: the prover may need to compute a lot of hashes!
- Running out: 100 can run out very quickly, but what is a good number?

Hash list

A hash list is a variation of hash. By rehashing the hashes multiple blocks of data to create a top hash. Verifying the top hash can assert the correctness of data.



Hash list example

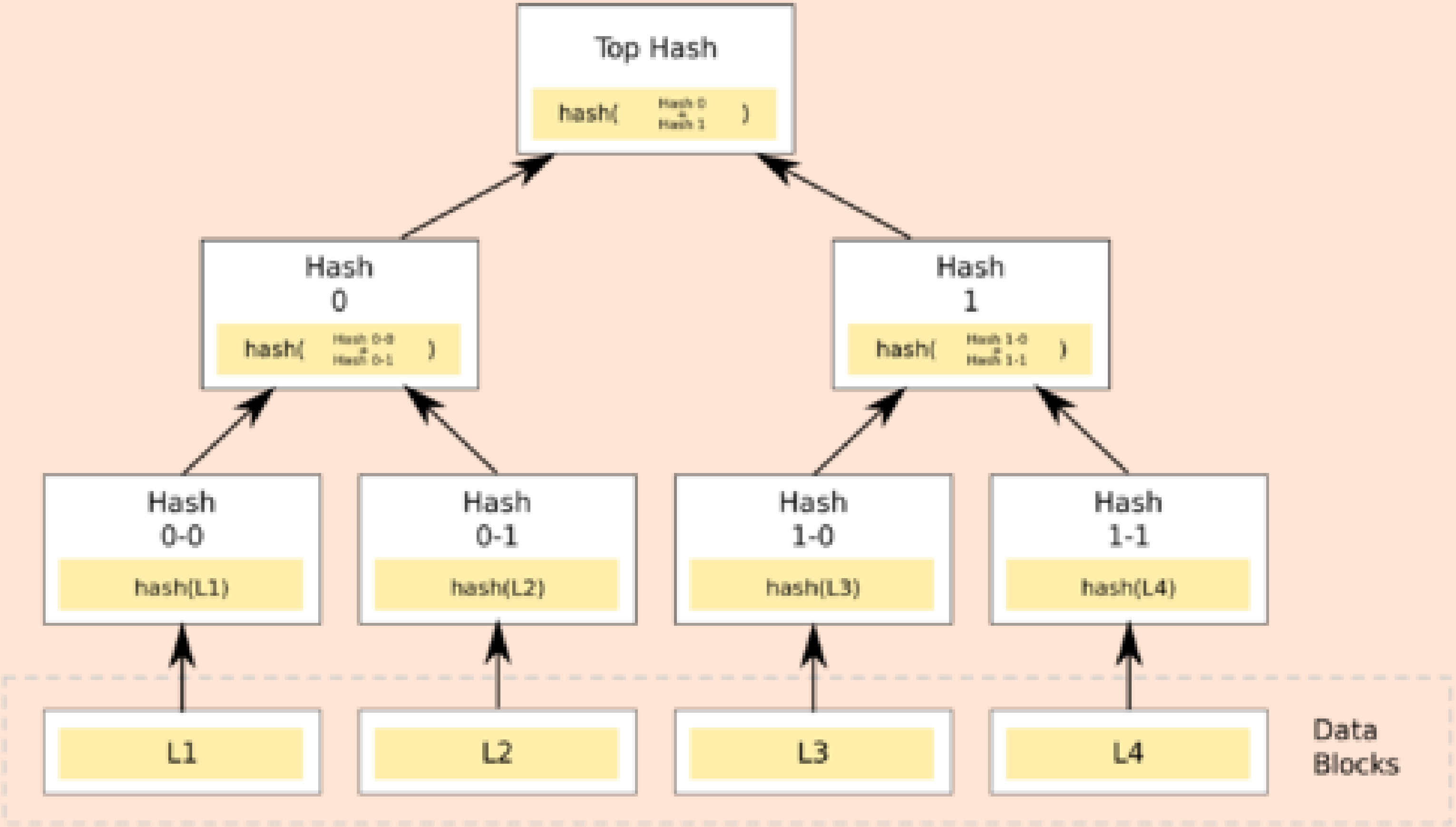
A distributed file system stores files in different locations. It can maintain the hash of the files in the following table.

