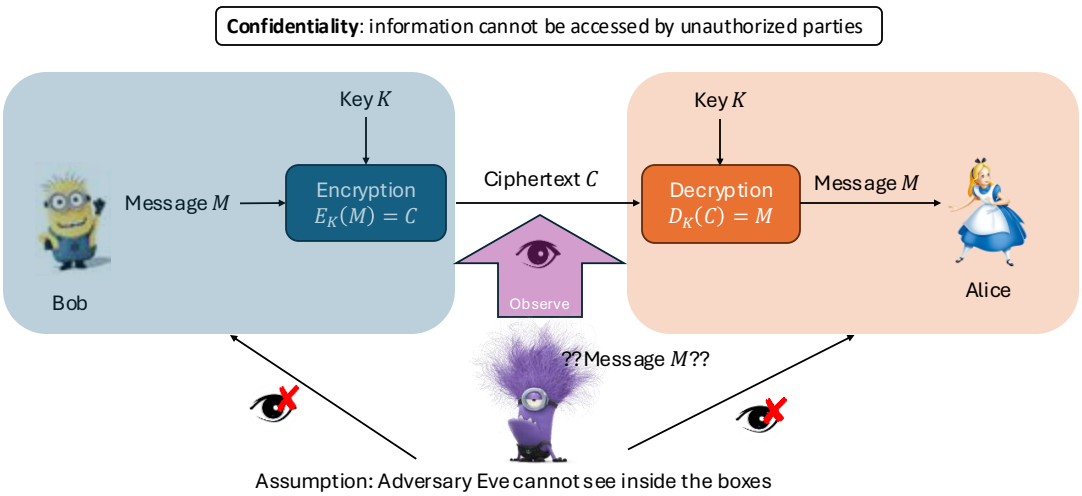
Computer Security and Privacy (COM-301)

Week 6: Applied Cryptography Continued

Theresa Stadler

Some slides/concepts adapted from: Carmela Troncoso, George Danezis

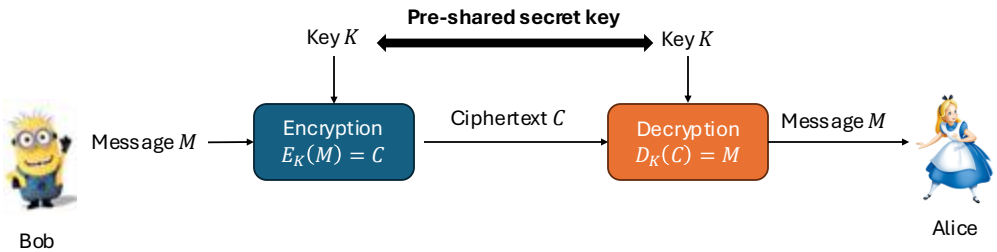
Recap: Last week



What we learned about last week:
The basic problem of cryptography
How to achieve message confidentiality against an adversary that does not have access to some secret key K
Secrecy relies on assumption that adversary cannot have access to the secret key

Recap: Last week

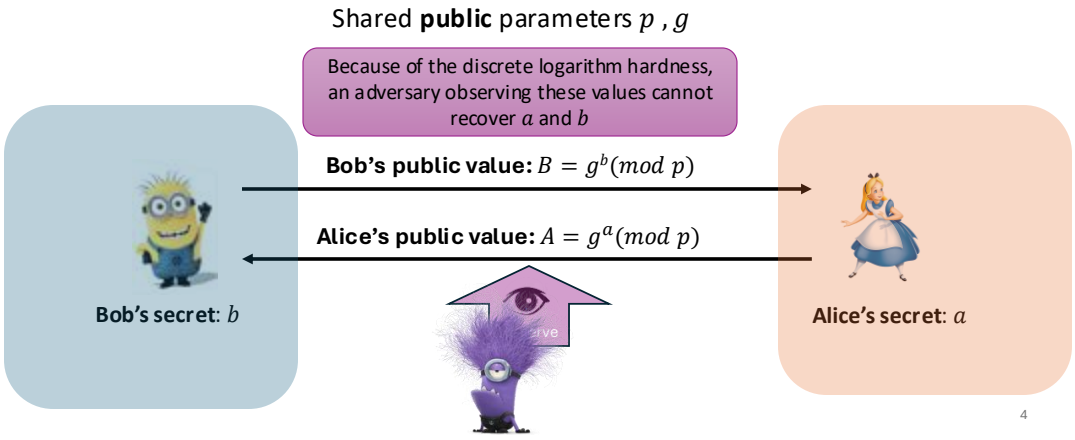
Symmetric encryption schemes: Encryption and decryption use the same key



Learned about OTP and stream ciphers which are a form of symmetric encryption where the encryption and decryption use same key

Recap: Last Week

Diffie-Hellman key exchange



Learned about DH key exchange which enables a sender and a receiver to obtain a shared key without an eavesdropper being able to compute this key.

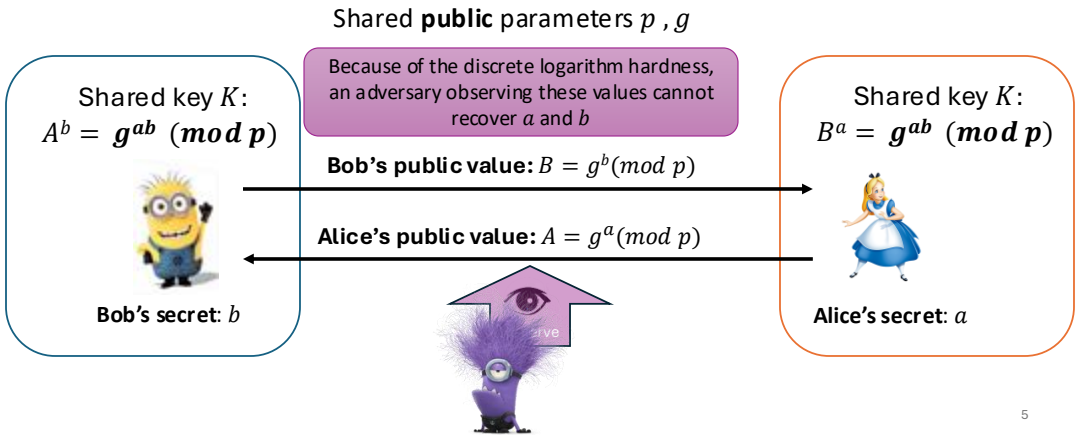
How it works is that Bob and Alice both choose a secret value at random.
Bob sends to Alice g^b *mod p*, where b is Bob's secret; and Gru sends to Bob g^a *mod p,* where a is Alice's secret key.

With these values, both Alice and Bob can compute the same secret key g^{xy}.

