Blockchain 101

1# Security Fundamental Concepts

About me

- My name is Miguel Garcia

- My background is on intrusion-tolerant systems

- I am a cryptocurrency skeptic, but I am a Blockchain enthusiast (however, I don't think it solves all the problems in the world)

About Blockchain101

- This is not a course on cryptocurrencies, I will not teach you how to become a bitcoin millionaire or create a mining pool.

- This course is a about the blockchain technology – which happens to be the core of the Bitcoin.

- It should provide you theoretical and practical knowledge about Blockchain.

Blockchain101: contents

1. <u>Security Fundamental Concepts</u>

2. Secure Distributed Systems

3. Blockchain in a Nutshell

4. Assembling the pieces: Blockchain prototype

Blockchain core parts

Blockchain technology has three key parts:

Security	Structure	Network
Cryptography	Ledger	P2P
Fault-tolerance		
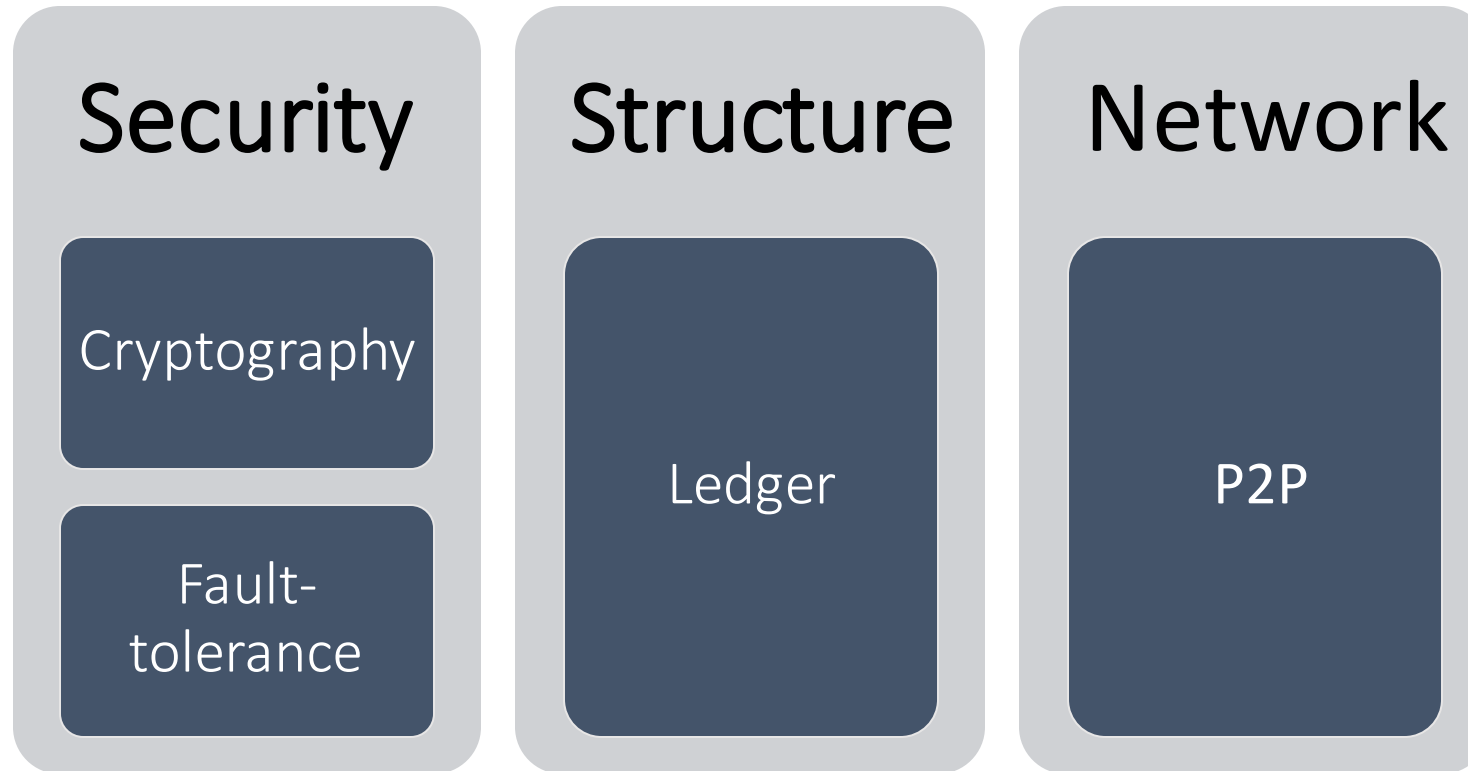
Security Properties

A system is said secure if it guarantees the following properties

- Confidentiality

- Integrity

- Authenticity

- Availability

Security Properties

Confidentiality:

- Only authorized people can see the data

- Imagine that we have Bob, Alice and Trent
- Bob sends as letter to Alice and only Alice should read the text
- Therefore, we need to ensure that Trent cannot read the text

Security Properties

Integrity:

- Data cannot be modified without authorization or undetected.

- Imagine that we have Bob, Alice and Trent
- Bob sends as letter to Alice
- Trent takes the letter and change some words
- The data should avoid this modification OR Alice should be able to detect this modification

Security Properties

Authentication:

- The one that sends data can prove that he or she is the author of the data

- Bob sends as letter to Alice

- Alice knows that Bob sent the letter

- Trent cannot send a letter to Alice saying that he is Bob

Security Properties

Availability:

- Data or services should be always available

How can we guarantee these properties?

Cryptography

- Long time ago, Cryptography was synonym of encryption

- The idea was to send text messages that adversaries couldn't read

- The main classical ciphers were based on **transposition**:
  - *Hello world   ehlol owrdl*

- And **substitution**:
  - *Hello world   Ijmmp xpsme*

- A more advanced cipher was the Caeser Cipher, that applied **substitution** cipher based on a fixed number of shifts

Cryptography

- During the WWII the **Enigma Machine** was one of the weapons of the Nazis that could communicate without the allies discover the meaning of the messages

- Thanks to **Allan Turing** that cracked the Enigma Machine it was possible to uncover the encoded messages and defeat the Nazis
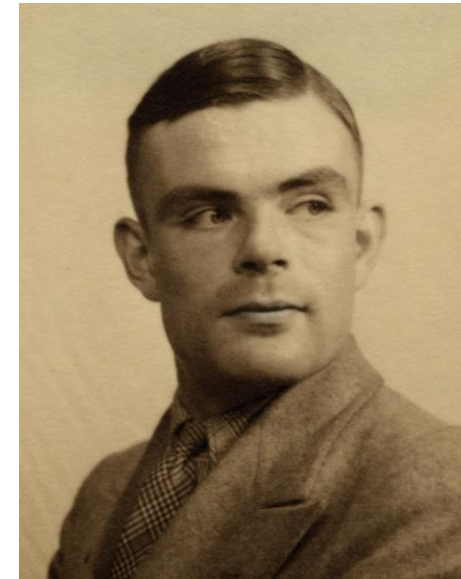
Cryptography (computer era)

