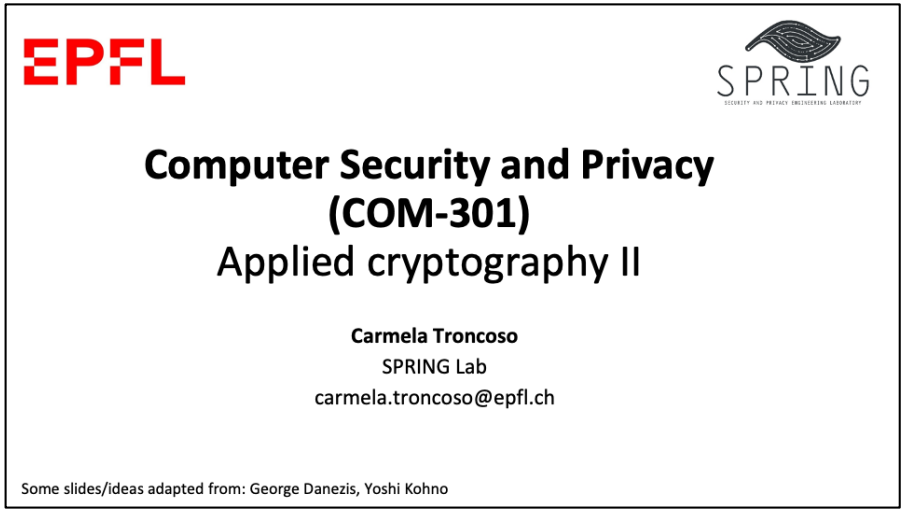
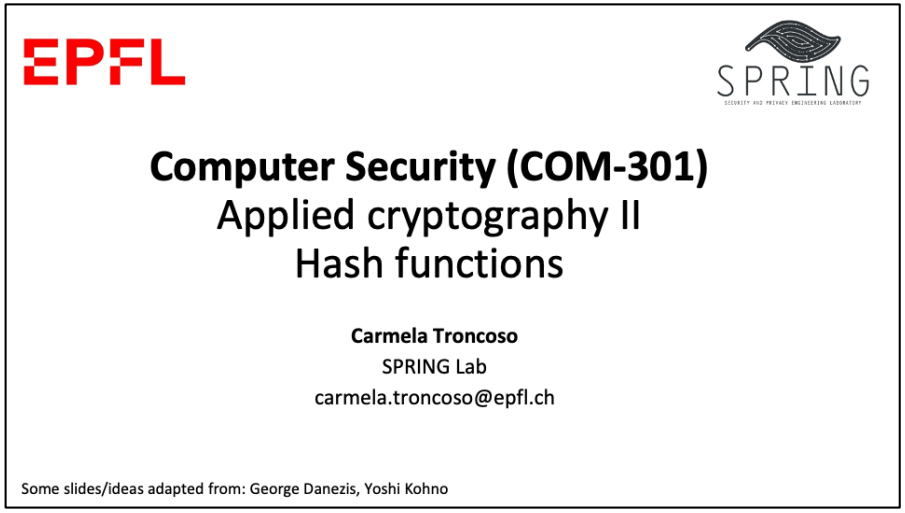
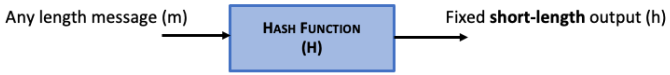

EPFL
SPRING
SECURITY AND PRIVACY ENGINEERING LABORATORY
Computer Security and Privacy
(COM-301)
Applied cryptography II
Carmela Troncoso
SPRING Lab
carmela.troncoso@epfl.ch
Some slides/ideas adapted from: George Danezis, Yoshi Kohno


EPFL
SPRING
SECURITY AND PRIVACY ENGINEERING LABORATORY
Computer Security (COM-301)
Applied cryptography II
Hash functions
Carmela Troncoso
SPRING Lab
carmela.troncoso@epfl.ch
Some slides/ideas adapted from: George Danezis, Yoshi Kohno

Hash functions



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)

Hash functions

THREE SECURITY PROPERTIES

PRE-IMAGE RESISTANCE

Given H(m), difficult to find m

SECOND PRE-IMAGE RESISTANCE

Given m, difficult to find an m'!=m such that H(m') = H(m)