File	Hash	Selected to download
file1	`0x342A`	✓
file2	`0x49BC`	✓
file3	`0xC52A`	
file4	`0xFF2E`	✓

- The system computes the hash of the hashes of the required file, $h = H(\texttt{0x342A}, \texttt{0x49BC}, \texttt{0xFF2E})$ and send it to the user.
- The user, after downloading the files, computes individual hash and computes the hash list to match it with h to assert the correctness of the files.

Hash tree (Merkle Tree)

- Another variation of hash is to arrange it into a tree structure. Each leaf node contains a block of data. Each parent hashes its children and eventually the root has a hash value.
- By verifying the root hash it can guarantee the data integrity of the entire tree.

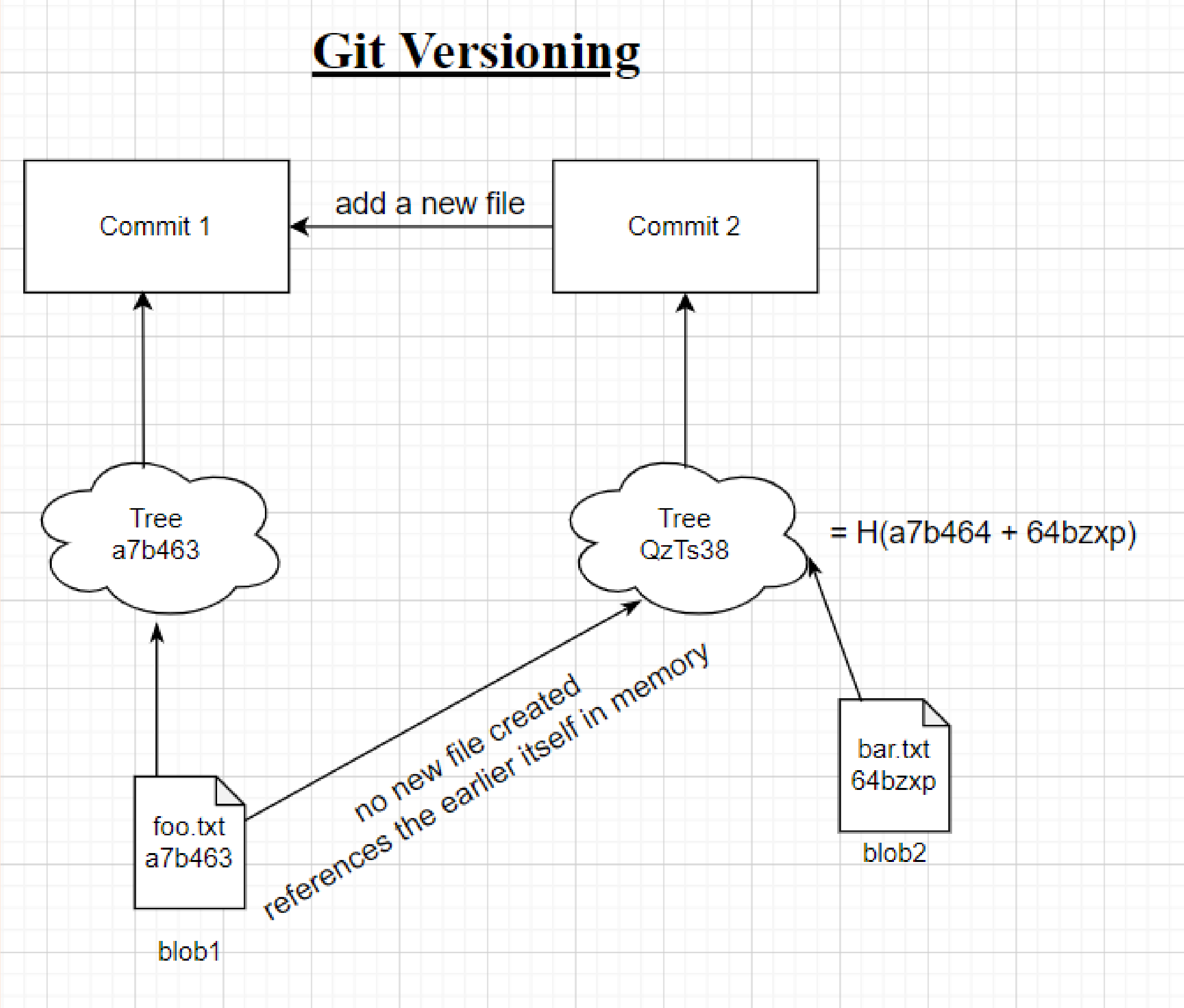


Usage: git, p2p system, Bitcoin

Hash tree (Merkle Tree) - Example Git

香港浸會大學 HONG KONG BAPTIST UNIVERSITY

- Every file in git is stored as a blob (binary large object). Blog is a file like object, with immutable raw data.
- If 2 files have the same content, then their hashes will be the same, so no new blob will be created even if the 2 files are in 2 different directories.
- Every commit object has 2 reference pointers, one pointing to its parent (previous) commit and the other referencing its merkle tree root hash.
- This merkle tree hash is computed by hashing all its "parent" nodes.



https://ieee.nitk.ac.in/blog/merkle-trees-and-their-application-in-git/

Encryption without authentication

香港浸會大學
HONG KONG BAPTIST UNIVERSITY