Yet, an adversary eavesdropping on the channel cannot compute the same key because they do not know x or y. And, due to the **hardness of the discrete logarithm problem** they cannot recover these values from what is observed on the wire (i.e., x cannot be recovered from $P_b = g^x$ *mod p).*

Recap: Last Week

Diffie-Hellman key exchange



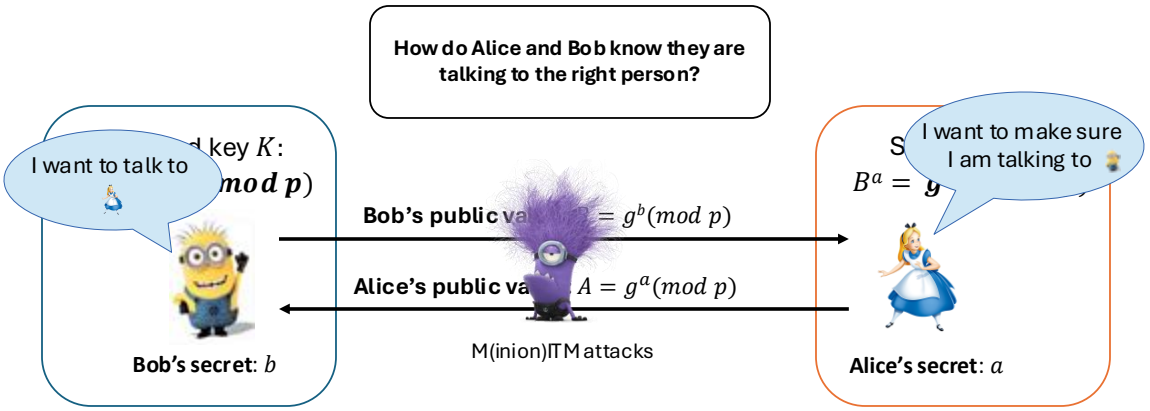
Learned about DH key exchange which enables a sender and a receiver to obtain a shared key without an eavesdropper being able to compute this key.

How it works is that Bob and Alice both choose a secret value at random.
Bob sends to Alice $g^b \bmod p$, where b is Bob's secret; and Gru sends to Bob $g^a \bmod p$, where a is Alice's secret key.

With these values, both Alice and Bob can compute the same secret key g^{ab}.

Yet, an adversary eavesdropping on the channel cannot compute the same key because they do not know a or b. And, due to the **hardness of the discrete logarithm problem** they cannot recover these values from what is observed on the wire (i.e., a cannot be recovered from $A=g^a \bmod p$).

Recap: Last week



DH solves Key Distribution, but creates the Identity Verification Problem.
Solution: DH must be paired with a mechanism to verify identity
Setup: Parties still need to trust a third party (a Certificate Authority) for public key ownership.

Problem I: Authenticity

Solve the problem we finished with last week: authenticity of sender of a message

How do we verify the identity of the person to whom we are talking?

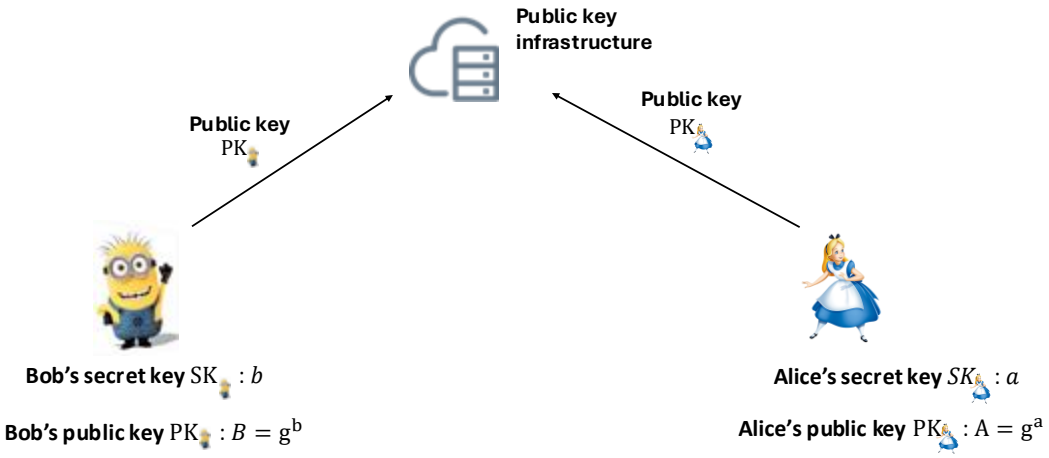
Bob's secret: b

...while avoiding Eve in the middle

Alice's secret: a

The problem with which we finished last week was how to verify that when we have an encrypted exchange with someone we can be sure we are talking to the right person

Public key cryptography

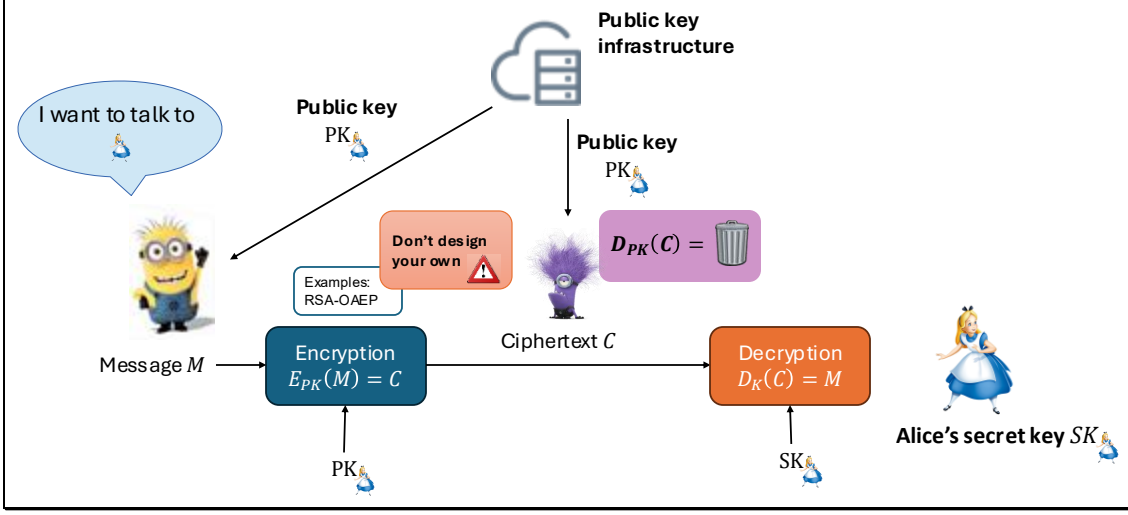


We are first gonna solve this problem in the DH setting that you finished with last week which we also call asyemtric or public key crypto.
In asymmetric cryptography every user has two keys:

One **secret key** that the user keeps to himself
One **public key** that *does not need to be secret*
(Omitting the mod p here for simplicity)

The public keys can be uploaded to public servers that form the backbone of so-called **Public Key Infrastructure.**

