Lecture 3: Authenticity (Data Origin: MAC & Signature)

3.1. Crypto background 1: Public Key Cryptography (PKC)

3.2 Crypto background 2: Cryptographic hash

3.3 Data integrity (Hash)

3.4 Data authenticity (MAC, Signature)

3.5 Some attacks & pitfalls:

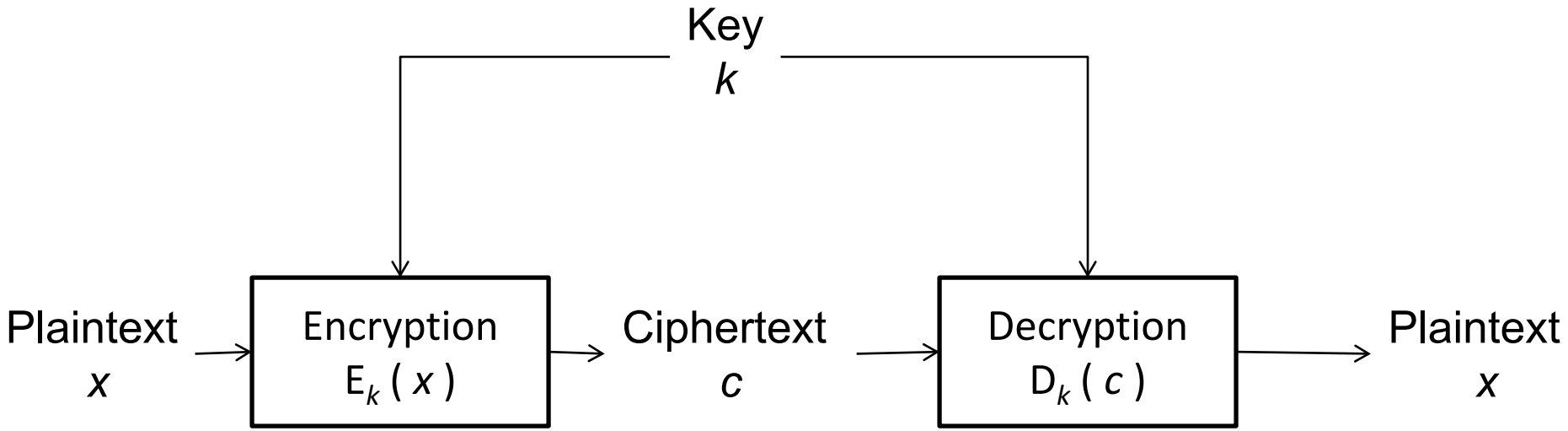
- 3.5.1 Birthday Attack on hash
- 3.5.2 Design flaw: Using encryption for authenticity

3.6 Application of hash: Password file protection (revisited)

3.1. Crypto Background: Public Key Cryptography (PKC)

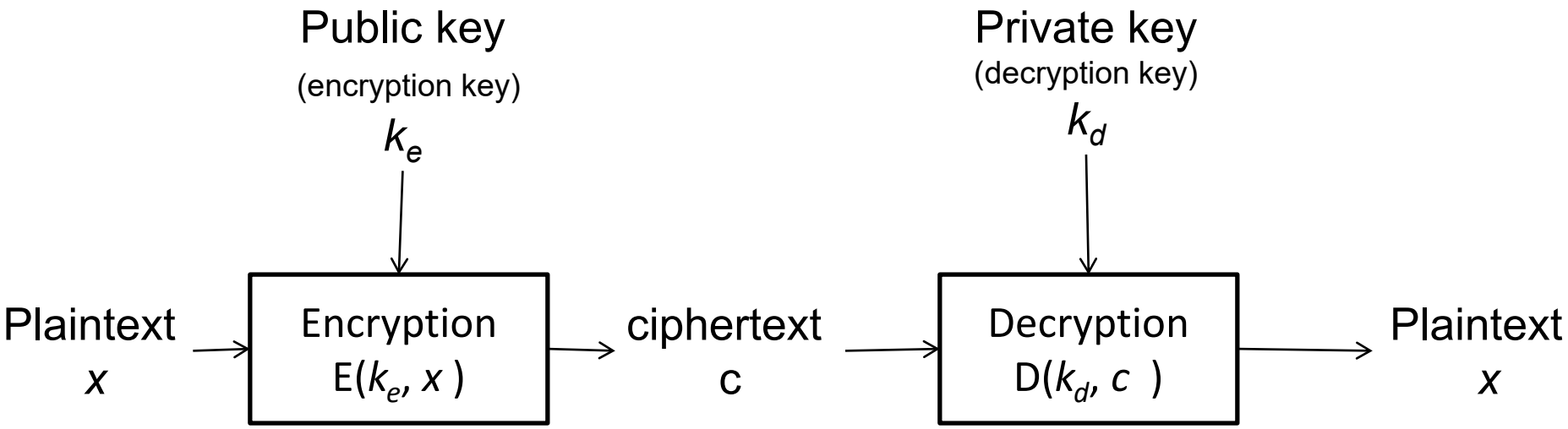
Overview: Symmetric-Key Scheme

A symmetric-key encryption scheme uses the same key for both encryption and decryption



Overview: Public-Key Scheme

A public-key (asymmetric-key) scheme uses *two different keys* for encryption and decryption



Note:
Very often, the *decryption key* consists of two parts $<k_e, k_d>$, where k_e is the encryption (public) key.
When it is clear in the context, we also call k_d the private key, although both k_e and k_d are required during decryption.