COLLISION RESISTANCE

Difficult to find any m, m' such that H(m) = H(m')

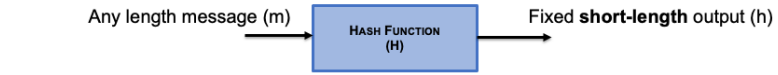
Cryptographic hash functions are designed to be fast and efficient to compute, and they have several important properties that make them valuable in various applications:

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



Three security properties

Pre-image resistance

Given H(m), difficult to find m

Second pre-image resistance

Given m, difficult to find an m'!=m such that H(m') = H(m)

Collision resistance

Difficult to find any m, m' such that H(m) = H(m')

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

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 H(m), difficult to find m

SECOND PRE-IMAGE RESISTANCE

Given m, difficult to find an m'!=m such that H(m') = H(m)

COLLISION RESISTANCE

Difficult to find any m, m' such that 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:

Hash functions

THREE SECURITY PROPERTIES

PRE-IMAGE RESISTANCE

Given H(m), difficult to get m

SECOND PRE-IMAGE RESISTANCE

Given m, difficult to get an m'!=m such that H(m') = H(m)

COLLISION RESISTANCE

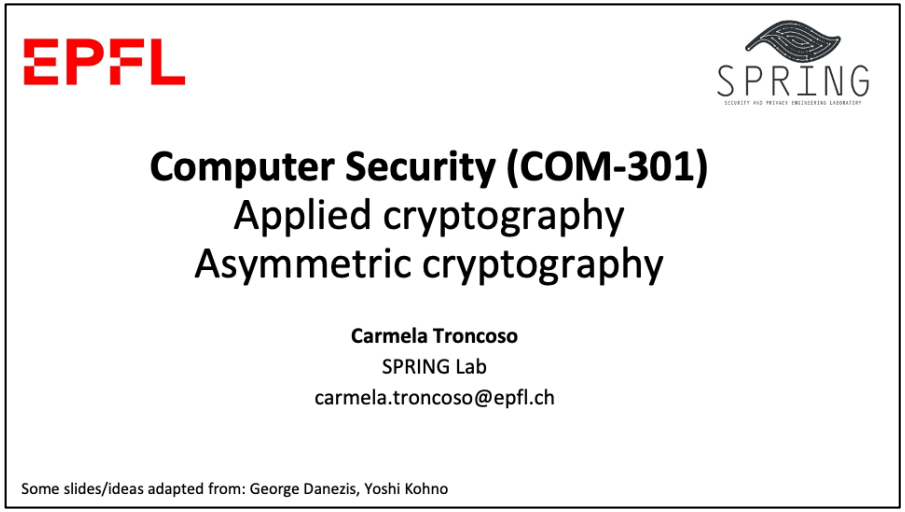
Difficult to fi

USES

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

An HMAC is a MAC built using hashes. They are hard to build

Even though it appears in the Computer Networks book, **H(K||m) is not a good HMAC**, this is because for some hash functions (those based on the so-called Merkle-Damgard structure) one can do length extension attacks similar to those in CBC-MAC -- see extra slides at the bottom).


EPFL
SPRING
SECURITY AND PRIVACY ENGINEERING LABORATORY
Computer Security (COM-301)
Applied cryptography
Asymmetric cryptography
Carmela Troncoso
SPRING Lab
carmela.troncoso@epfl.ch
Some slides/ideas adapted from: George Danezis, Yoshi Kohno

Symmetric Cryptography

Block ciphers, Stream Ciphers, MACs

Gru and Bob need to **share** a **secret** key

Secure key distribution is a problem!

Diffie, Whitfield, and Martin Hellman. "New directions in cryptography." Information Theory, IEEE Transactions on 22.6 (1976): 644-654

Even though symmetric keys are small, the fact that Gru and Bob need to share a key is problematic:

1) how to send it?
2) how many keys do we need to ensure that everyone can talk with everyone?