Confidentiality: Encryption & Decryption



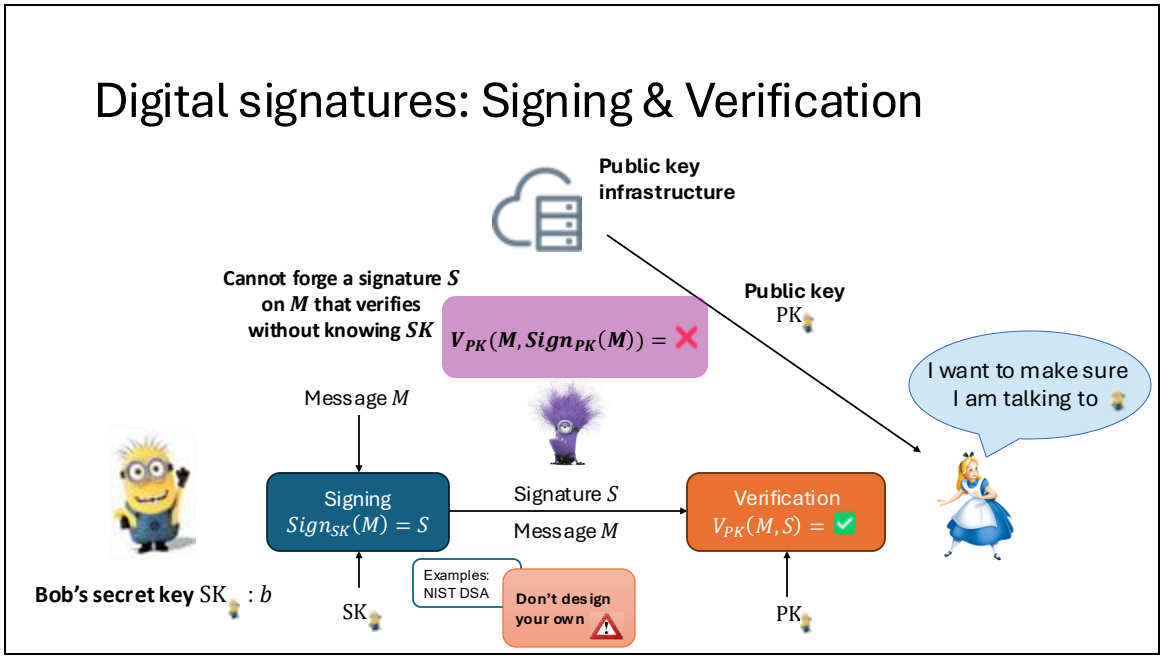
When Bob wants to write to Alice:

- Finds Alice's public key in the repository.
- Encrypts the message with Alice's public key
- Sends the resulting ciphertext to Alice

- This message **can only** be decrypted with Alice's secret key

- Decrypting the message with the public key returns garbage. This is what provides us with confidentiality

- And as a reminder to not roll your own crypto: If you implement use standardised ecnryption schemes for these boxes that provide us with en and decryption



Going to follow the same schema to solve the identity verification problem

Bob has a secret key. Using this key Bob performs a **Digital signature** on the message.

To verify a signature, Alice gets Bob's public key from the repository and performs a Verification function using the message, the signature, and *Bob's public key***.** This function returns yes if the signature valid, and no otherwise.

To be secure, and provide integrity, Digital Signature algorithms need to be such that it is not possible to create a Signature for a message that will verify under a given public key, unless you have the corresponding secret key.

Under these properties, a digital signature enables Alice to know that she is talking to Bob: only Bob, that knows the secret key, could have created the signature

Digital signatures

Computationally costly compared with most symmetric key algorithms of equivalent security

Signing and encrypting **is slow**

→ Sign **hash** of messages

Public key infrastructure

Public key
PK

Signing
$Sign_{SK}(M) = S$

Signature S

Verification
$V_{PK}(M, S) =$ ✅

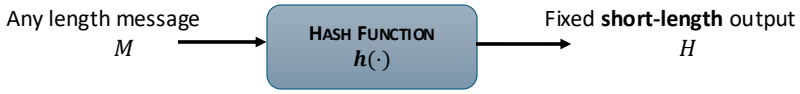
SK

PK

The commodity of not having to pre-share a key comes at a cost. Asymmetric cryptography is much more expensive than symmetric counterparts with the same security level (both for encryption and signing).

If we would do it naively like this we would have a problem: If we sign long messages, both signing and verification will take a very long time and will be not practical
We can solve this problem with has functions

Hash functions



THREE SECURITY PROPERTIES

PRE-IMAGE RESISTANCE

Given $\mathbf{H} = \mathbf{h}(\boldsymbol{M})$, difficult **to find** $\boldsymbol{M}$

SECOND PRE-IMAGE RESISTANCE

Given $\boldsymbol{M}$, difficult **to find an** $\boldsymbol{M'} != M$ such that $\mathrm{h}(M') = h(M)$

COLLISION RESISTANCE

Difficult to find any M, M' such that $\mathrm{h}(M) = h(M')$

A cryptographic hash function is an **unkeyed** cryptographic primitive. It takes as input a message, and outputs a short fixed-length string of bits.

The correspondence between input and output is **deterministic.** Given a message m, a hash function H will always output the same string h=H(m)

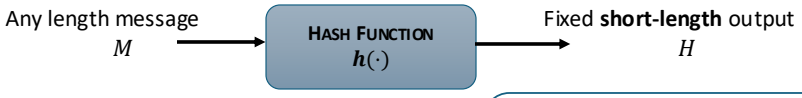
Cryptographic hash functions are designed to be fast and efficient to compute, and they have several important properties that make them valuable in various applications:

Pre-image resistance: given a hash of a message H(m) it is hard to recover the original message m. This property says that hash functions are not invertible.

Second pre-image resistance: given a message m and its hash H(m), it is hard to find another message m' (different from the original message) with the same hash (H(m') = H(m)).

Collision resistance: it is hard to find two messages m, m' (m != m') thet have the same hash (H(m) = H(m'))

Hash functions



MD5 (1991): 128 bit hash – insecure
SHA0, SHA1: 160 bits – insecure
SHA-2 (224/256 /384/512) – OK but slow
SHA-3 (224/256 /384/512)

Don't design your own

THREE SECURITY PROPERTIES

PRE-IMAGE RESISTANCE

Given $\mathbf{H} = \mathbf{h}(\boldsymbol{M})$, difficult **to find** $\boldsymbol{M}$

SECOND PRE-IMAGE RESISTANCE

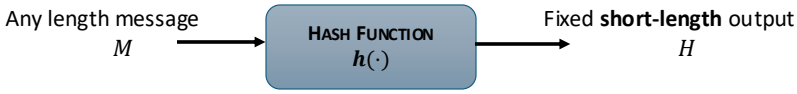
Given $\boldsymbol{M}$, difficult **to find an** $\boldsymbol{M'} ! = M$ such that $\mathrm{h}(M') = h(M)$

COLLISION RESISTANCE

Difficult to find any M, M' such that $\mathrm{h}(M) = h(M')$

As with any other primitive, designing good hash functions that fulfil these properties is hard. There are many standardized functions, use those.

Hash functions



THREE SECURITY PROPERTIES

PRE-IMAGE RESISTANCE

Given $\mathbf{H} = \mathbf{h}(\boldsymbol{M})$, difficult **to find** $\boldsymbol{M}$

SECOND PRE-IMAGE RESISTANCE

Given $\boldsymbol{M}$, difficult **to find an** $\boldsymbol{M'} != M$ such that $\mathrm{h}(M') = h(M)$

COLLISION RESISTANCE

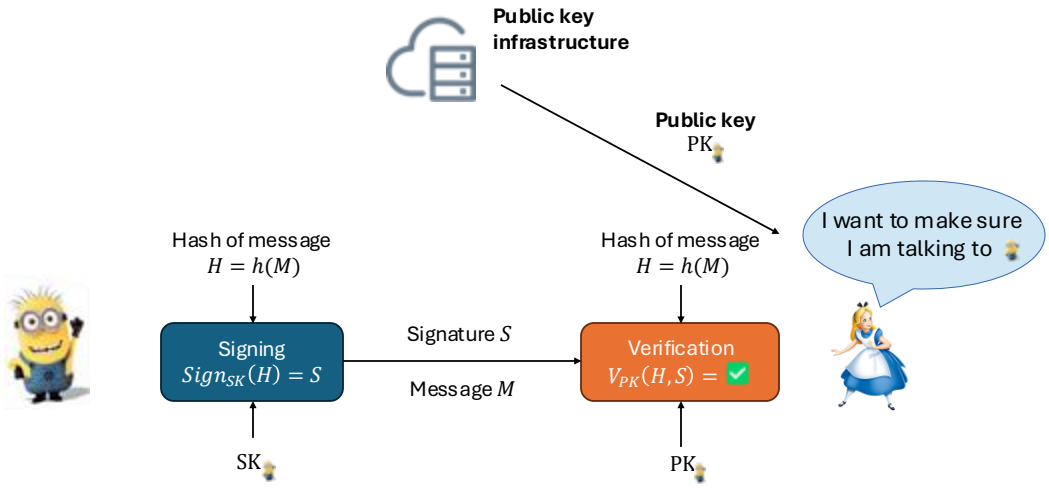
Difficult to find any M, M' such that $\mathrm{h}(M) = h(M')$