Why is It Called "Public Key"?

- In a typical usage, the owner Alice *keeps her private key secret*; but tells everyone of her public key (e.g. by posting it in her Facebook)
- A following usage scenario is possible
- Suppose Bob has a ***plaintext x*** for Alice.
  Bob can encrypts it (using **Alice's *public key***),
  and posts the ***ciphertext c*** on Alice's Facebook.
- Another entity, Eve, also obtains Alice's public key and the posted ciphertext *c* from Alice's Facebook page.
  Yet, without Alice's private key, Eve is unable to derive *x*.
- Alice, with her secret ***private key***,
  can decrypt and obtain the plaintext *x*

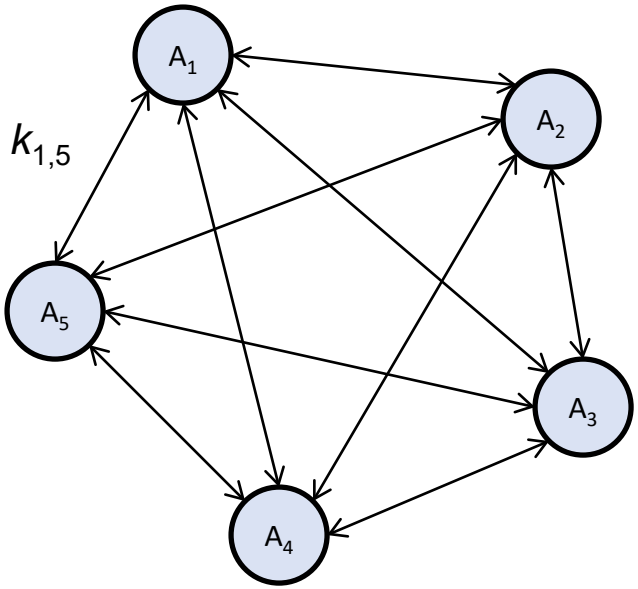
Security Requirements of a Public Key Scheme

- Two *security requirements*:
  1. Given the public key and the ciphertext, but not the private key, it is difficult to determine the plaintext
  2. Without a knowledge of the private key, the ciphertext resembles a sequence of random values
- The requirements imply that it must be difficult to get the private key from the public key

Public-Key Scheme and Key Management Issue

- Suppose we have multiple entities, A_1, A_2, …, A_n:
  - Each of them can compute a pair of <private key, public key>
  - Each announces his/her public key, but keeps the private key secret
- Now, suppose A_i wants to encrypt a message m to be read only by A_j:
  - A_i can uses A_j's public key to encrypt m
  - By the property of PKC, only A_j can decrypt it
- If we don't use the PKC, then, any two entities must share a symmetric key:
  - Many keys are required especially if the no of entities is large
  - Establishing all the secret keys is also difficult

Number of Keys Required (with *n* Entities)

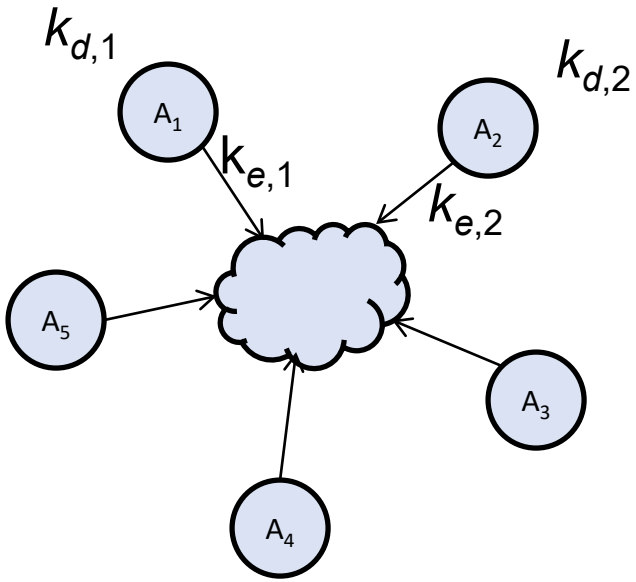


Symmetric-key setting:

Every pair of entities requires one key

Let $k_{i,j}$ be the key shared by A_i and A_j

Total number of keys needed: ***n*(*n*-1)/2**



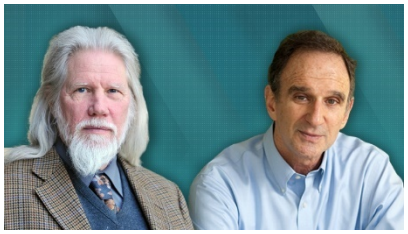
Public-key setting:

Each entity A_i publish its public key $k_{e,i}$ and keep its private key $k_{d,i}$

Total number of public keys: ***n***
Total number of private keys: ***n***

Popular PKC Schemes

- RSA: Key size ~2,048 bits (256 bytes)
- ElGamal: ElGamal can exploit techniques in Elliptic Curve Cryptography (ECC), thus reducing the key size to ~ 300 bits
- Historical notes:



The idea of an asymmetric public-private key cryptosystem is attributed to Whitfield Diffie and Martin Hellman, who published the concept in 1976

Diffie and Hellman, 2015 Turing Award winners

Ron Rivest, Adi Shamir, and Leonard Adleman proposed RSA algorithm in 1977

Rivest, Shamir and Adleman, 2002 Turing Award winners

"Classroom" RSA

Optional

Public key: a composite modulus n, the encryption exponent e.