- Consider one time pad again, $E_K(M) = M \oplus K$. If the attacker change one bit, can the receiver detect it?
  - e.g. "How are you today?" ➡️ "How are yov today?"
- This error can be due to **Transmission Error** or **Attacks**.
- We need an *automatic* way which securely guarantees the message is intact.

💡 This is an example of transmission error where the BIG5 code of 血(A6E5) and 文(A4E5) differ by one bit.

https://www.gotrip.hk/網絡熱話/九巴-報站機-ctb062-rf-696927/

Message Authentication Code (MAC)

- A Message authentication code (MAC) is a small bit of information that can be used to check the *authenticity* and the *integrity* of a message.
- Don't be confuse as a checksum, which only check if there is any transmission error.
  💡 Why can't we use a checksum as a MAC?

> 🧑‍🏫 💬 Authenticity : from the designated person
> Integrity: unmodified

Checksum: say your HKID $LD_1D_2D_3D_4D_5D_6(C)$:

$$L \times 8 + D_1 \times 7 + D_2 \times 6 + D_3 \times 5 + D_4 \times 4 + D_5 \times 3 + D_6 \times 2 + C \mod 11 = 0$$

The need of MAC

There are three common ways to combine a ciphertext with a MAC.

1. Authenticate and encrypt. $C = E_K(P), t = MAC_{K'}(P)$, you send both ciphertext C and the tag t.
2. Authenticate, then encrypt. $t = MAC_{K'}(P), C = E_K(P||t)$, you send C only.
3. Encrypt, then authenticate. $C = E_K(P), t = MAC_{K'}(C)$, send both C and t.

K and K' can be the same or different

Remarks

- Authenticate-then-encrypt (2) provides more secrecy than Authenticate-and-encrypt (1).