Asymmetric cryptography

Each participant has **two** keys:

- One **secret** key that only they know
- One **public** key that they can reveal

Pairs of (secret, public) keys are created with specific algorithms

Secret Key: SK_{Bob}

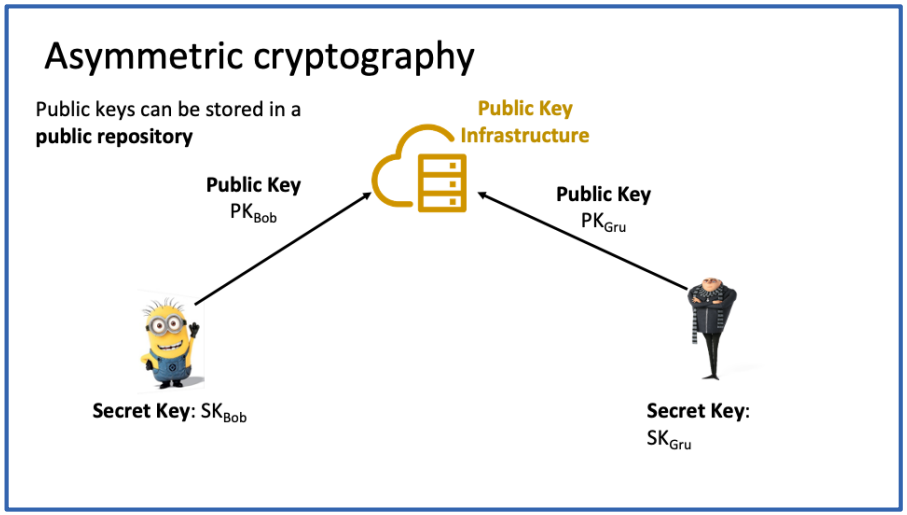
Public Key: PK_{Bob}

Secret Key: SK_{Gru}

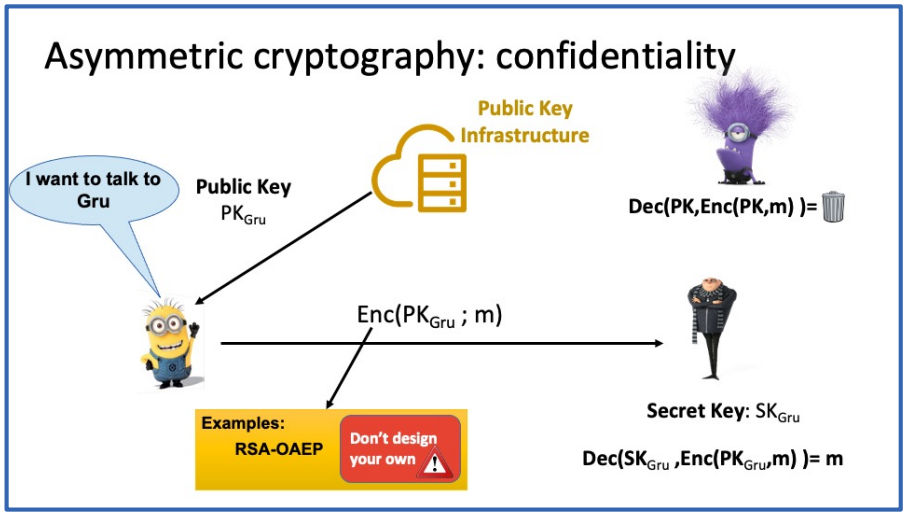
Public Key: PK_{Gru}

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*

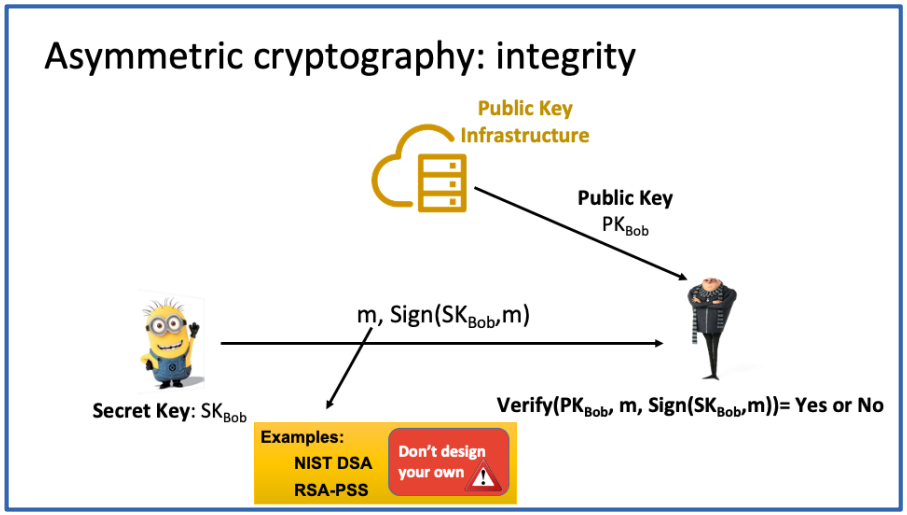


The public keys can be uploaded to public servers that form the backbone of so-called **Public Key Infrastructure.**

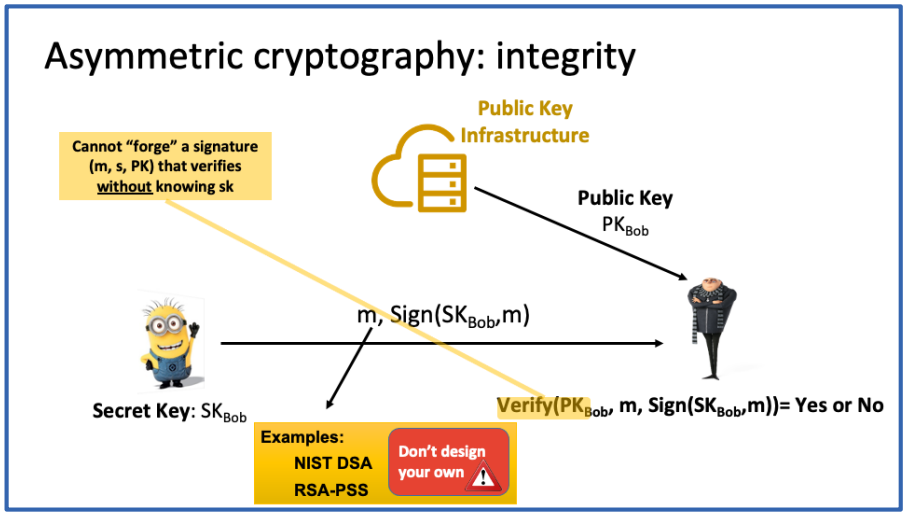


When Bob wants to write to Gru:

- Finds Gru's public key in the repository.
- Encrypts the message with Gru's public key
- This message **can only** be decrypted with Gru's secret key (decrypting a message with the public key returns garbage)



To verify a signature, Gru gets Bob's public key from the repository.



This enables Gru to *not only* know the message has not been modified, but also that he is talking to Bob: only Bob, that knows the secret key, could have created the signature

Digital Signatures

Properties:

- Integrity of message
- Authenticity sender
- *Non-repudiation* (why are they different from MACs?)

Application: **Public Key Infrastructure**: Certificates

(1) Authority signs a mapping between names, or names and encryption public keys.

(2) Authority signs mapping between names and verification keys.

Digital signatures provide:

- **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)
- **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.
- **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). This is different than symmetric key MAC mechanisms, where *both* sender and receiver of a message can produce the MAC(k,m) and both can claim that the message was sent by the other party.

While typically the signature is sent along with the message, and thus confidentiality is not a consideration, signatures do generally provide **message confidentiality.** Given a signature, it is generally not possible to recover the message.

Digital signatures are used in the Internet Public Key infrastructure to sign

Certificates that authenticate the web servers / domain names we use in the internet.

Digital Signatures

Properties:

- Integrity of message
- Authenticity sender
- *Non-repudiation* (why are they different from MACs?)

Encryption key pair != Signature key pair ⚠

Application: **Public Key Infrastucture:** Certificates

(1) Authority signs a mapping between names, or names and encryption public keys.