Terminology

Method	Function	Output
Encrypt	*E(Clear Message, Key)*	Ciphered Message
Decrypt	*D(Ciphered Message, Key)*	Clear Message
Hash	*H(Message)*	Message Digest
Sign	*S(Message)*	Signture
Validate	*V(Signature)*	Boolean (true or false)

Symetric-key Cryptography

- Symetric-key cryptography uses the same key to encrypt and decrypt
- Therefore, is also called secret-key or shared-key cryptography

Alice

Hello

A!D$#

A!D$#

Bob

Hello

Symetric-key Cryptography

Properties:

- **E(K,M) = c** , then **D(K,c) = M**

- **D(K, (E(K,M)) = M**

Attributes:

- Given **E(K,M) is infeasible to find M without K**

- Given **E(K,M)=c is infeasible to find K**

Symetric-key Cryptography

E (Key, Data) = ciphered data

Encrypt

Variable size **key**

VByCFfclNy+OXk96fA3PpnlG03Rgm+rItxLTN8N0hXY+BuSpYwpjEItbFhgUE33RN0qlxLbIB0FnH/xqs9yWn6h7GlZlEo/cpSNlpfU7Ezd7XWTjtt2JlpHidIvBGYhAkpA5OhWp8GgryVB+..........
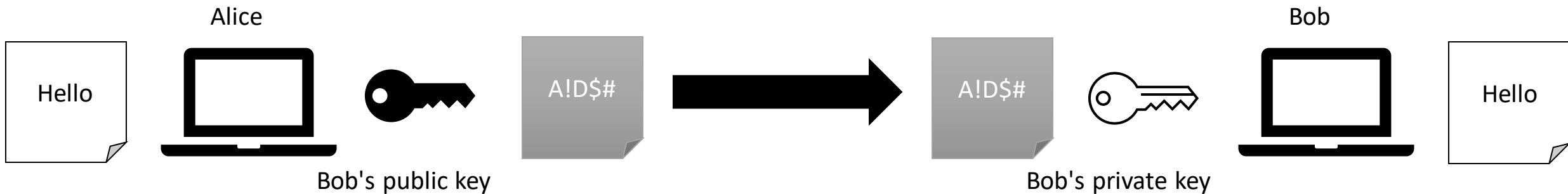
Input **data**: text, bytes

This a clear text message

Ciphered data: bytes

ASDadkasERLDLASck495$%!"#

Symetric-key Cryptography

D(Key, Ciphered data) = data

Decrypt

Variable size **key**

VByCFfclNy+OXk96fA3PpnlG03Rgm+rItxLTN8N0hXY+BuSpYwpjEItbFhgUE33RN0qlxLbIB0FnH/xqs9yWn6h7GlZlEo/cpSNlpfU7Ezd 7XWTjtt2JlpHidIvBGYhAkpA5OhWp8GgryVB+……….
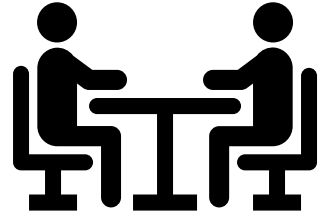
Input **data**: bytes

ASDadkasERLDLASck495$%!"#

Clear data: text, bytes

This a clear text message

Asymetric Cryptography

- Asymetric-key cryptography uses two keys

- Public key is used to encrypt and private key to decrypt

- Alice uses Bob' public key to encrypt, Bob uses its private key to decrypt

Alice

Hello

Bob's public key

A!D$#

Bob

Bob's private key

A!D$#

Hello

Discussion slide

Could we use the private key to encrypt and the public key to decrypt?

Asymetric Cryptography

Properties:

- **E(Ku,M) = c** , then **D(Kp,c) = M**

- **D(Ku, (E(Kp,M)) = M**

Attributes:

- Given **E(Ku,M) is infeasible to find M without Kp**

Asymetric-key Cryptography

E (Public Key, Data) = ciphered data

Encrypt

Variable size **key**

VByCFfclNy+OXk96fA3PpnlG03Rgm+rItxLTN8N0hXY+BuSpYwpjEItbFhgUE33RN0qlxLbIB0FnH/xqs9yWn6h7Gl
ZlEo/cpSNlpfU7Ezd7XWTjtt2JlpHidIvBGYhAkpA5OhWp8GgryVB+……….

Input **data**: text, bytes

This a clear text message

Ciphered data: bytes

ASDadkasERLDLASck495$%!"#

Asymetric-key Cryptography

D(Private Key, Ciphered data) = data

Decrypt

Variable size **key**

ASDldoweifkf++33$32llapjrlçajERKFJas´++r+a#$$klasdçaspjpamkasligellasllERASllfiElfELasṕro4lc-
.fo39$$332ças004#"o23"0400lliff//flaslrpprp+asçept.....
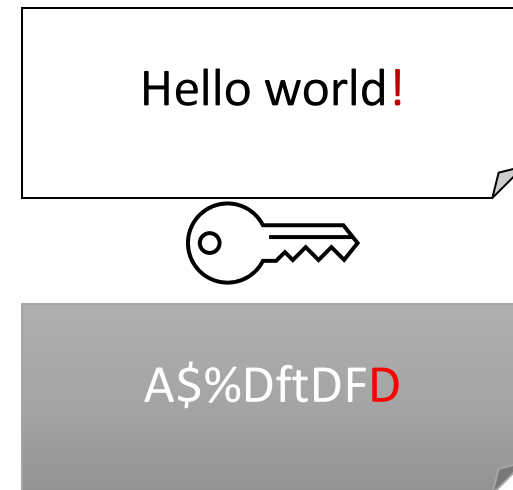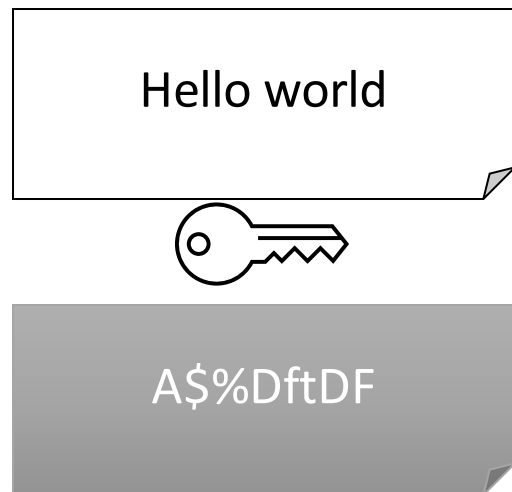
Input **data**: bytes

ASDadkasERLDLASck495$%!"#

Clear data: text, bytes

This a clear text message

Cryptography properties

Confusion:

- The adversary **should not** detect changes in the ciphered text if we change one symbol in the clear text.