USES

Support digital signatures, build HMAC, password storage, file integrity, secure commitments, secure logging, blockchains, …

Cryptographic hash functions are designed to be fast and efficient to compute, and they have several important properties that make them valuable in various applications:

Digital signatures on hash functions



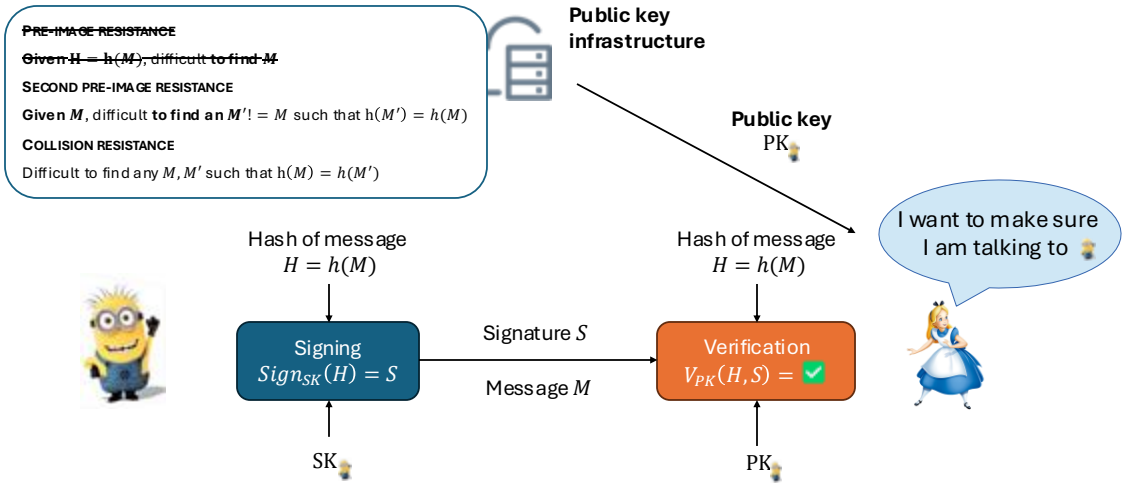
To sign a message efficiently, Bob:

1) Hashes the message
2) Signs the message hash
3) Sends to Alice the message, and the signature over the hash.

To verify the integrity and provenance of the message, Alice:

1) Computes the hash of the message (since hashing is deterministic, Alice obtains the same hash as Bob in Bob's step 1)
2) Verifies that the signature on the hash she just computed is valid

Digital signatures on hash functions



For digital signatures on hashes to be equivalent to sign the message, the hash function must have the following properties:

Preimage resistance is not needed, as the message m is already sent in public

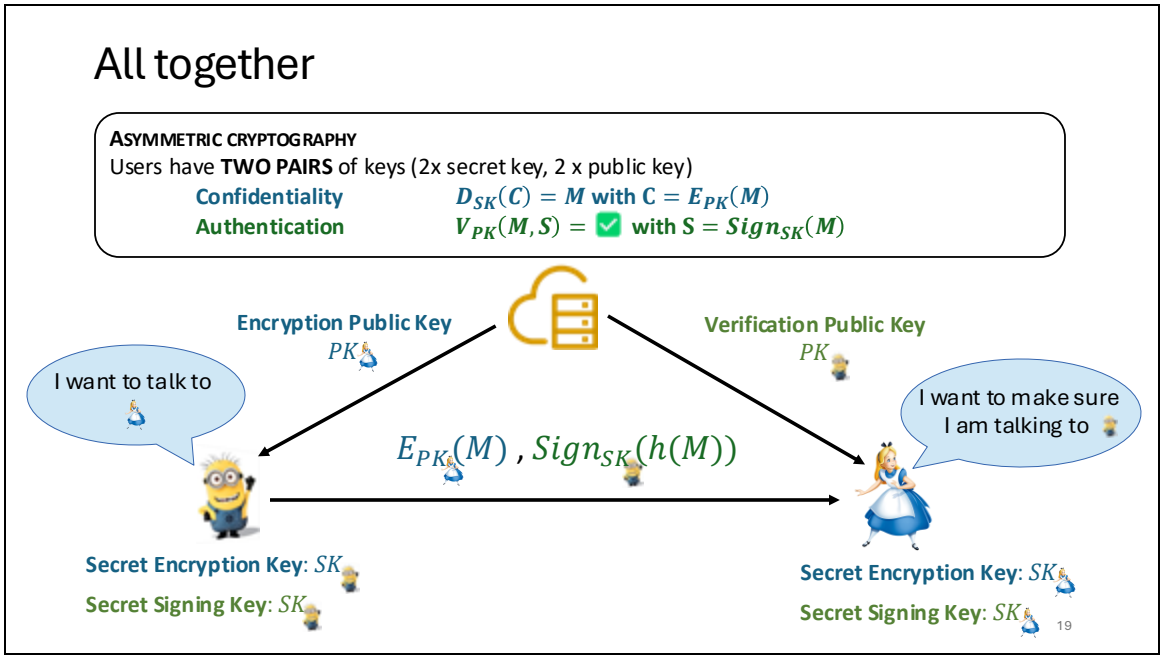
Second pre-image resistance says that, given a signature on h(m), the hash of message m; it is hard to find a second message that leads to the same hash and therefore to the same signature. In other words, given a signature on a hash, you cannot have an alternative message to the true one.

Collision resistance says that you cannot produce two messages that lead to the same hash. That means that you cannot have someone sign a message m, and then claim that they signed another message m'

Ok so this solves the problem that Alice wants to make sure she is talking to Bob, but what problem does it create if the actual exchange would look like this?

And all together now…

Message confidentiality an authenticity for asymmetric cryptography

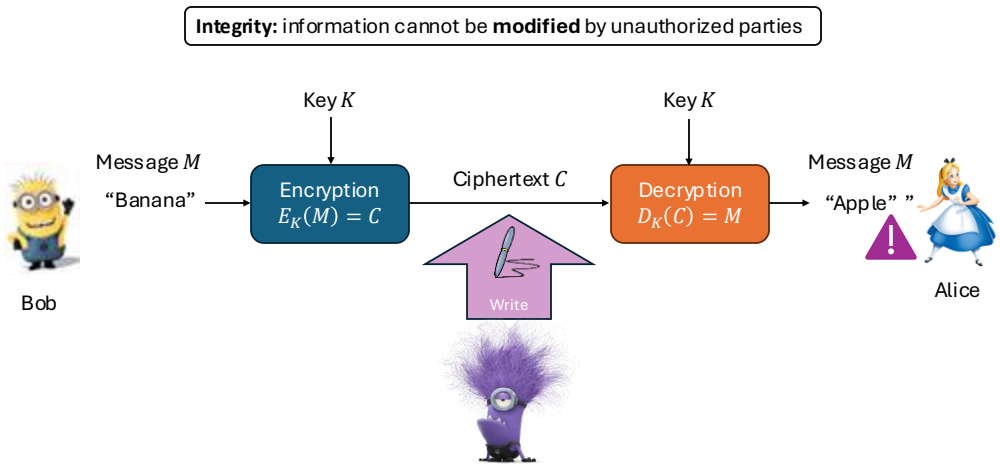


Putting it all together: We now have message confidentiality and sender authenticity