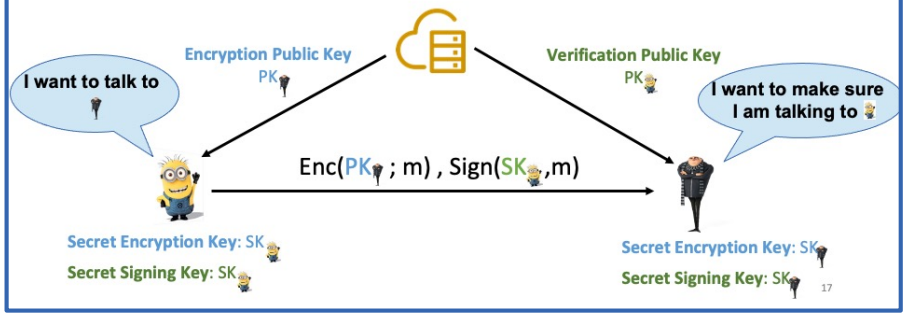
(2) Authority signs mapping between names and verification keys.

All together

ASYMMETRIC CRYPTOGRAPHY

Users have two pairs of keys (secret key SK, public key PK)

Confidentiality	**Dec(SK,Enc(PK,m))= m**
Integrity/Authentication	**Sig(SK,m) = s; Verify(PK,Sig(SK,m)) = YES/NO**



Asymmetric cryptography limitations

Computationally costly compared with most symmetric key algorithms of equivalent security

Signing and encrypting **is slow**

Not suitable to encrypt **large amounts of data**
There are not good "cipher modes"

In practice
Sign hash of messages

Hybrid encryption
(only encrypt small symmetric key)

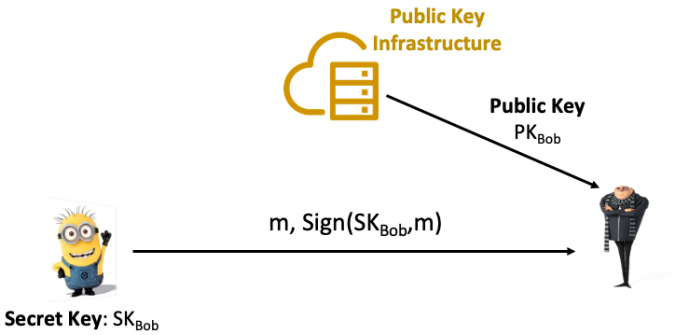
18

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).

Also, there are not good cipher modes to encrypt large messages: we can only do one block.

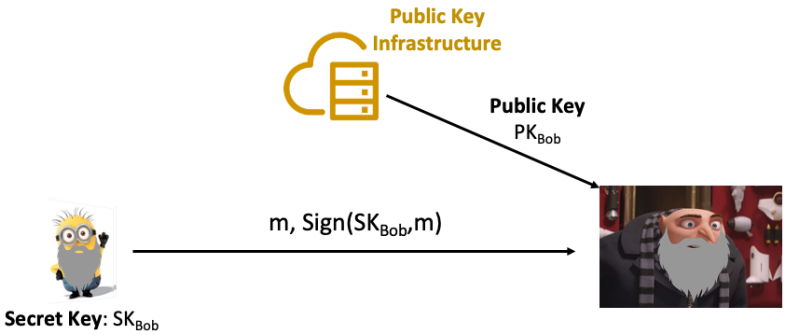
To overcome these shortcomings instead of signing long messages we sign a hash, and to encrypt we use hybrid encryption (see next slides)