- Therefore, it should be a complex relation between the clear text and the ciphered text

Hello world

A$%DftDF

Hello world!

A$%DftDFD

?
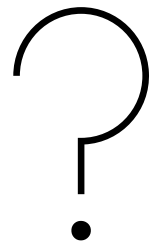
Cryptography properties

Diffusion:

- The clear text information should be spread all over the ciphered text

- Therefore, an attacker must collect a lot of ciphered text data to figure out how the cipher works

Hello world

A$%DftDF

Hello world!

%TdasDLD
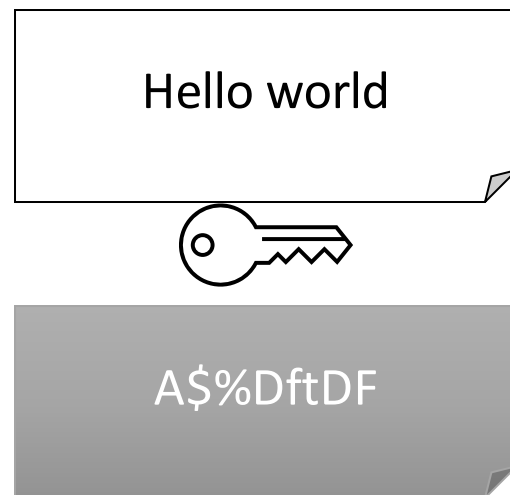
Symetric-key vs Asymetric-key Cryptography

Symetric-key Cryptography:

- The security relies on the shared key, once is lost the security is broken

- Key distribution: (n (n-1) / 2) keys for n participants

- 10 participants need 45 keys, one key for each pair of participants

Asymetric-key Cryptography :

- There is no need for key distribution, the public key is public so anyone can use it to encrypt data

- The security relies on the protection of the private key

- Is slower than symetric-key cryptography
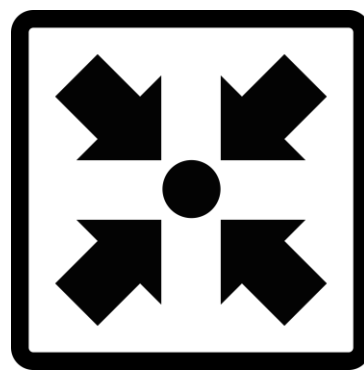
Meeting point slide

- To guarantee confidentiality, we need cryptography

- In particular, by using secret-key or public-key cryptography

- Secret-key cryptography uses one shared key to encrypt and decrypt

- Public-key cryptography uses the destination public key to encrypt, and destination private key to decrypt – only the private-key owner can decrypt the message

- Symetric-key cryptography = Secret-key cryptography

- Asymetric-key cryptography = Public-key cryptography

We already know how to guarantee confidentiality.
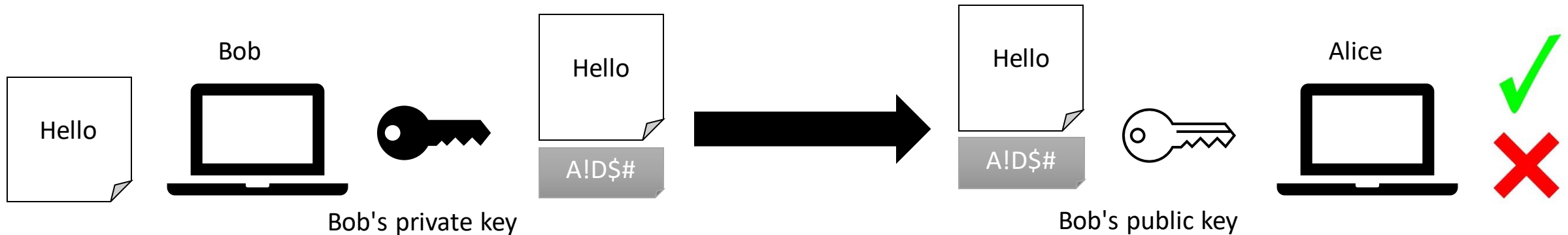How can we guarantee authenticity and integrity?

Digital Signatures

- Digital signatures are very similar to asymetric-key cryptography, however, **instead of using public key to ecnrypt we use private key to sign**

- **Very important:** Signing a message is not the same as encrypting a message

- **Another important note**: it is assumed that only the owner has access to the private key

- **Signatures** (alone) do not provide confidentiality but **provide authenticity and integrity** (and other properties)

Digital Signatures extra properties

- **Authenticity**: who signed the message is uniquely identifiable by his/her signature.

- **Tamperproof:** who signed, signed deliberatily

- **Integrity:** A valid signature guarantees that a message is not modified without being noticed

- **No-reuse:** A signture, or part of it, is not reusable in another message

- **No-repudiation:** the signer cannot deny his/her signature

Digital Signatures

- Sender: **sign(M, Kp)** = **S** (signture)

- Sender: sends message and signature

- Receiver: **verify(M, S, Ku)** = **true or false**

Bob

Hello

Hello

A!D$#

Bob's private key

Hello

A!D$#

Bob's public key

Alice

Digital Signatures: RSA

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R.L. Rivest, A. Shamir, and L. Adleman[*]

- It was published in 1978 by Rivest,, Shamir and adleman (RSA)
- RSA can be used for both cipher and signatures
- It is the most used algorithm on the web.
- É muito utilizado em aplicações de comércio electrónico

Digital Signatures:
RSA: Key generation

- Public Key: *(n, e)*
- Private Key: *d*

- **Step 1**: Choose two large prime numbers *p* and *q*

Digital Signatures:
RSA: Key generation

- Public Key: *(n, e)*
- Private Key: *d*

A prime number is a natural number greater than 1 that is not a product of two smaller natural numbers.

- **Step 1**: Choose two large **prime** numbers *p* and *q*

Digital Signatures:
RSA: Key generation

- Public Key: *(n, e)*
- Private Key: *d*

A prime number is a natural number greater than 1 that is not a product of two smaller natural numbers.