- The modulus n is the product of two large primes p and q, i.e. $n=pq$
- The values of p and q are not revealed to the public

Private key: d

The relationship among the 3 numbers $n=pq$, e, d:

$$de \mod (p-1)(q-1) = 1$$

Encryption: Given message m, compute the ciphertext c:

$$c = m^e \mod n$$

Decryption: Given ciphertext c, compute the plaintext m:

$$m = c^d \mod n$$

(Note: For RSA mathematical background, see: [PF12.3] or
Susanna Epp, "Discrete Mathematics with Applications", 4th ed, Chapter 8, Section 4)

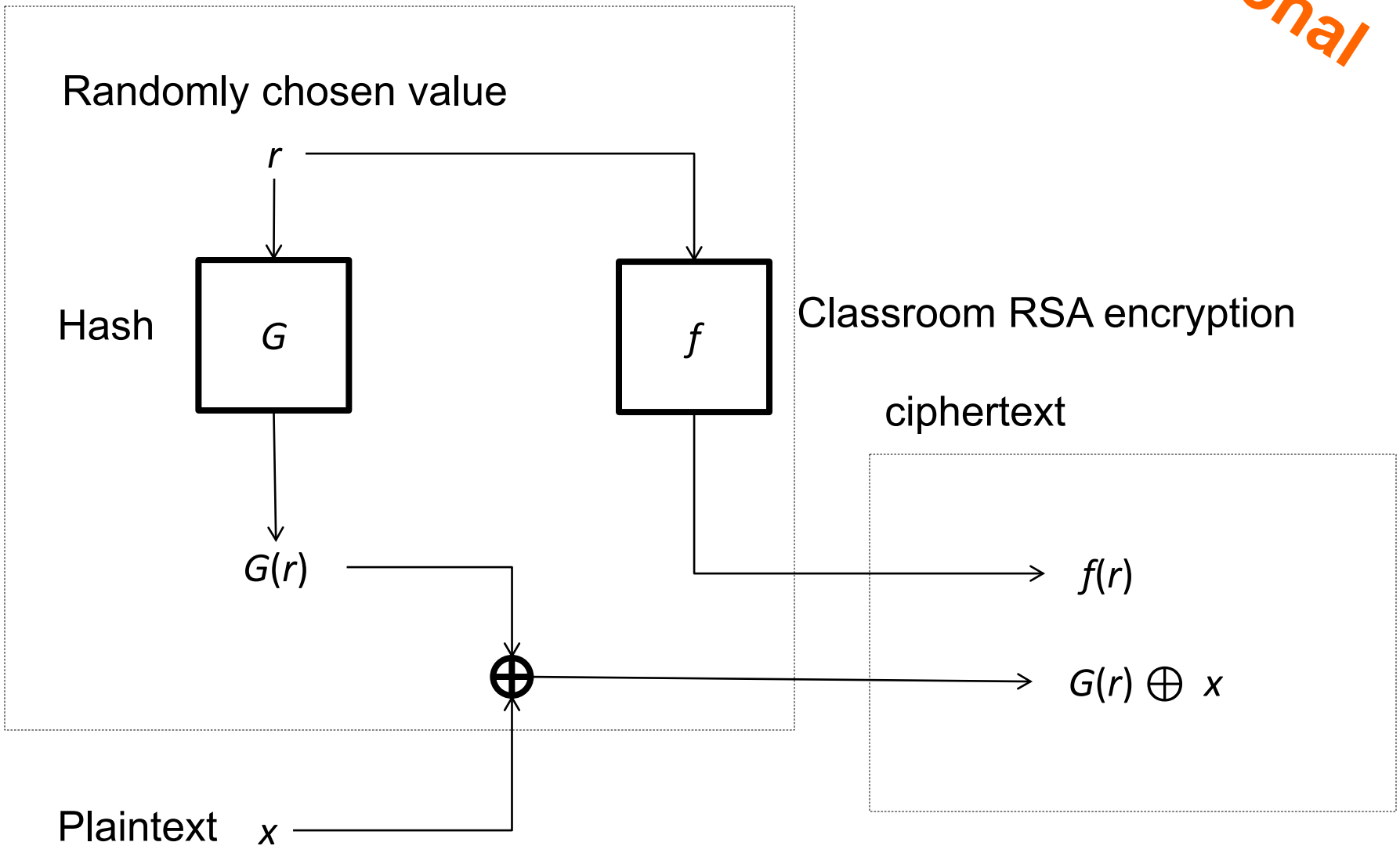
RSA Standards

- Similar to symmetric-key encryption, some form of **IV** is required so that encryption of the ***same*** plaintext at different times would give different ciphertexts
- Hence additional mechanisms need to be added to the "classroom" RSA
- The standard Public-Key Cryptography Standards (PKCS)#1 (http://en.wikipedia.org/wiki/PKCS_1) adds "optimal padding" to achieve the above

(**Note**: the "classroom RSA" is not "semantic secure", and leaks some information about the plaintext. PKCS#1 attempts to fix such a problem.)