Digital signatures on hash functions

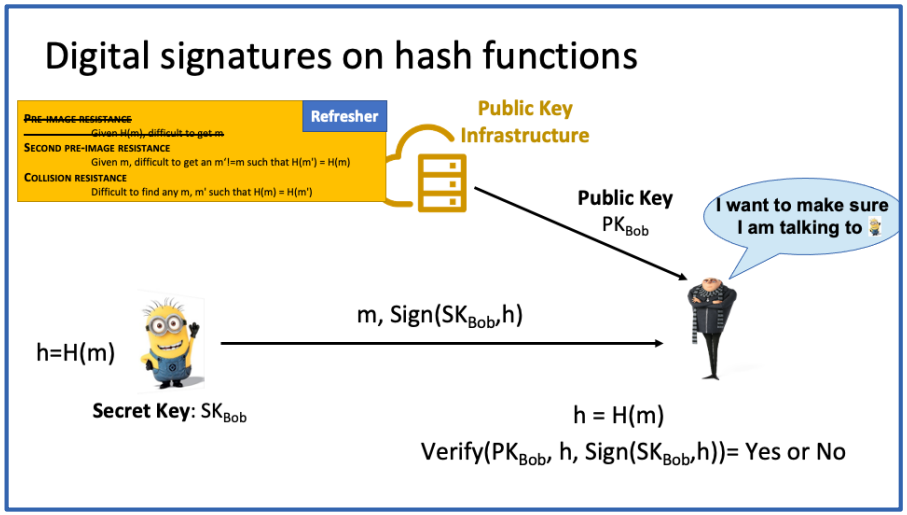


Signing is a slow property.

Digital signatures on hash functions



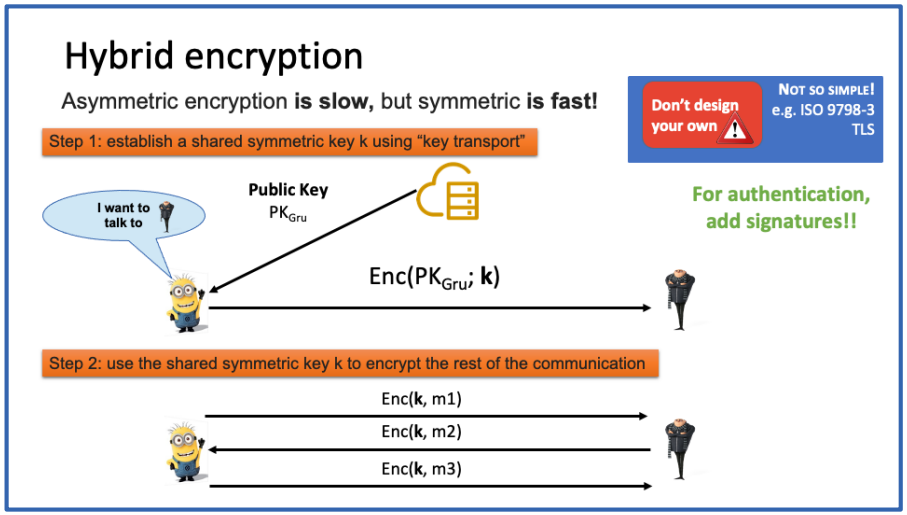
If we sign long messages, both signing and verification will take a very long time and will be not practical



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.

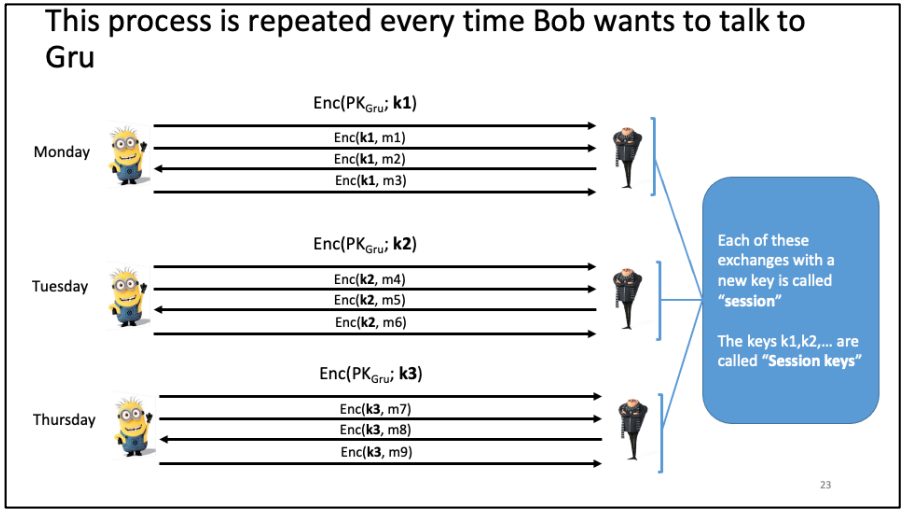
Collision resistance says that you cannot produce two messages that lead to the same hash. That means that you cannot have someone sign a message m, and then claim that they signed anoter message m'