- **Step 1**: Choose two large **prime** numbers *p* and *q*

p = 43
q = 47

Digital Signatures:
RSA: Key generation

- Public Key: *(n, e)*
- Private Key: *d*

- **Step 2**: Compute n = p x q

n = 43 x 47
n = 2021

Digital Signatures:
RSA: Key generation

- Public Key: *(n, e)*
- Private Key: *d*

- **Step 3**: Compute ϕ(n)

$$\phi(n) = \phi(p-1)(q-1)$$

Digital Signatures:
RSA: Key generation

- Public Key: *(n, e)*

- Private Key: *d*

- **Step 3**: Compute *ϕ*(n)

$$\phi(n) = \phi(p-1)(q-1)$$
$$\phi(2021) = (43-1)(47-1)$$

Digital Signatures:
RSA: Key generation

- Public Key: *(n, e)*
- Private Key: *d*

- **Step 3**: Compute *φ*(n)

$$\phi(n) = \phi(p\text{-}1)(q\text{-}1)$$
$$\phi(2021) = (43\text{-}1)(47\text{-}1)$$
$$\phi(2021)=1932$$

Digital Signatures:
RSA: Key generation

- Public Key: *(n, e)*

- Private Key: *d*

> The greatest common divisor (gcd) of two integers is the largest positive integer that divides each of the integers.
> For example, the gcd of 8 and 12 is 4

- **Step 4**: Select *e, e is the public exponent*

Such that $e \in \{ 1, \ldots, \phi(n-1) \}$
gcd(*e*, 1932) = 1
For example: *e* = 155

Digital Signatures:
RSA: Key generation

- Public Key: *(n, e)*

- Private Key: *d*

- **Step 5**: Compute private key *d*

d x e ≡ 1 mod φ(n)
d x 155 ≡ 1 mod 1932
d = 1583

Digital Signatures:
RSA: Sign

- Now Bob wants to send a signed message to Alice
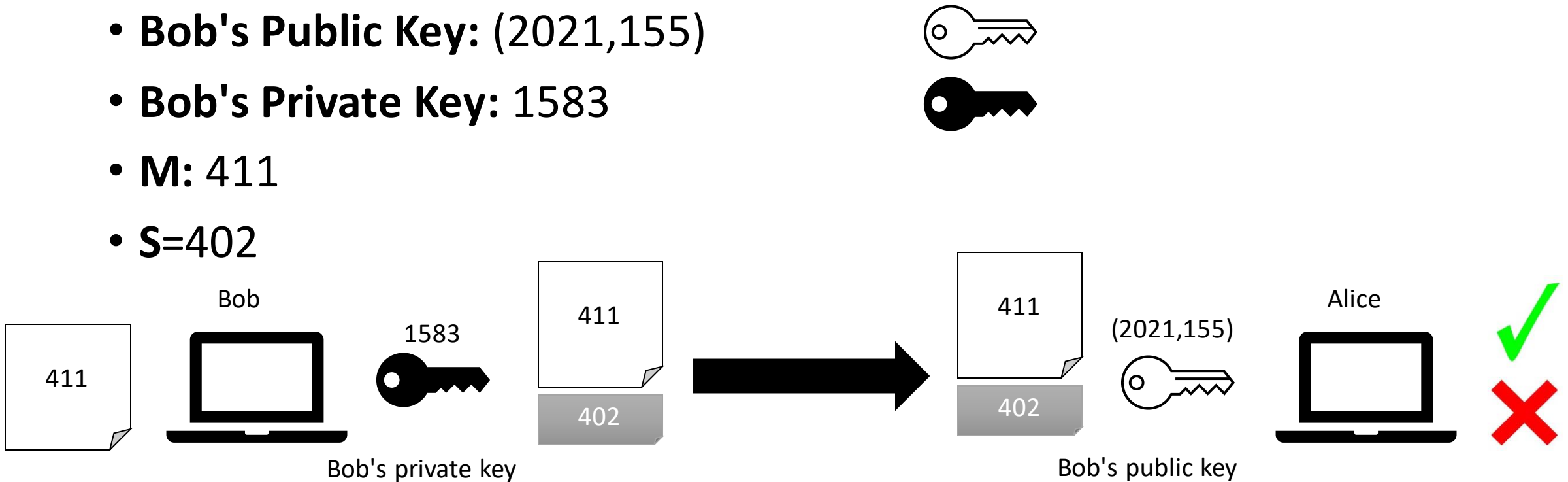- **Bob's Public Key:** (2021,155)
- **Bob's Private Key:** 1583

- The message $m \in \{ 1, \ldots, n\text{-}1 \}$, e.g., *m = 411*

Remember that we are using small prime numbers for the sake of simplicity.
n-1 should accommodate any messages with larger prime numbers.

Digital Signatures:
RSA: Sign

- Now Bob wants to send a signed message to Alice
- **Bob's Public Key:** (2021,155)
- **Bob's Private Key:** 1583
- **M:** 411

Signature algorithm

$$S = 411^{1583} \bmod 2021$$
$$S = 402$$

Digital Signatures:
RSA: Sending M

- Now Bob wants to send a signed message to Alice
- **Bob's Public Key:** (2021,155)
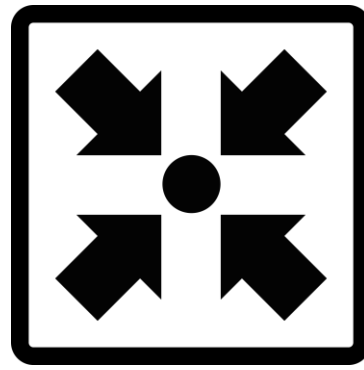- **Bob's Private Key:** 1583
- **M:** 411
- **S**=402

Bob

411
1583
Bob's private key

411
402

411
402
(2021,155)

Alice

Bob's public key

Digital Signatures:
RSA: Validating S

- Now Bob wants to send a signed message to Alice
- **Bob's Public Key:** (2021,155)
- **Bob's Private Key:** 1583
- **M:** 411
- **S**=402

Validating algorithm

$$M' = 402^{155} \bmod 2021$$
$$M'=402$$
If M'=M, return *true*
If M'!=M, return *false*

Meeting point slide

- Signatures provide message authenticity and integrity

- Only the owner of the private key can sign a message

- Anyone with his/her public key can verify the message

- The signature depends on the data, therefore a signature cannot be used to other data/messages
  - It is not the same as a human handwritten signture

- The sender needs to send the signature and the message

- The receiver needs to use the public key, the signature and the message to validate the integrity and authenticity

Signtures guarantee integrity and authentication.
Can we guarantee integrity with a simple solution?