1. $C = E_K(P), t = MAC_{K'}(P)$
2. $t = MAC_{K'}(P), C = E_K(P||t)$

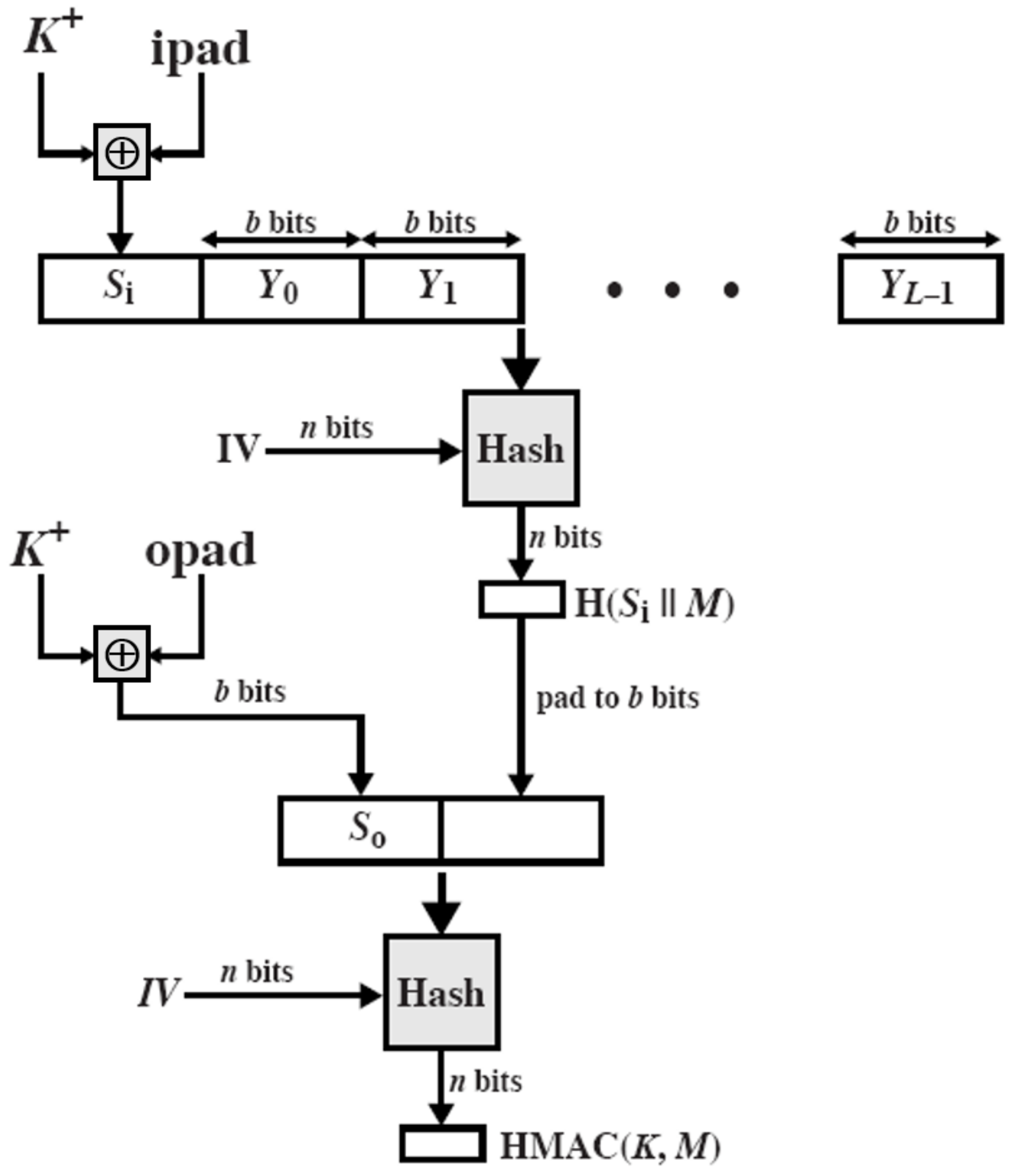
- Simply consider what if the same message is sent again.
- (1) produces the same t and is observed by the attacker. (2) can produces different C which attacker does not know they are encrypted from the same message.

Requirement of MAC

- *Computable*, i.e. very fast
- *Unforgeable*, i.e. cannot be forged by attacker
- *One-wayness*, i.e. message cannot be recovered from MAC.