RSA with Padding (Just to illustrate how complicated it could be)

Optional



Pitfall: Using RSA in the Symmetric-Key Setting

Consider a sample scenario:

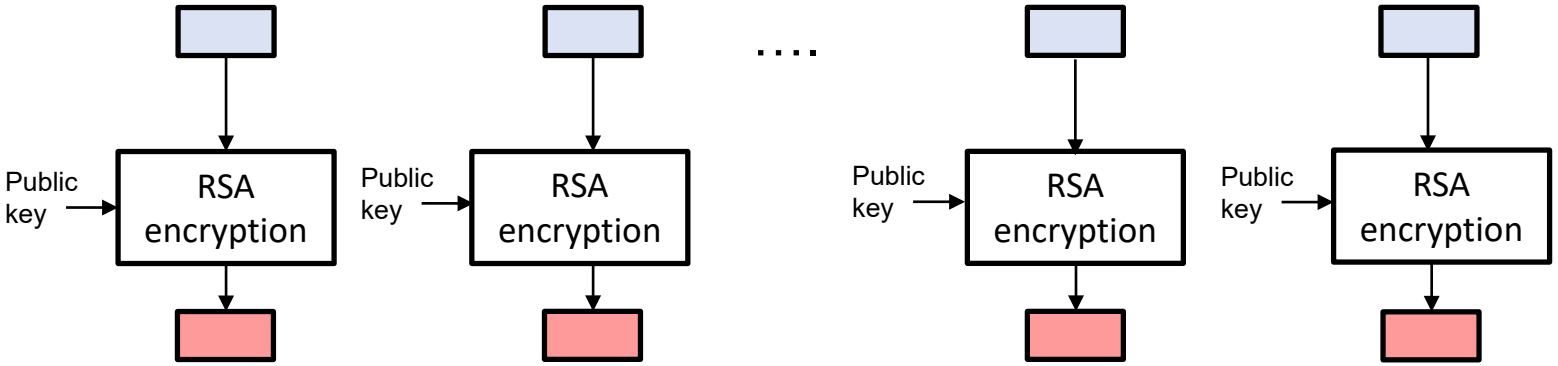
"For his Final Year Project, Bob is tasked to write an app that employs an end-to-end encryption to send images from a mobile phone to another mobile phone over WiFi. The sender and recipient use SMS to establish the required symmetric key."

- Bob feels that RSA is cool ☺
- Instead of employing AES, Bob employs RSA as the encryption scheme: the public/secret key pair is treated as the symmetric key
- Bob claims that RSA is "more secure" than AES: it can be proved to be as difficult as factorization, which is believed to be hard

Pitfall: Using RSA in the Symmetric-Key Setting

- Bob's RSA implementation:
  To encrypt a large image (say 1MB),
  Bob divides the file into chunks of 128 bytes,
  and then apply RSA to each chunk

- Implementation diagram:



Issue 1: Efficiency/Performance

RSA is significantly slower than AES:

- Comparing with the same key strength (e.g. by following NIST recommendation): 128-bit AES ≈ 3072-bit RSA
- Crypto++ Benchmarks (https://www.cryptopp.com/benchmarks.html): a C++ library

Algorithm	MiB/Second	Cycles Per Byte	Microseconds to Setup Key and IV	Cycles to Setup Key and IV
AES/GCM (2K tables)	102	17.2	2.946	5391
AES/GCM (64K tables)	108	16.1	11.546	21130
AES/CCM	61	28.6	0.888	1625
AES/EAX	61	28.8	1.757	3216
AES/CTR (128-bit key)	139	12.6	0.698	1277
AES/CTR (192-bit key)	113	15.4	0.707	1293
AES/CTR (256-bit key)	96	18.2	0.756	1383

In different order of magnitude!

Operation	Milliseconds/Operation	Megacycles/Operation
RSA 1024 Encryption	0.08	0.14
RSA 1024 Decryption	1.46	2.68
RSA 2048 Encryption	0.16	0.29
RSA 2048 Decryption	6.08	11.12

Solution

When a large file ***F*** is to be encrypted under the public-key setting, for efficiency, the following steps are carried out:

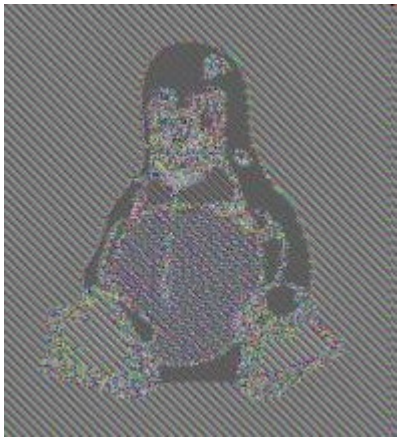
1. Randomly choose an AES key ***k***
2. Encrypt ***F*** using AES with ***k*** as the key, to produce the ciphertext ***C***
3. Encrypt ***k*** using RSA to produce the ciphertext ***q***
4. The final ciphertext consists of two components: (***q***, ***C***)

Question: What about decryption steps?