Hash Function

- Hash functions also provide integrity
- They are cheaper – but don't provide authenticity
- A Hash function receives as an input an arbitrary size and outputs a fixed sized string
- The most important feature of these functions is that they are deterministic
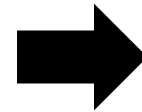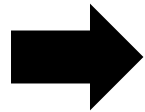- In other words, for the same input they produce always the same ouput
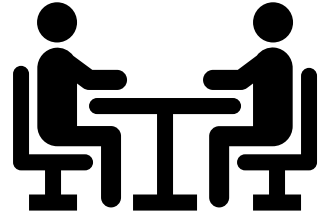
Hash Function

input

output

Hello world
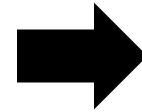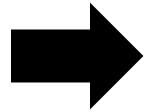
OD%O$ASL&/DFOR4562ODL"%

Hello **W**orld

OP#$kfjr4%&DASCdlf kgg4$(/F

Discussion slide

Can you think of a hash function solution?

Cryptographic Hash Function

- It is a one-way function; it is infeasible to find the inverse function of the hash function

- If you change one bit in the input the output should change drastically (this is called the avalanche effect)

- A few important properties that these functions guarantee:
  - Given M it should be easy to compute H(M) = *hash*
  - Given the *hash* it is infeasible to find M
  - It is hard to find M' and M such that H(M') = H(M)
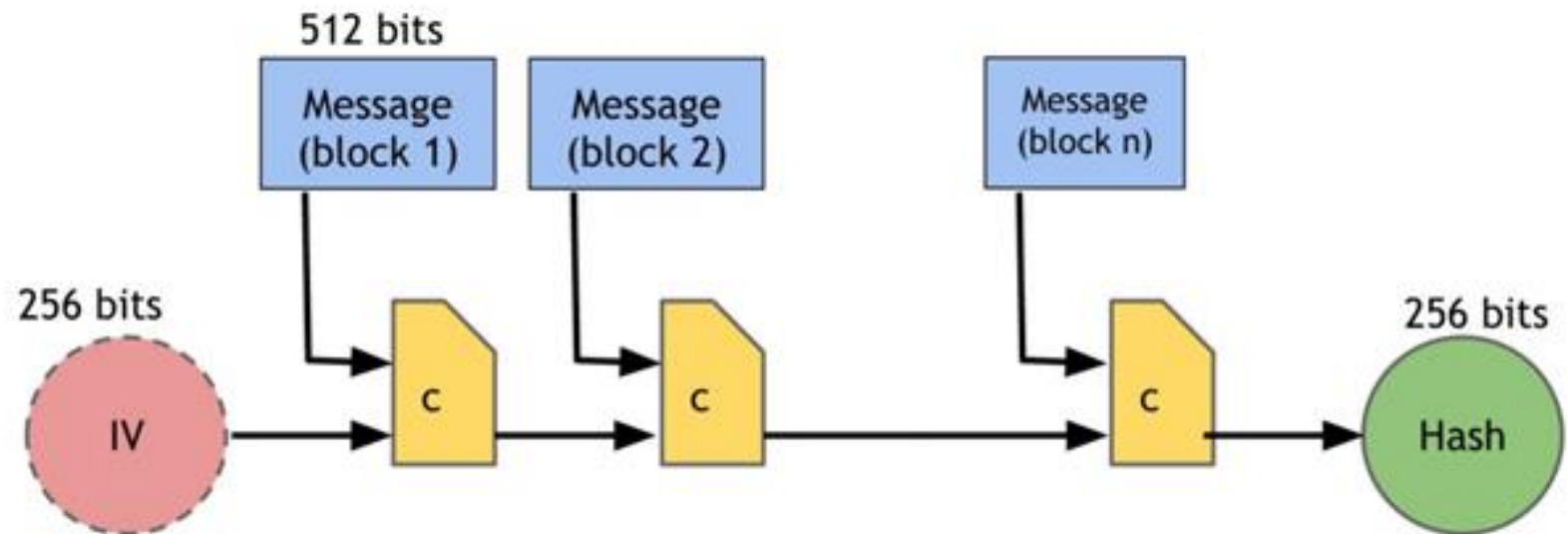
Cryptographic Hash Function: SHA256

- The SHA (Secure Hash Algorithm) is cryptographic hash function.

- A cryptographic hash is like a signature for a data set, or like the fingerprints of the data.

- SHA256 algorithm generates an almost-unique, **fixed size** 256-bit (32-byte) hash.

- It is suitable for checking integrity of your data, challenge hash authentication, anti-tamper, digital signatures, blockchain.

Cryptographic Hash Function: SHA256

- The SHA (Secure Hash Algorithm) is cryptographic hash function.

- A cryptographic hash is like a signature for a data set, or like the fingerprints of the data.

- SHA256 algorithm generates an almost-unique, **fixed size** 256-bit (32-byte) hash.

- It is suitable for checking integrity of your data, challenge hash authentication, anti-tamper, digital signatures, blockchain.

MD5 is no longer secure!