Constructing MAC

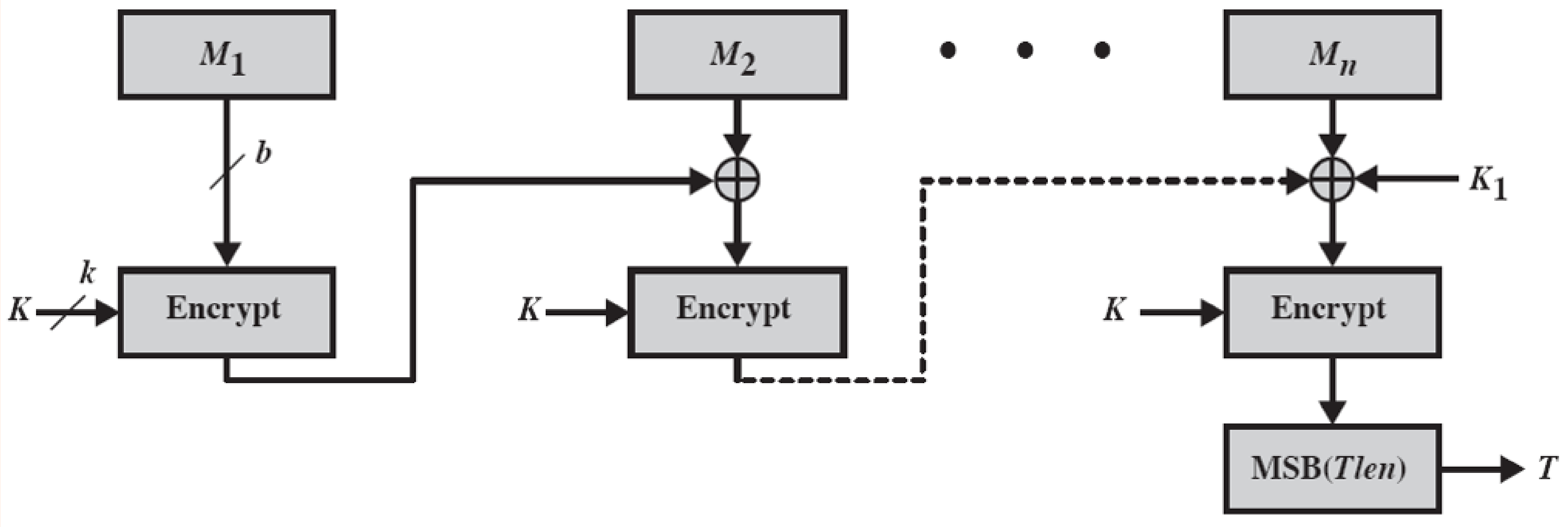
- **HMAC**: Hash-based Message Authentication Code (HMAC) is a standard in FIPS, used in IPSec and TLS.



Constructing MAC (con't)

CMAC: Cipher-based message authentication Code (CMAC) is also a standard defined in FIPS. It is a CBC-mode + a MAC.
CMAC provides both security and integrity while **HMAC** has only integrity.

Note: both encryption results and T are sent to the receiver.



Other MAC construction

Counter with cipher block chaining message authentication code (CCM): combining CTR model and CBC-MAC to provide confidentiality and authenticity of data.

- Used in WPA2/IPSec/TLS/BLE
- Require a unique IV (initial vector).

Galois/Counter Mode (GCM): also performs encryption and data integrity at the same time.

- Fast and secure.
- Used in IPSec/TLS.
- Require a unique IV (initial vector).