Motivated this through identity verification problem but there is actually another problem in crytogrpahic echanges that we just solved through digital signatures which is

Problem II: Integrity

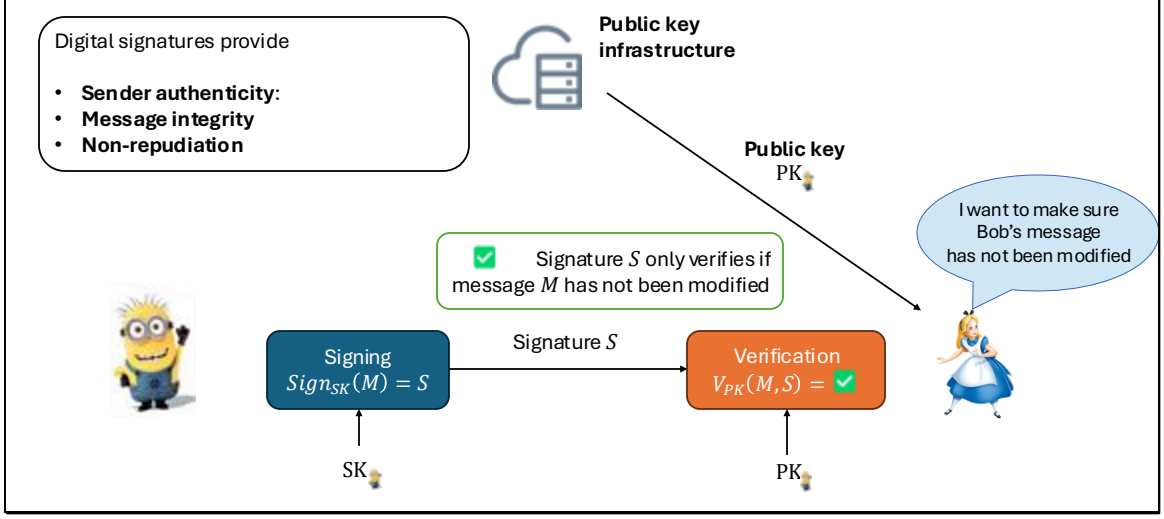
Integrity



Integrity means that non-authorized parties cannot modify data. That is if Bob sends "Banana" he can be sure that Alice will receive "Banana"

A cryptographic algorithm that provides integrioty should protect against Eve the adversary sitting in the middle being able to tamper with the ciphertext such that Alice when she dcrypts reads Apples instead of Bananas

Message Integrity through Digital Signatures



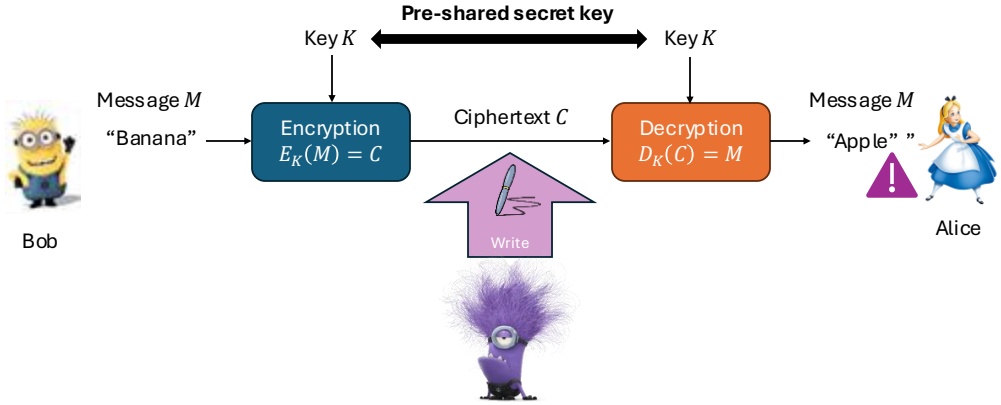
In asymmetric crypto we have already seen a way to solve the integrity problem: Digital signatures

Digital signatures provide:

- **Authenticity of the sender**: since no party than the sender can create a valid signature for a message, the server can verify that the identity of the sender. No adversary can produce a valid signature to the public key of the sender and therefore cannot impersonate the sender.
- **Integrity of the message:** since no party than the sender can create a valid signature for a message, no adversary can modify the message without being detected (if they modify the message, the signature

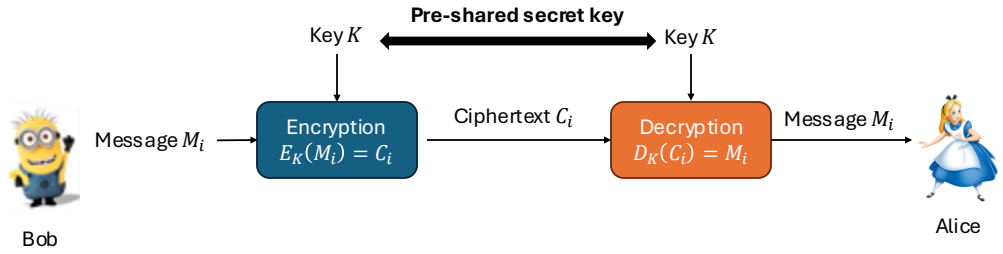
•would not be valid)
•**Non-repudiation:** since no party than the sender can create a valid signature for a message, if a message has a valid signature, the sender cannot claim they did not sign it (no-one else can produce such signature).

And for symmetric cryptography?



Rest of lecture: How can we provide message integrity for symmetric exchanges

Block cipher



The algorithms work on blocks the size of the key

→Typically 128/256 bits

Problem: Messages are longer than a block!

→Requires iteration → Block ciphers have a "**Mode of Operation**"

The building block that we need for that are block ciphers

An encryption algorithm of a block cipher **works on small blocks**.

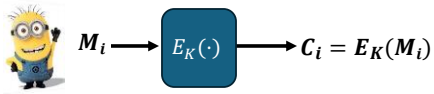
Encryption receives the key and a message block and outputs an encrypted block of the same size as the input.
The encryption algorithm is such that given the output it looks random. In other words, the output is independent from the input.

To **decrypt**, the receiver uses an algorithm Decrypt, that is (generally) not the same as encryption. Algorithm. Given an encrypted block and a key, the decryption algorithm outputs the plaintext.

Messages are more than one block, so we need to have a way of chaining blocks together. This "chaining" is called a **mode of operation**.