Cryptographic Hash Function: SHA256

- Input: any size
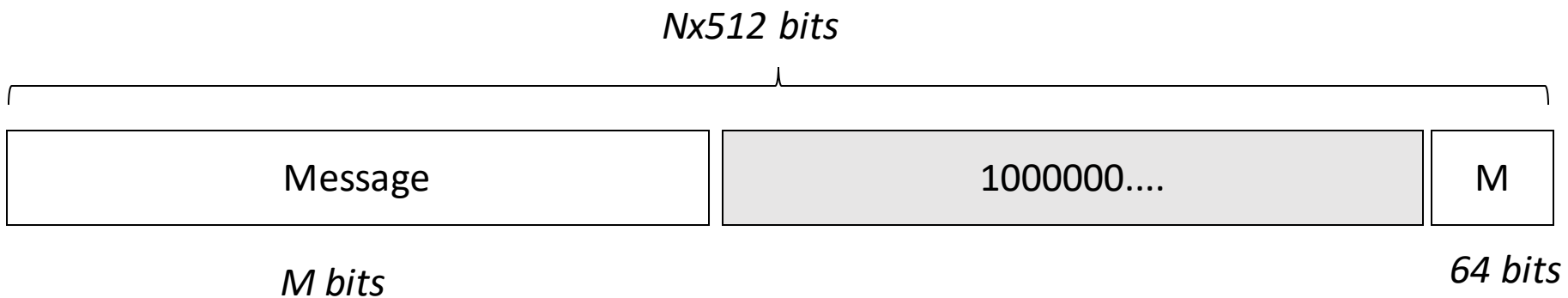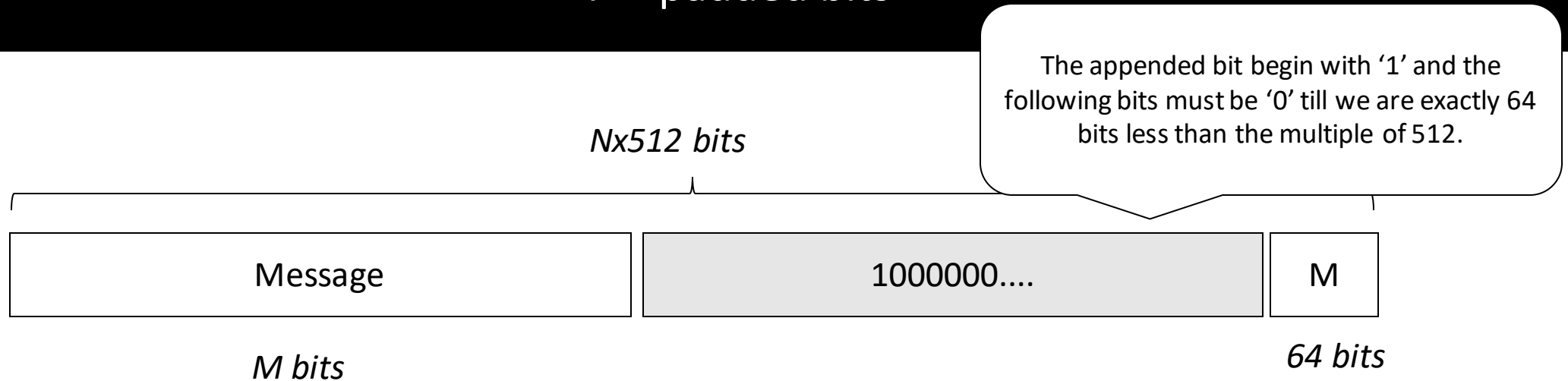- Block: 512 bits
- Output: 256 bits
- Rounds: 64

Cryptographic Hash Function: SHA256

- **Step1** add the padding to the input message

$$M + P + 64 = n \times 512$$
$$M = \text{length of original message}$$
$$P = \text{padded bits}$$

Nx512 bits

Message	1000000....	M

M bits

64 bits

Cryptographic Hash Function: SHA256

- **Step1** add the padding to the input message

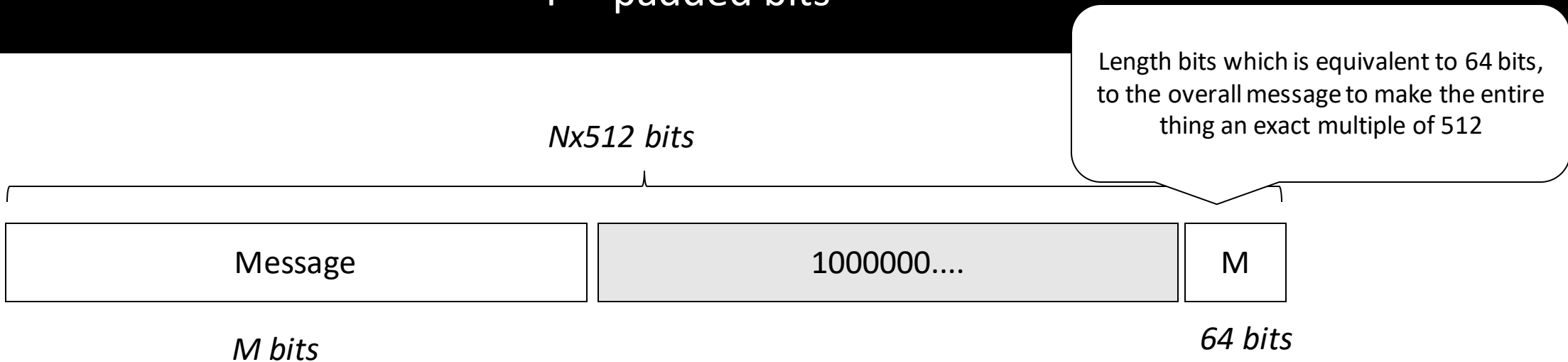M + P + 64 = n x 512
M = length of original message
P = padded bits

The appended bit begin with '1' and the following bits must be '0' till we are exactly 64 bits less than the multiple of 512.

Nx512 bits

| Message | 1000000.... | M |

M bits

64 bits

Cryptographic Hash Function: SHA256

- **Step1** add the padding to the input message

M + P + 64 = n x 512
M = length of original message
P = padded bits

Length bits which is equivalent to 64 bits, to the overall message to make the entire thing an exact multiple of 512

Nx512 bits

Message	1000000....	M

M bits

64 bits

Cryptographic Hash Function: SHA256

- **Step2** Initialize the buffers

a = 0x6a09e667
b = 0xbb67ae85
c = 0x3c6ef372
d = 0xa54ff53a
e = 0x510e527f
f = 0x9b05688c
g = 0x1f83d9ab
h = 0x5be0cd19

k[0..63] :=
  0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
  0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
  0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
  0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
  0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
  0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
  0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
  0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2

Cryptographic Hash Function: SHA256

- **Step3** Compression function

The entire message (n x 512 bits long) is divided into *n* chunks of 512 bits.
Each of these 512 bits, are then put through 64 rounds of operations.
The output is the input for the chunk