Issue 2: Extension to Multiple Blocks

- Dividing the plaintext into pieces, and independently apply encryption to each of them *could leak* important information!
- Note that this issue applies to symmetric-key encryption as well
- Suppose the image below is divided into blocks, and encrypted with some *deterministic encryption scheme** using the same key
- Since it is deterministic, any two plaintext blocks that are exactly the same (e.g. from the white background) will be encrypted into the same ciphertext

Plaintext



Ciphertext

Issue 2: Extension to Multiple Blocks

Some additional notes: *

- An encryption scheme is "**deterministic**" in a sense that the encryption algorithm always produces **the same output** (i.e. ciphertext) when given the same input (i.e. the key and plaintext)
- An example: AES without the IV
- In contrast, a "**probabilistic**" encryption scheme produces **different ciphertexts** even with the same input is given
- AES is deterministic.
  However, if we employ **AES with a randomly-chosen IV**, then it becomes probabilistic (since the IV is different)

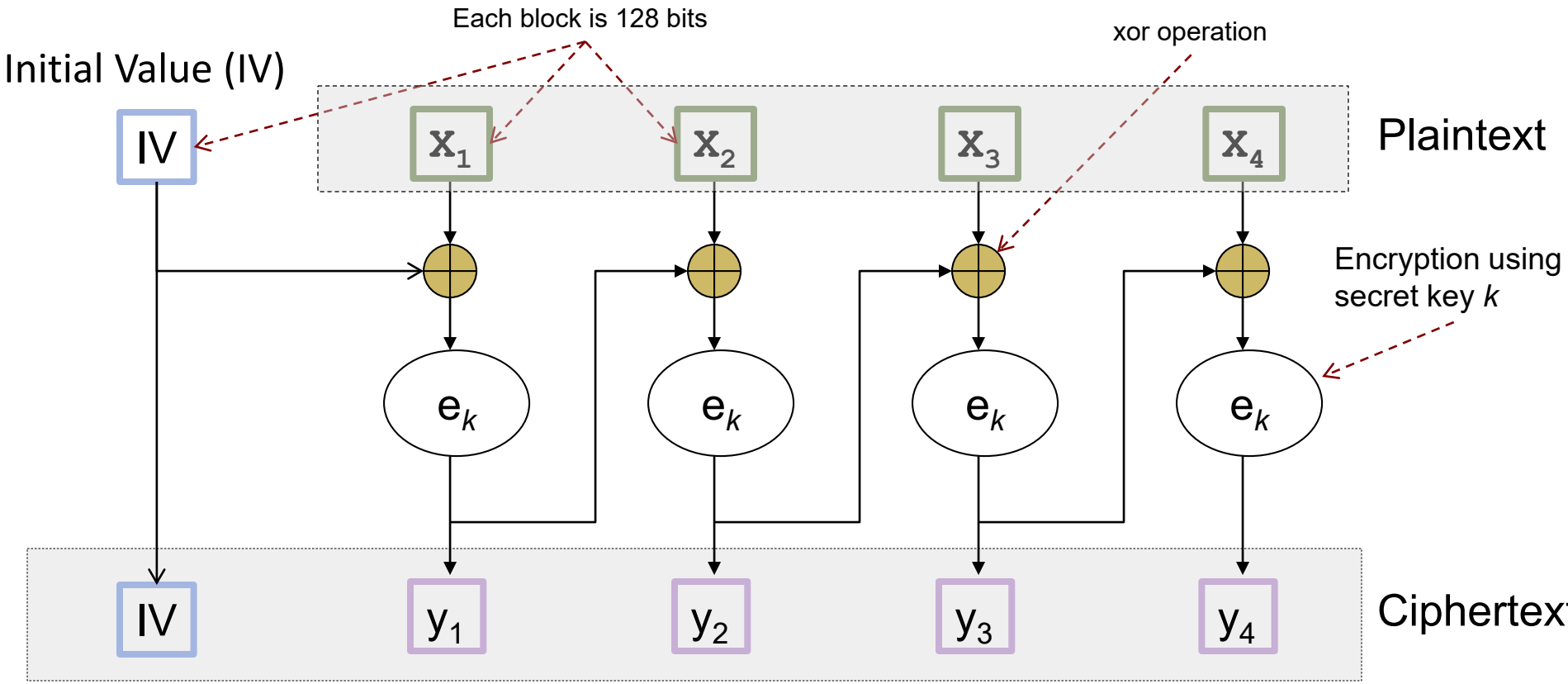
Solution

- To prevent the leakage of information shown in the previous slide, **additional mechanisms** are required
- A ***mode-of-operation*** describes how the blocks are to be "**linked**" so that different blocks at different locations would give different ciphertext, even if all the blocks have the same content
- Popular mode-of-operation is *Cipher Block Chaining* (CBC)
- Avoid the *Electronic Codebook (ECB)* mode, where "we can see the penguin"!