Mode 1: ELECTRONIC CODE BOOK (ECB)

Straightforward scheme: encrypt & decrypt single blocks



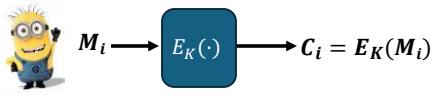
$M = M_1 M_2 M_3 M_4$

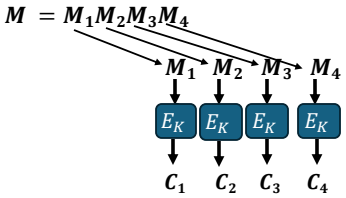
Each block M_i is the same size as the key

Naively, if you would have to come up with a way to solve this problem you would probably use block ciphers ECB mode.
We will always denote message blocks which are size of key like this

Mode 1: ELECTRONIC CODE BOOK (ECB)

Straightforward scheme: encrypt & decrypt single blocks



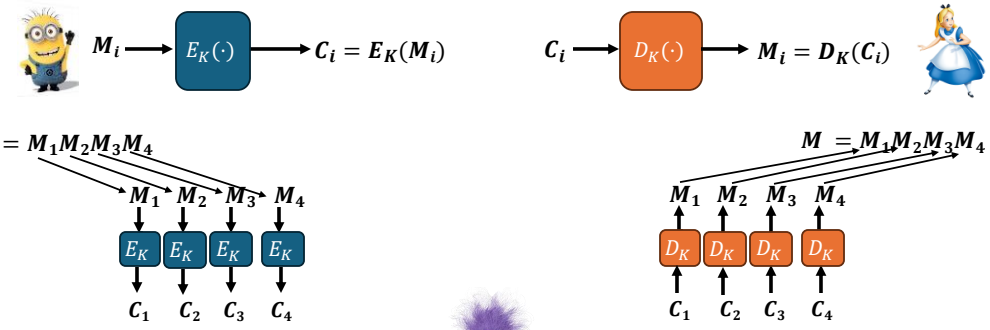


Naively, if you would have to come up with a way to solve this problem you would probably use block ciphers ECB mode.

Electronic Code Book (ECB) is a mode of operation in which blocks are encrypted *independently.*

Mode 1: ELECTRONIC CODE BOOK (ECB)

Straightforward scheme: encrypt & decrypt single blocks



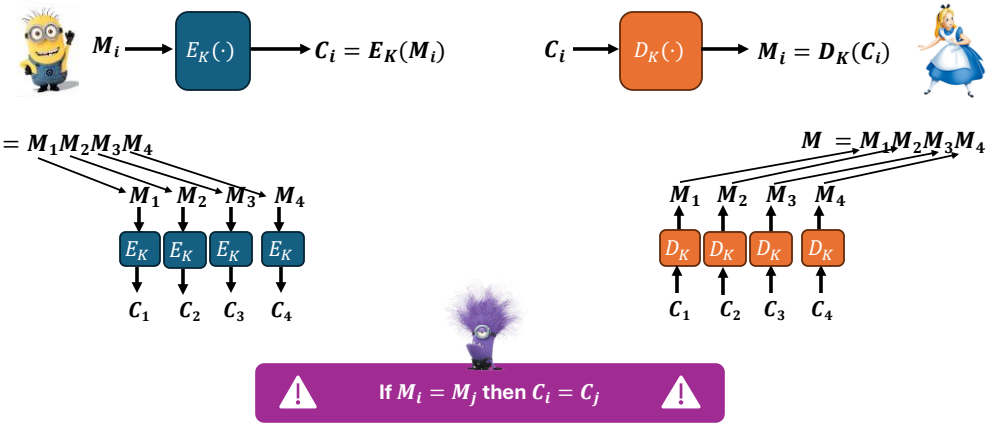
Can someone spot a problem?

To decrypt, one also uses the decryption algorithm on individual ciphertext blocks.

What is the problem with this naïve approach to encrypting blocks?

Mode 1: ELECTRONIC CODE BOOK (ECB)

Straightforward scheme: encrypt & decrypt single blocks



Thus, if two blocks in the message are the same, they will have the same appearance when encrypted!

Mode 2: CIPHER BLOCK CHAINING (CBC)

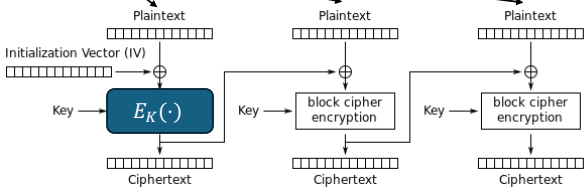
Add IV and propagate information across blocks to introduce randomness

$M_i \rightarrow E_K(\cdot) \rightarrow C = E_K(M_i)$

CBC Encryption $E_K(\cdot)$

$$C_0 = IV$$
$$C_i = E_K(M_i \oplus C_{i-1})$$

$M = M_1 M_2 M_3$



https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#/media/File:CBC_encryption.svg

Cipher Block Chaining is a mode of operation in which a block encryption depends on previous blocks (as defined in the formula)

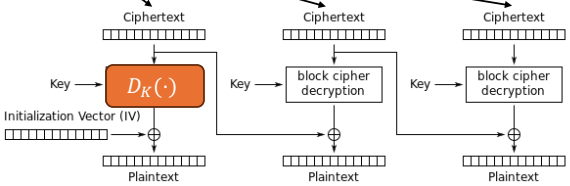
Note that every time a block is encrypted, by the properties of the block cipher, the ciphertext is random. Because the IV and/or the plaintext changes every time, this random value is different every time. Thus, when XORed with the next plaintext it creates a random string. If the IV changes, this is similar to a one-block one-time pad, but if one reuses the IV for the same plaintext and key, this value will be repeated.

Mode 2: CIPHER BLOCK CHAINING (CBC)

Add IV and propagate information across blocks to introduce randomness

$C_i \rightarrow D_K(\cdot) \rightarrow M_i = D_K(C_i)$

$C = C_1 C_2 C_3$



CBC Decryption $D_K(\cdot)$

$$C_0 = IV$$

$$M_i = D_K(C_i) \oplus C_{i-1}$$

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#/media/File:CBC_encryption.svg

To decrypt **Cipher Block Chaining** one has to do the inverse operation. Notice where the XOR with the IV and previous ciphertext happen as opposed to the previous slide.

If IV is incorrect, only the first block is corrupted!! Note that the rest only depend on the ciphertext.

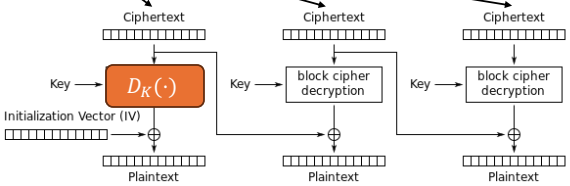
Blocks cannot be decrypted without having the previous ciphertext! One needs to XOR it with the output of the decryption.

Mode 2: CIPHER BLOCK CHAINING (CBC)

Add IV and propagate information across blocks to introduce randomness

$C_i \rightarrow D_K(\cdot) \rightarrow M_i = D_K(C_i)$

$C = C_1 C_2 C_3$



https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#/media/File:CBC_encryption.svg

CBC Decryption $D_K(\cdot)$

$C_0 = IV$

$M_i = D_K(C_i) \oplus C_{i-1}$

What if IV is incorrect? Is the full decryption wrong?

Can you decrypt a block alone? What do you need?

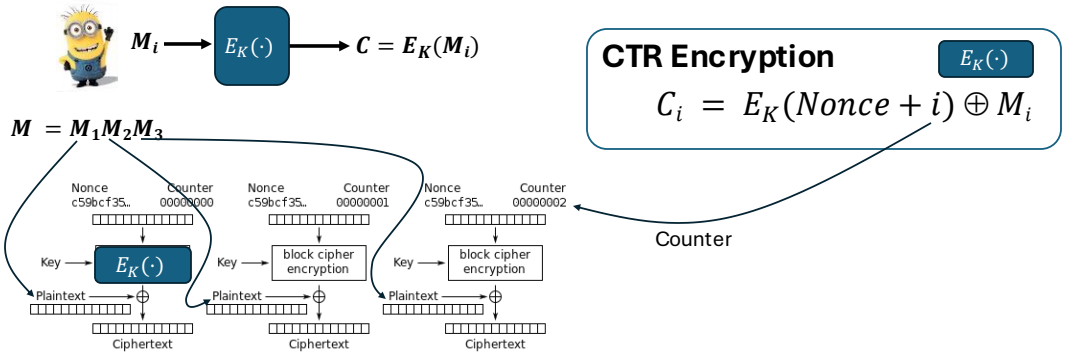
To decrypt **Cipher Block Chaining** one has to do the inverse operation. Notice where the XOR with the IV and previous ciphertext happen as opposed to the previous slide.