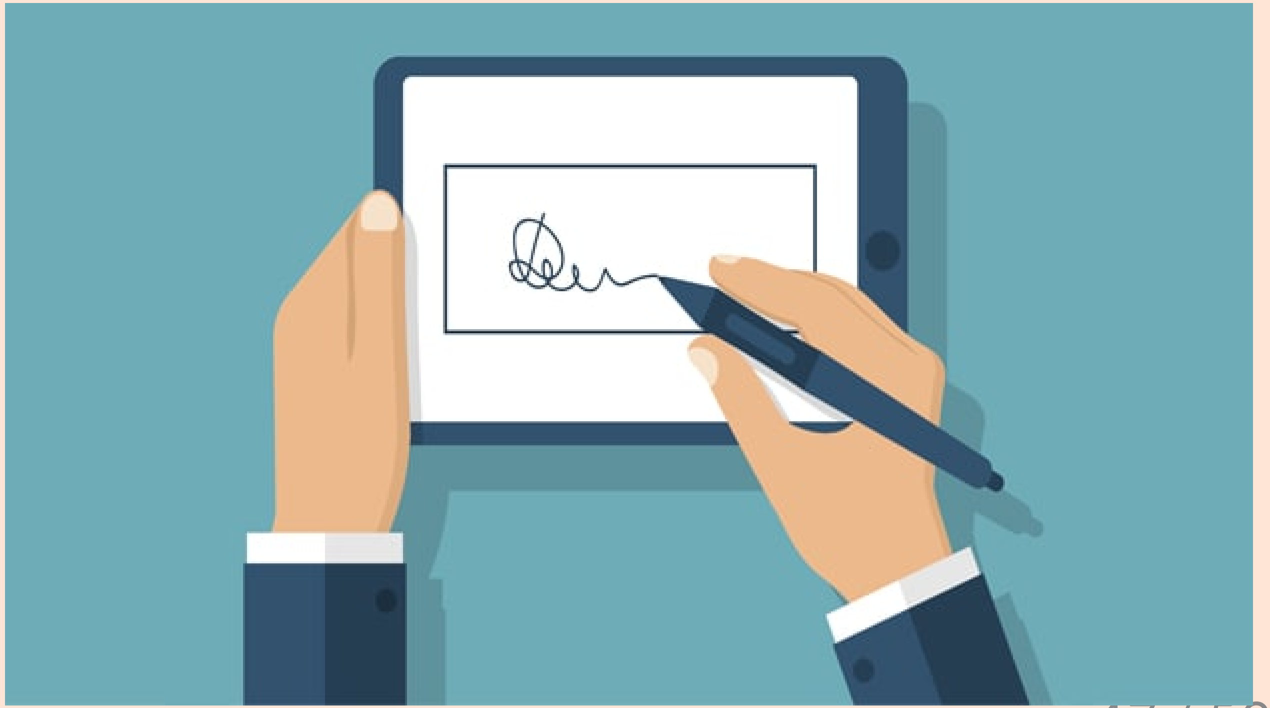
Digital Signature

Limitation of MAC

Like symmetric encryption, MAC requires a key to verify. Cannot verify if there is no key.
Once the key is released (like the lottery example), anyone can create the MAC.
Is there any asymmetric key version of MAC?

Digital Signature!

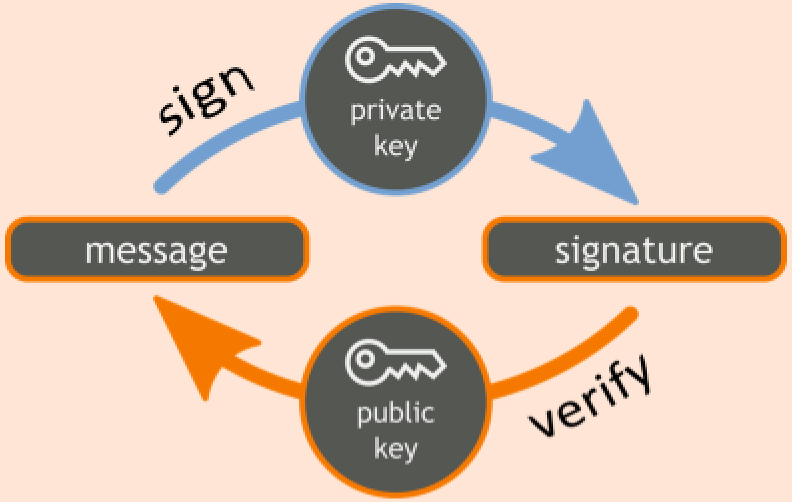
👉 This is actually not a digital signature.



Digital Signature

Challenge: require message to be publicly verifiable.
e.g.:

- Message from the University president;
- A will to give away your wealth to children and grandchildren;
- Contract
- Software Distribution



Fail Attempts on the challenge:

- MAC: does not work, since secret key is not shared by all. (or everyone can create the MAC)
- Public key encryption: does not work as everyone can send an encrypted message

Solution: Digital Signature!

Digital Signature

Recall a trapdoor function $T_k : \mathcal{A} \to \mathcal{B}$, it is easy to compute $T_k(a)$ but difficult to compute $T_k^{-1}(b)$ if k is unknown.

Idea: Use a trapdoor function to create a tuple $< m, \sigma >$ such that $T_k(\sigma) = m$

- Everyone will be able to compute/verify the signature.
- Attacker cannot forge a signature.
- Signer can sign on a signature easily.

Note: σ read as sigma.

🧑‍🏫 💬 You can *imagine* signature as the private key owner *decrypt* a plaintext and produce some *garbage output*. These *garbage output* is the signature. Why? Because only the key owner can produce it!

Example of Digital Signatures

- RSA-PSS - based on factorization
- DSA - based on discrete log
- ECDSA - based on ECC algorithm

ECDSA is shortest, fastest and more secure as we believe.

Comparing MAC and DS

The major difference between a MAC and a DS is that MAC is not publicly verifiable.

- However, when MAC is served as a **committment** (lottery example), it is publicly verifiable. In general, anyone has the key can create the MAC.

DS can be publicly verified.

- Anyone has the public key can verify the document is signed by a signer.