Question: Why not just randomly choose an IV **for each block**, and hence achieve a probabilistic encryption so as to prevent the leakage?

(Answer: It will significantly increase the size of the final ciphertext. See also Slide 64.)

Mode-of-Operation: Cipher Block Chaining (CBC) on AES



Note: In the above figure, we treat **IV as part of the final ciphertext**. The terminology is inconsistent in the literature. Some documents may state that "the final message to be sent are the IV and the ciphertext" (i.e. IV is not called the "ciphertext"). In this module, when it is crucial, we will explicitly state whether IV is excluded (e.g. AES without IV).

Issue 3: Security of RSA

- RSA is not necessary "more secure" than AES
- It can be shown that, getting ***the private key from the public key*** is as difficult as factorization
- But, it is not known whether the problem of getting ***the plaintext from the ciphertext and public key*** is as difficult as factorization
- As mentioned before, the "classroom" RSA has to be modified so that different encryptions of the same plaintext lead to different ciphertexts (i.e. probabilistic)
- Such modification is not straightforward (e.g. PKCS#1)

Remarks on PKC Strengths

- The main strength of RSA and PKC is the "public-key" setting, which allows an entity in the public to perform an encryption **without** a (pre-established) pair-wise *secret key*
- This "secret-key-less" feature is also very useful for providing ***authentication***: more in this lecture later
- In practice, PKC is rarely used to encrypt a large data file (as already discussed earlier)

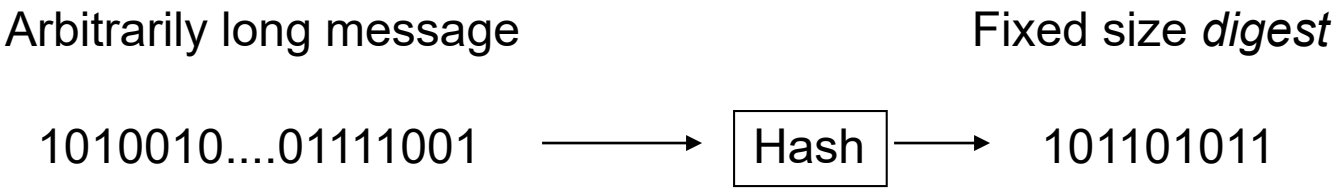
Question:
Assuming that a PKC encryption scheme on average takes 1M cycles to decrypt 16 bytes.
How long does it take to decrypt a 4 GBytes movie on a single-core 4GHz chip?

(For comparison, note that RSA-OAEP take 42M cycles per invocation [Gollmann] pg272.)

3.2. Crypto Background: Cryptographic Hash

Hash (With No Secret Key Involved)

A (cryptographic) hash is a function that takes an arbitrarily long message as input, and outputs a *fixed-size* (say 160 bits) ***digest***

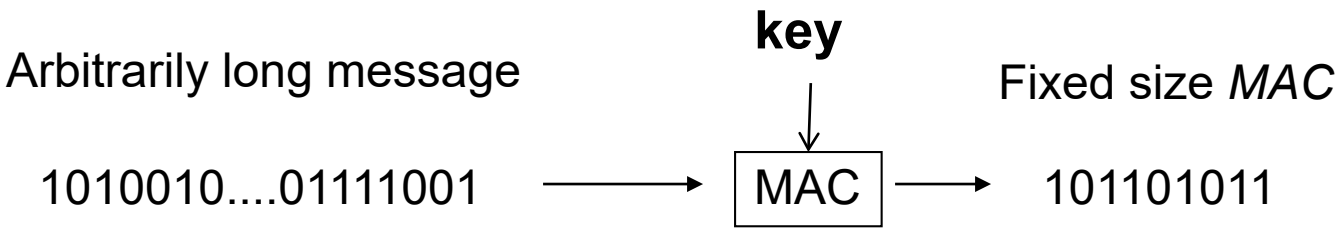


Security requirements:

- "Preimage resistant":
  given a digest *d*, it is difficult to find a *m* such that $h(m) = d$
- "Second-preimage resistant":
  given m_1, it is difficult to find a second preimage $m_2 \neq m_1$
  such that $h(m_1) = h(m_2)$
- "Collision-resistant":
  it is difficult to find **two different messages** $m_1 \neq m_2$
  that "hash" into the same digest, i.e. $h(m_1) = h(m_2)$

MAC (aka Keyed-Hash, with a Secret Key *Involved*)

A keyed-hash is a function that takes an arbitrarily long message and **a secret key** as input, and outputs a fixed-size (say 160 bits) ***MAC (Message Authentication Code)***



Security requirement:

- Without knowing the key, it is difficult to forge the MAC

Popular Hash

- SHA-0, SHA-1, SHA-2, SHA-3

Some historical notes:

- **SHA-0** was published by NIST in **1993**.
  It produces a 160-bits digest.
  It was withdrawn shortly after publication and superseded by the revised version SHA-1 in **1995**.
- In **1998**, an attack that finds collision of SHA-0 in $\mathbf{2^{61}}$ operations was discovered.
  (Simply using the straight forward birthday attack,
  a collision can be found in $2^{160/2} = \mathbf{2^{80}}$ operations).
  In **2004**, a collision was found, using 80,000 CPU hours.
  In **2005**, Wang Xiaoyun et al. (Shandong University) gave an attack that can finds collision in $\mathbf{2^{39}}$ operations.