Q1: If IV is incorrect, only the first block is corrupted!! Note that the rest only depend on the ciphertext.

Q2: Blocks cannot be decrypted without having the previous ciphertext! One needs to XOR it with the output of the decryption.

Mode 3: COUNTER MODE (CTR)

Use increasing nonce to add randomness without dependencies between blocks



In Counter mode, we use a unique number composed by a **Nonce** and a **counter** so that the encryption receives a new number every time.
When encrypted, this number becomes a random one-time-pad that we XOR with the plaintext to obtain the ciphertext.

The counter keeps increasing so that, if the nonce is new, the output of the block cipher for every block will be different.

Mode 3: COUNTER MODE (CTR)

Use increasing nonce to add randomness without dependencies between blocks

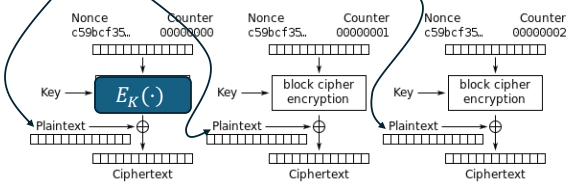
$M_i \rightarrow E_K(\cdot) \rightarrow C = E_K(M_i)$

Nonce: Random number used only **once**

CTR Encryption $E_K(\cdot)$

$$C_i = E_K(\boldsymbol{Nonce} + i) \oplus M_i$$

$M = M_1 M_2 M_3$



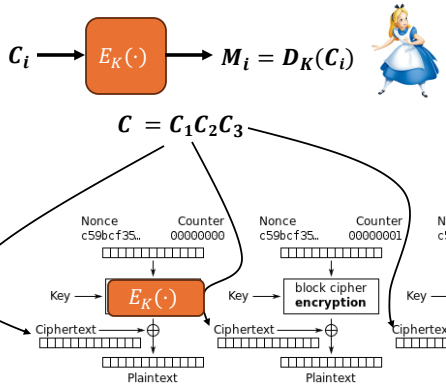
Counter

Do we need a decryption algorithm?

Nonce = number used only once. It **must** be new fro every message. Otherwise we are feeding the encryption with the same number and we obtain the same random number at the output: *effectively we would be reusing a one time pad!*

Mode 3: COUNTER MODE (CTR)

Use increasing nonce to add randomness without dependencies between blocks



CTR Decryption $E_K(\cdot)$

$$M_i = E_K(Nonce + i) \oplus C_i$$

For counter mode we **do not need the Decryption algorithm.**

As CTR mode operates like a one time pad, to decrypt we need the same random sequence of bits, i.e., we need the same sequence which is obtained using the same algorithm: **encryption.**

Block ciphers

STRENGTHS

High diffusion: information from one plaintext symbol is diffused into several ciphertext symbols

Immunity to tampering: difficult to insert symbols without detection

WEAKNESSES

Slow: an entire block must be accumulated before encryption / decryption can begin

Error propagation: in some modes of operation errors affect several bits/blocks

*Individual modes may not fall in these categories and may offer a different trade-off

Strengths:

Because of the chaining and the entangling of blocks there is great diffusion: every bit affects several blocks

The chaining also help detecting insertion or modification: if something changes it will be propagated and decryption will not make sense.

Weaknesses:
Block ciphers are slow as compared with a stream cipher because they need to wait for a full block before encryption/decryption.

When there is an error, in some modes such as CBC where encryption is chained, the error gets propagated to other blocks

Modes of operation: summary

Electronic Code Book (**ECB**)

- Directly encrypt and decrypt single blocks
- Large information leakage due to lack of randomness across blocks' encryption

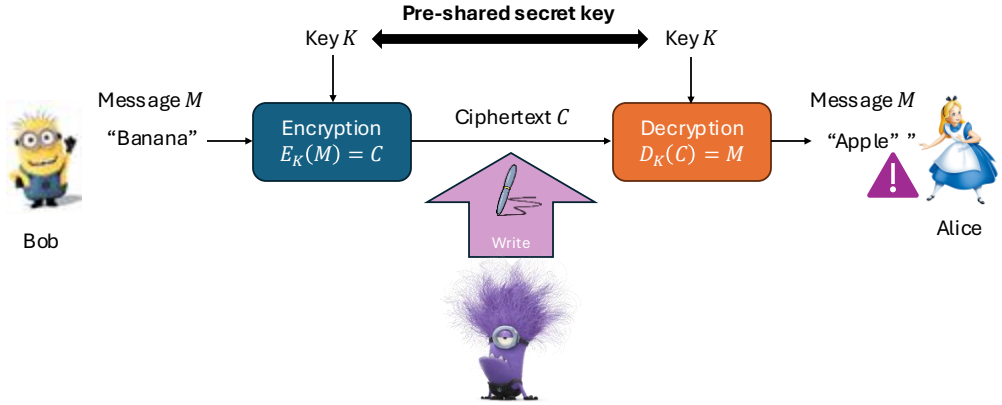
Cipher Block Chaining (**CBC**)

- Avoids ECB problems by using previous blocks to add randomness to every encryption
- Propagates errors and prevents single-block decryption

Counter mode (**CTR)**

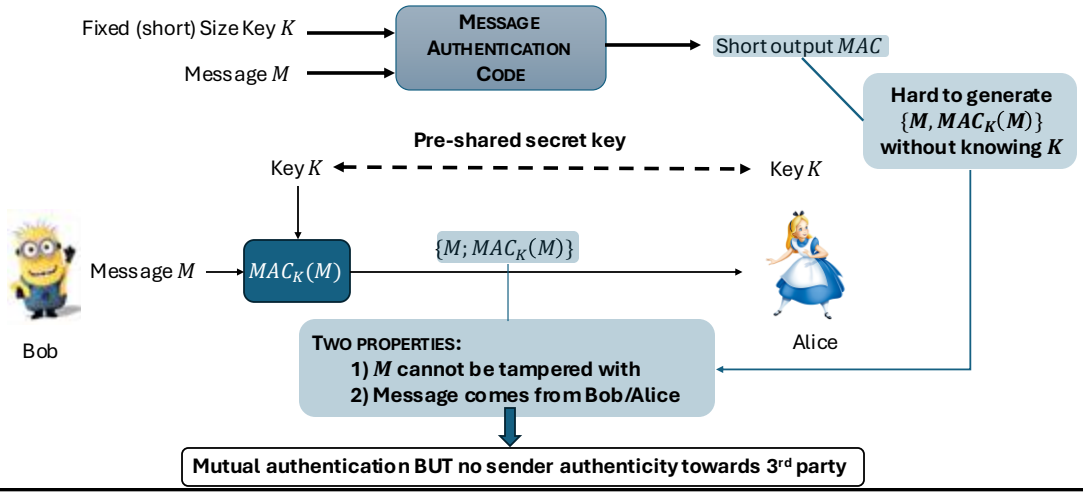
- Uses an increasing "nonce" to introduce randomness across block's encryptions
- Enables parallel encryption and decryption

So what about integrity?