Cryptographic Hash Function: SHA256

- **Step3** Com

T   nks of 512 bits.
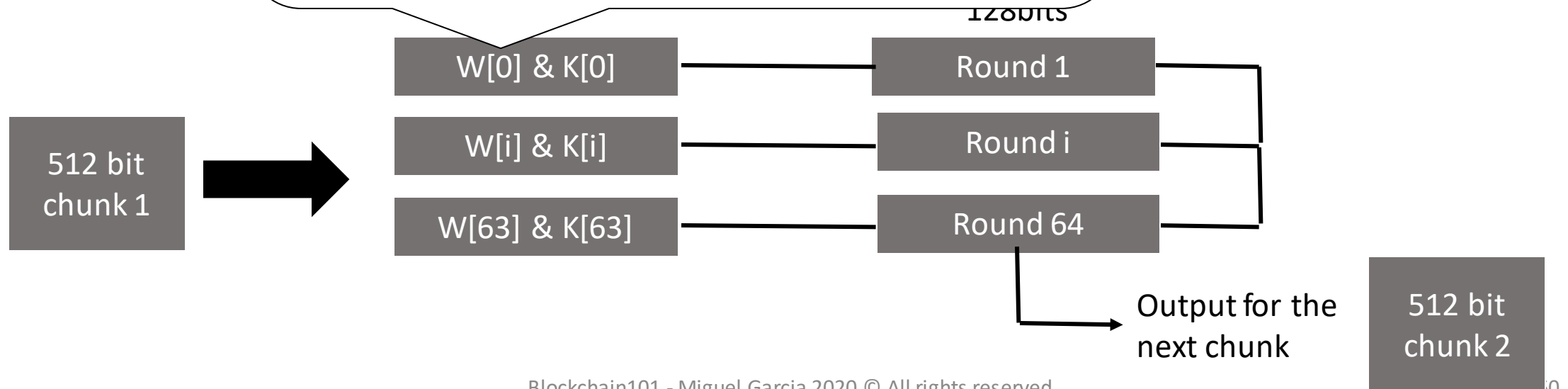   of operations.
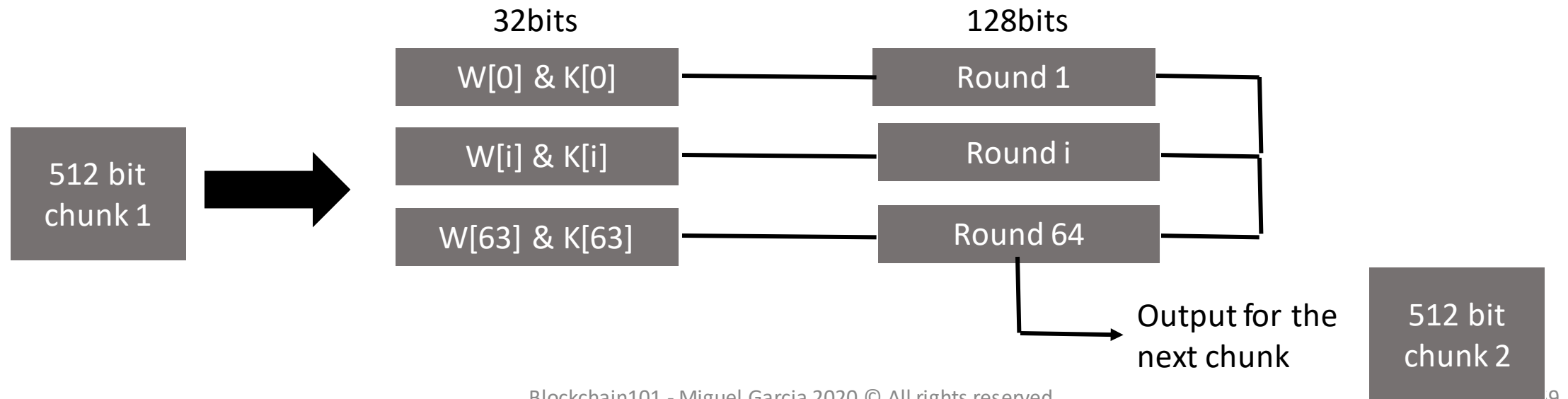
$W(i) = W^{i-16} + \sigma^0 + W^{i-7} + \sigma^1$

where:

$\sigma^0 = (W^{i-15} ROTR^7(x))$ XOR $(W^{i-15} ROTR^{18}(x))$ XOR $(W^{i-15} SHR^3(x))$

$\sigma^1 = (W^{i-2} ROTR^{17}(x))$ XOR $(W^{i-2} ROTR^{19}(x))$ XOR $(W^{i-2} SHR^{10}(x))$

ROTR^n(x) = Circular right rotation of 'x' by 'n' bits

SHR^n(x) = Circular right shift of 'x' by 'n' bits

128bits

W[0] & K[0]	Round 1
W[i] & K[i]	Round i
W[63] & K[63]	Round 64

512 bit chunk 1

Output for the next chunk

512 bit chunk 2

Cryptographic Hash Function: SHA256

- **Step3** Compression function

$Ch(E, F, G) = (E \text{ AND } F) \text{ XOR } ((\text{NOT } E) \text{ AND } G)$

$Ma(A, B, C) = (A \text{ AND } B) \text{ XOR } (A \text{ AND } C) \text{ XOR } (B \text{ AND } C)$