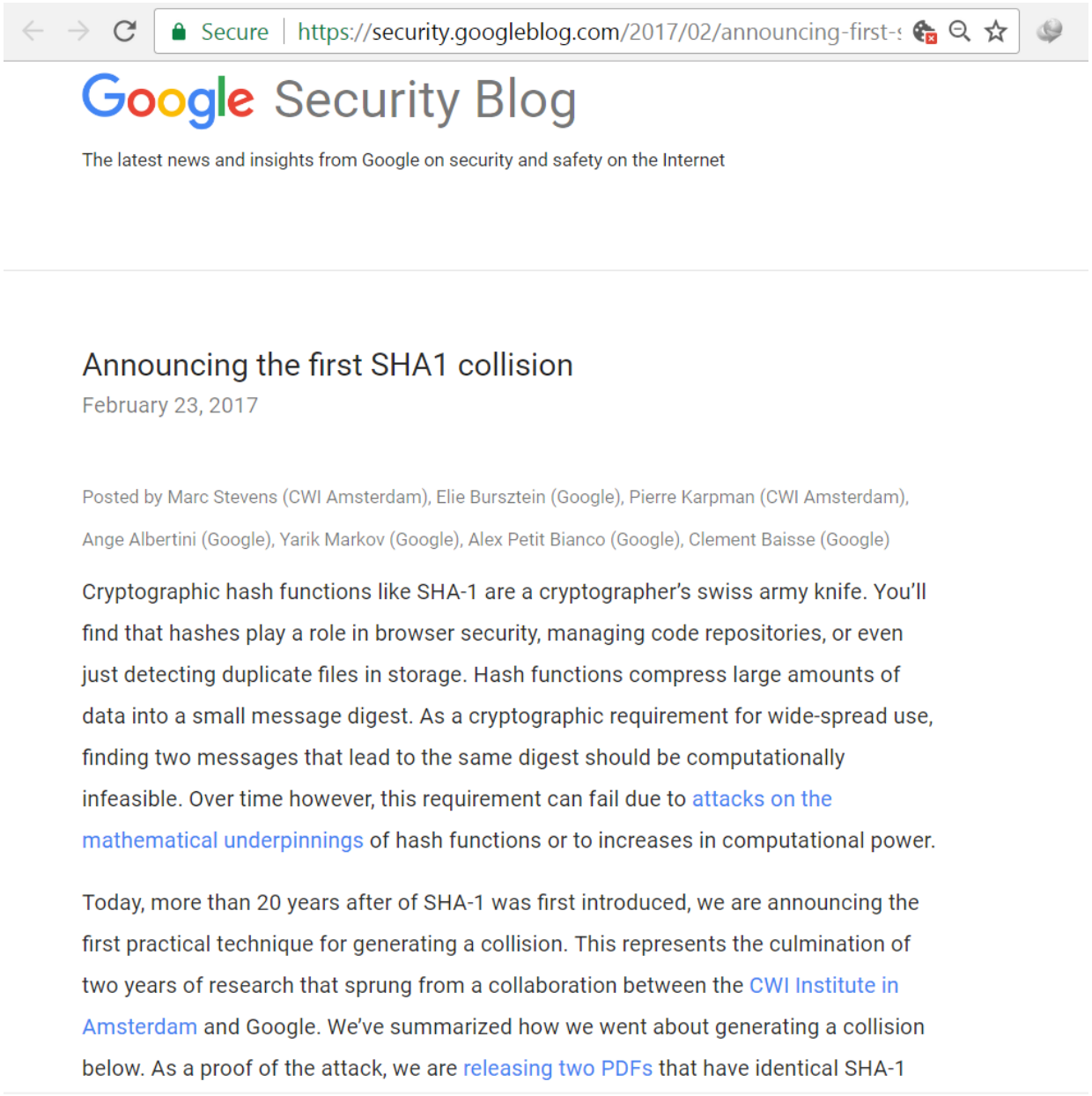
Popular Hash (Historical Notes)

- **SHA-1** is a popular standard. It produces 160-bits message digest.
  It is employed in SSL, SSH, etc.
- In **2005**, Xiaoyun Wang et al. gave a method of finding collision in SHA-1 using 2^{69} operations, which was later improved to 2^{63}.
  In Feb **2017**, researchers from CWI and Google announced the first successful SHA1 collision (https://shattered.io) – see the next 2 slides
- In **2001**, NIST published SHA-224, SHA-256, SHA-384, SHA-512, collectively known as **SHA-2**.
  The number in the name indicates the digest length.
- No known attack on full SHA-2 but there are known attacks on "partial" SHA-2, for e.g. attack on a 41-round SHA-256 (whereas the full SHA-256 takes 64 rounds).
- In Nov **2007**, NIST called for proposal of **SHA-3**.
  In Oct **2012**, NIST announced the winner, Keccak (pronounced "catch-ack").