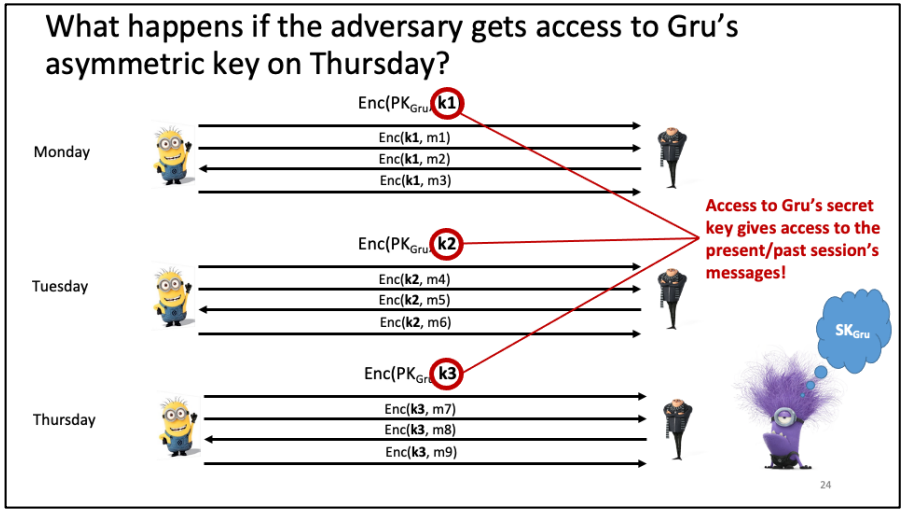
In Hybrid encryption the idea is to get the best of the both worlds.

To avoid sharing keys we use public keys, but as we cannot encrypt a lot with them, we use to send a symmetric key

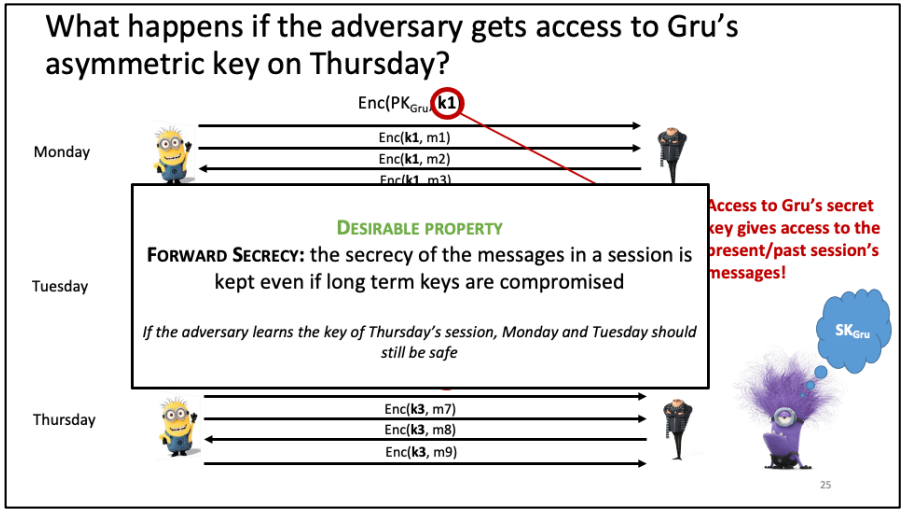
And then we use this symmetric key to encrypt the rest of the communication



Every time Bob communicates with Gru, he can create a new key. These are called session keys.



But there is a problem in this scheme: If the adversary gets hold of Gru's secret key , the secrecy of all past sessions is compromised



We want to avoid this. A key compromise at time **t** should not compromise the secrecy of any past conversation.