$\sum(A) = (A \ggg 2) \text{ XOR } (A \ggg 13) \text{ XOR } (A \ggg 22)$

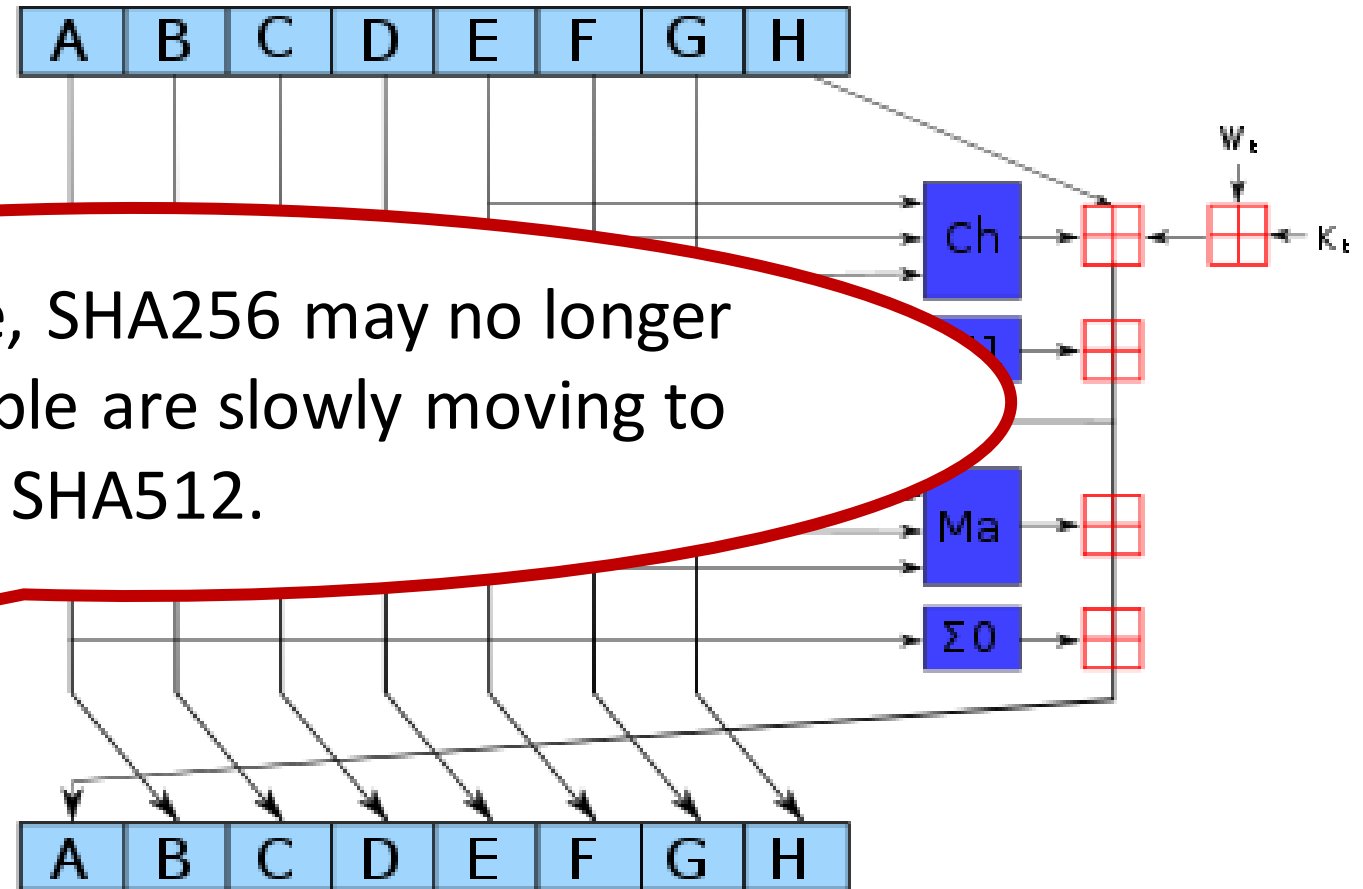$\sum(E) = (E \ggg 6) \text{ XOR } (E \ggg 11) \text{ XOR } (E \ggg 25)$

⊞ = addition modulo 2

Cryptographic Hash Function: SHA256

- **Step3** Compression function

In a near future, SHA256 may no longer be secure. People are slowly moving to SHA512.

Meeting point slide
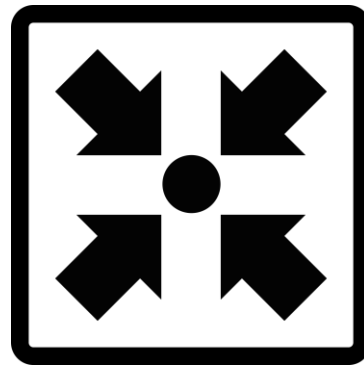
- Cryptographic hash functions are one-way functions
  - This means it (should be) is impossible to find the input based on the output
- They are also deterministic
  - This means that for the same input the output is always the same
- The hash function output has a fixed size output, the input can have any size
- It guarantees integrity, it can be used – with caution – to encrypt data
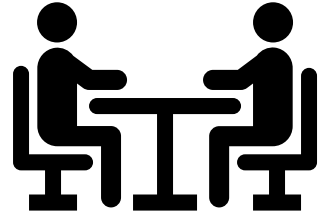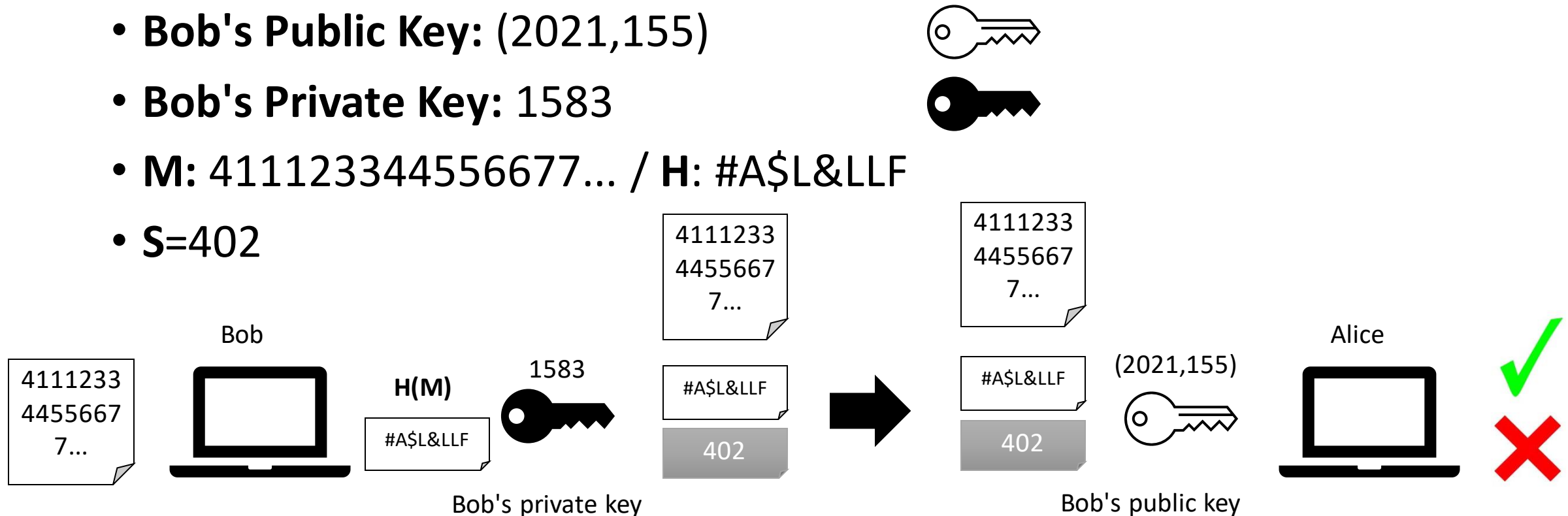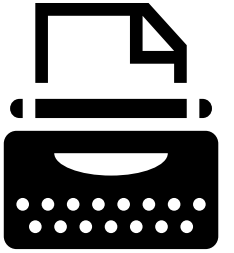
Discussion slide

Can you think of a use for hash functions in digital signatures?

Digital Signatures + Hash functions:

- Now Bob wants to send a signed message to Alice
- **Bob's Public Key:** (2021,155)
- **Bob's Private Key:** 1583
- **M:** 4111233445566 77... / **H**: #A$L&LLF
- **S**=402

Coding time

- Python overview
  - Basic structures
- Hash function example: SHA256
- Signature example: RSA