Let's come back to our original problem which is how to get message integrity

Integrity from symmetric encryption: Message Authentication Codes (MAC)



A **Message Authentication Code** receives two inputs:

- A small key that must be kept secret
- The message whose integrity needs to be secured

The MAC algorithm outputs a short string that helps Alice to verify the integrity of the message sent by Bob.

The key property of the MAC is that *it is very difficult to produce a pair [message; MAC(k,message)] without knowing the key*

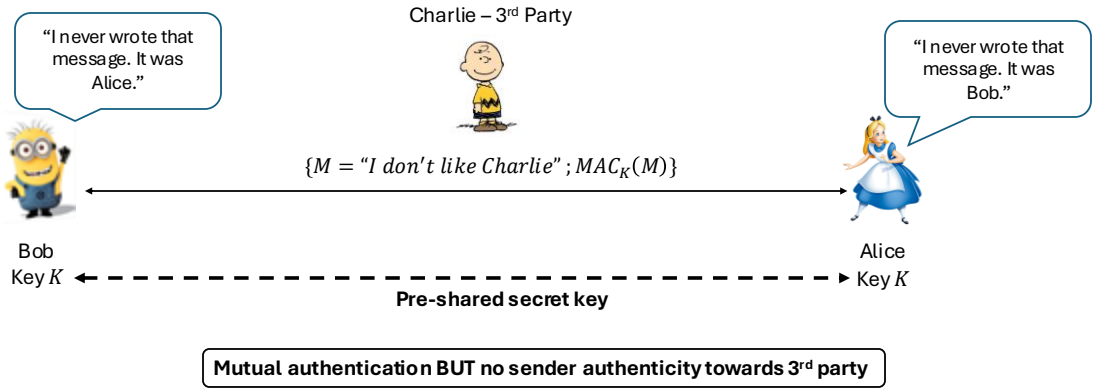
Bob and Alice have a per-share secret key. Bob inputs message and key into MAC and send MAC and message to Alice

To verify the integrity of the message, Alice inputs the message and the key on the MAC algorithm, and compares the output to the MAC she received. If they are the same, the message has not been tampered with.

Because only Bob and Alice know the key, when receiving a message with a valid MAC they know that only the other could have written it and the message has not been modified.

However, they cannot prove to a third party that they are not the author of the message

Repudiation



What does this mean?
If we imagine an exchange like this which CHarlie our third party observes.

Even though Alice and Bob know they are talking to each other (only any party with knowledge of K could have produced a valid MAC, message pair, they both could have produced the message.

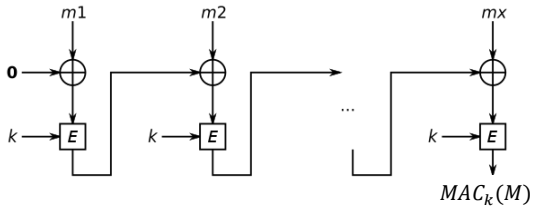
So the message can be repudiated which means both A and B can claim it was the other who wrote the message.

Example MAC: **CBC-MAC**

Turning a block cipher into a MAC

CBC MAC

$C_0 = 0$ [any fixed IV]
$\mathrm{C_i} = \mathrm{E_K}(\mathrm{M_i} \oplus \mathrm{C_{i-1}})$
$\mathrm{MAC_K}(\mathrm{M_1} \ldots \mathrm{M_x}) = \mathrm{C}_n$



Differences with respect to CBC Encryption:

- CBC-MAC is deterministic
- Only output of CBC-MAC is the last block

Limitation: Only secure if length of message M is known

CBC can be turned into a MAC by:

Fix the IV (for example 0) and then do CBC. The MAC is the output of the last encryption block: one could have only gotten there for one message and one key. Note that the last block cannot be used to recover anything about the message: it looks random (as it is the output of a block cipher)

If we use CBC to produce a MAC we have some crucial differences to CBC Encryption: As the IV is fixed, the result for one message is always the same (no value changes!) As opposed to all blocks we will only send last cipher block.

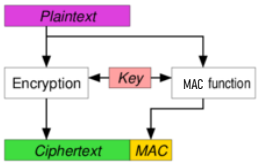
It is only secure because if the length of m is known. This is because the length of the message determines the output of which block is the MAC. But, if Alice does not know the length of the message, it is easy to get a MAC for an extension of the message:
If you have T= CBC-MAC(k,M), and T'=CBC-MAC(k,M'); then T' is a correct CBC MAC for M || T XOR M' where || is concatenation.

And all together now…

Message confidentiality and integrity for symmetric encryption schemes

How to obtain confidentiality and integrity?

ENCRYPT-AND-MAC



✘ No integrity on the ciphertext → Cipher can be attacked need to decrypt to know if valid

✓ Integrity of the plaintext can be verified

✘ May reveal information about the plaintext → repeated msg, recall the IV of the MAC is fixed (can be solved with a counter)

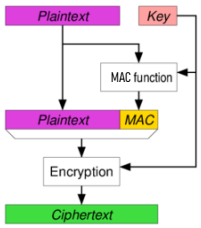
Bellare, M., & Namprempre, C. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *International Conference on the Theory and Application of Cryptology and Information Security*, 2000.
Bellare, M., Kohno, T., & Namprempre, C. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security*, 2004.
https://en.wikipedia.org/wiki/Authenticated_encryption

The MAC is computed on the plaintext, so it cannot protect the ciphertext. Thus, to check whether it is correct one needs to decrypt (and decrypting is expensive).

Because the MAC is deterministic, it may reveal information about the message (e.g., if the message is sent twice the MAC is the same for both messages)

How to obtain confidentiality and integrity?

MAC-THEN-ENCRYPT



✗ No integrity of ciphertext
(in theory) possible to change ciphertext and have a valid MAC
need to decrypt to know if valid

✓ Integrity of the plaintext can be verified

✓ No information on the plaintext either, since it is encrypted

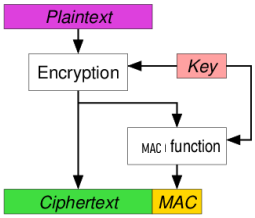
Bellare, M., & Namprempre, C. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *International Conference on the Theory and Application of Cryptology and Information Security*, 2000.
Bellare, M., Kohno, T., & Namprempre, C. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security*, 2004.
https://en.wikipedia.org/wiki/Authenticated_encryption

Still no check on the ciphertext

But no info on the MAC because the adversary does not see it (only sees the encrypted blob)

How to obtain confidentiality and integrity?

ENCRYPT-THEN-MAC



- ✓ Integrity of ciphertext → ensures you only read valid messages! Cipher cannot be attacked!
- ✓ Integrity of the plaintext can be verified
- ✓ No information on the plaintext either, since it is encrypted

Bellare, M., & Namprempre, C. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *International Conference on the Theory and Application of Cryptology and Information Security*, 2000.
Bellare, M., Kohno, T., & Namprempre, C. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security*, 2004.
https://en.wikipedia.org/wiki/Authenticated_encryption

Conclusions

What we have learned about applied cryptography

Symmetric cryptography

- Confidentiality: Stream ciphers, Block ciphers (modes of operation!)
- Integrity / Authentication: Message Authentication Codes (MACs)

Asymmetric cryptography

- Confidentiality: Encryption
- Integrity / Authentication: Digital signatures

Hash functions

- Three security properties
- Support Digital Signatures + other functions