Key agreement for forward secrecy – The Math

Arithmetic modulo a number: clock arithmetic

- 6 (mod 12) = 6 (mod 12)
- 12 (mod 12) = 0 (mod 12)
- 14 (mod 12) = 2 (mod 12)

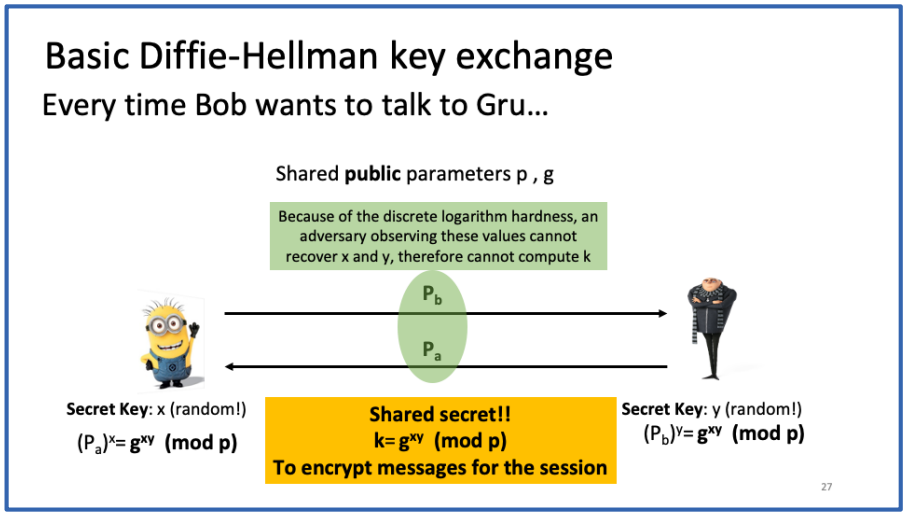
Arithmetic modulo a large prime p (>1024 bits)

- Addition and multiplication (mod p) can be computed
- Exponentiation can be computed [Given (a, x) → a^x mod p?]
- Discrete logarithms are **HARD**! [Given (a, a^x mod p) → x?]

The main math that we use to be able to build a shared secret is *modular discrete arithmetic*

Concretely modular discrete arithmetic in which the modulo is a large prime.
When the modulo is a large prime, addition, multiplication, and exponentiation, are easy to compute.

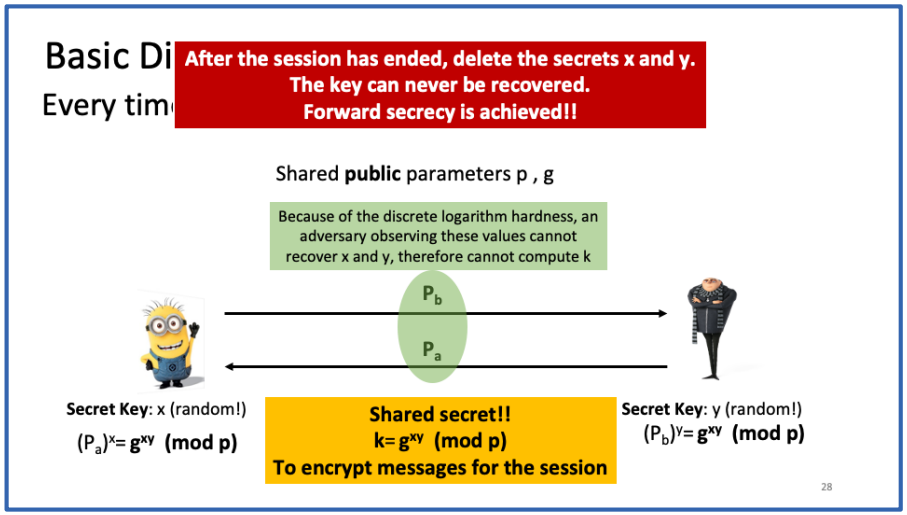
However, computing a discrete logarithm is **hard.** This means that, if the modulo p is well chosen, given a^x mod p, it is not computationally feasible to compute x



A Diffie-Hellman key exchange exploits the **hardness** of the discrete logarithm problem to enable 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 sends to Gru $g^x \bmod p$, where x is Bob's secret key; and Gru sends to Bob $g^y \bmod p$, where y is Gru's secret key. With these values, both Gru 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 $P_b = g^x \bmod p$).



Once a session ends, Gru and Bob delete their secrets x and y. Like this, because there is no any record of these values, the key k can never be recovered.

A system based on establishing shared keys using Diffie-Hellman is forward secure **as long as in every session new fresh secret keys are used.**

Summary of the crypto lectures

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

Hybrid encryption

best both worlds!

Forward secrecy

Diffie Hellman