A MAC runs faster than a DS because a MAC requires a hash operation only.

- Meanwhile a DS takes a lot of computation effort (at least 100x time required).

Same as public key encryption scheme, a DS requires the authenticity of the public key. If a hacker replaces a public key with its own public key, the signature cannot be verified.

Hands-on work

To perform a hash function is very simple. We use `openssl dgst`

```
# openssl dgst --help
Usage: dgst [options] [file...]
  file... files to digest (default is stdin)
 -help                  Display this summary
 -c                     Print the digest with separating colons
 -out outfile           Output to filename rather than stdout
 -passin val            Input file pass phrase source
 -sign val              Sign digest using private key
 -verify val            Verify a signature using public key
 -prverify val          Verify a signature using private key
 -signature infile      File with signature to verify
 -hex                   Print as hex dump
 -binary                Print in binary form
 -hmac val              Create hashed MAC with key
 -mac val               Create MAC (not necessarily HMAC)
 -sigopt val            Signature parameter in n:v form
 -macopt val            MAC algorithm parameters in n:v form or key
 -*                     Any supported digest
```

Hands-on work - Hash

For example, we like to hash on a file `123.txt`

```
$ openssl dgst -sha256 123.txt
SHA256(123.txt)= 181210f8f9c779c26da1d9b2075bde0127302ee0e3fca38c9a83f5b1dd8e5d3b
```

- The output is 64 hex characters, which is 256-bits long.
- The hash will only process the file content but not the filename (different filename same content produces the same hash).

🧑‍🏫 💬 You can select other hash algorithms by replacing `sha256` by `sha512`, `sha1`, `sha3-224` etc.

Hands-on work - Hash

Adding more files after the command will hash the file separately

```
$ openssl dgst -sha256 123.txt abc.txt
SHA256(123.txt)= 181210f8f9c779c26da1d9b2075bde0127302ee0e3fca38c9a83f5b1dd8e5d3b
SHA256(abc.txt)= fb2431e305a0859203d2b2cd2b1d9a2fc1efaa860d2f3e918612ebdf4361a67f
```

If you want to produce only one hash for multiple files, you can use `cat` and `|`

```
$ cat 123.txt abc.txt | openssl dgst -sha256
(stdin)= deb96a7ec6dde7a2415566e39e598d77553079bc715d034c8d3c65147271e23a
```

🧑‍🏫 💬 `cat` means viewing the text file.

Hands-on work - HMAC

You can produce a HMAC (with key) using the following command

```
$ openssl dgst -hmac my_secret_key -sha256 123.txt
HMAC-SHA256(123.txt)= b320f0ba0d1a83ab95002ac6258ae1472e3cb26c170c21800c99fadfeb0029f6
```

By changing the value `my_secret_key` (which is the ASCII key that you use to generate the HMAC), it will produce a different result

```
$ openssl dgst -hmac top_secret -sha256 123.txt
HMAC-SHA256(123.txt)= 616ce6d19a70821a61f9f589fda2fa5ae0fcb53f76af5f63f7a4553ea171ceca
```

Hands-on work - Digital Signature

- To work with a digital signature, you need to generate a pair of public-private key (either RSA or EC key).
- Assume your private key is `private.pem` and your public key is `public.pem`. Generate your signature by

```
openssl dgst -sign private.pem -sha256 -out signature.sig file_to_sign
```

- The signature will be generated to the file `sigature.sig`.

> 🧑‍🏫 💬 Remember, you need the private key to sign and you need the public key to verify the signature.

Hands-on work - Digital Signature

HONG KONG BAPTIST UNIVERSITY

To verify the signature, we need to obtain the public key from the signer.

```
openssl dgst -verify public.pem -sha256 -signature signature.sig file_to_sign
```

The command will report it is success or not.

> 🧑‍🏫 💬 You can try to check the length of the signatures generated by a RSA private key and by a ECC private key. ECC will be much shorter.

Summary

- Hash
- MAC
- Digital Signatures

Quick Questions

1. Can a hash be inverted?
2. Can a MAC be inverted?
3. Does a hash need a key?
4. Can a hash be brute forced?
5. Does producing a MAC needs a public key, private key or a secret key?
6. Does producing a Digital Signature needs a public key, private key or a secret key?