  (See: NIST announcement http://www.nist.gov/itl/csd/sha-100212.cfm)

Recent Successful Collision Attacks on SHA-1 (Feb 2017)

SHAttered
(shattered.io)
Feb 23, 2017



Secure | https://security.googleblog.com/2017/02/announcing-first-s

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Announcing the first SHA1 collision

February 23, 2017

Posted by Marc Stevens (CWI Amsterdam), Elie Bursztein (Google), Pierre Karpman (CWI Amsterdam), Ange Albertini (Google), Yarik Markov (Google), Alex Petit Bianco (Google), Clement Baisse (Google)

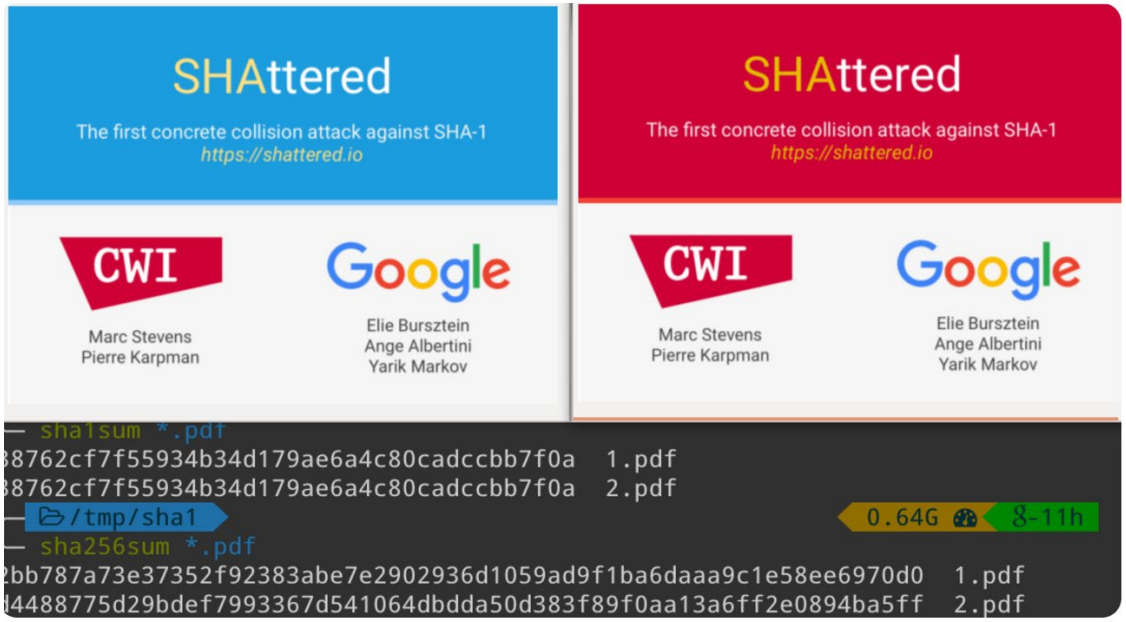
Cryptographic hash functions like SHA-1 are a cryptographer's swiss army knife. You'll find that hashes play a role in browser security, managing code repositories, or even just detecting duplicate files in storage. Hash functions compress large amounts of data into a small message digest. As a cryptographic requirement for wide-spread use, finding two messages that lead to the same digest should be computationally infeasible. Over time however, this requirement can fail due to attacks on the mathematical underpinnings of hash functions or to increases in computational power.

Today, more than 20 years after of SHA-1 was first introduced, we are announcing the first practical technique for generating a collision. This represents the culmination of two years of research that sprung from a collaboration between the CWI Institute in Amsterdam and Google. We've summarized how we went about generating a collision below. As a proof of the attack, we are releasing two PDFs that have identical SHA-1

Recent Successful Collision Attacks on SHA-1 (Feb 2017)

- Done by a team from CWI and Google
- Two PDF files with the same hash values as attack proof:
  - https://shattered.io/static/shattered-1.pdf
  - https://shattered.io/static/shattered-2.pdf
- Defense mechanisms:
  - **Use SHA-256 or SHA-3 as replacement**
  - Visit shattered.io to test your PDF

Read
https://shattered.io/



Other Popular (but Obsolete) Hash

- MD5

Some historical notes:

- Designed by Rivest, who also invented MD, MD2, MD3, MD4, MD6.
- MD6 was submitted to NIST SHA-3 competition, but did not advance to the second round of the competition.
- MD5 was widely used. It produces 128-bit digest.
- In **1996**, Dobbertin announced a collision of the compress function of MD5.
- In **2004**, collision was announced by Xiaoyu Wang et al.
  The attack was reported to take one hour.
- In **2006**, Klima give an algorithm that can find collision within one minute on a single notebook!

Security implication: ***Do not*** use MD5!

Popular Keyed-Hash (MAC)

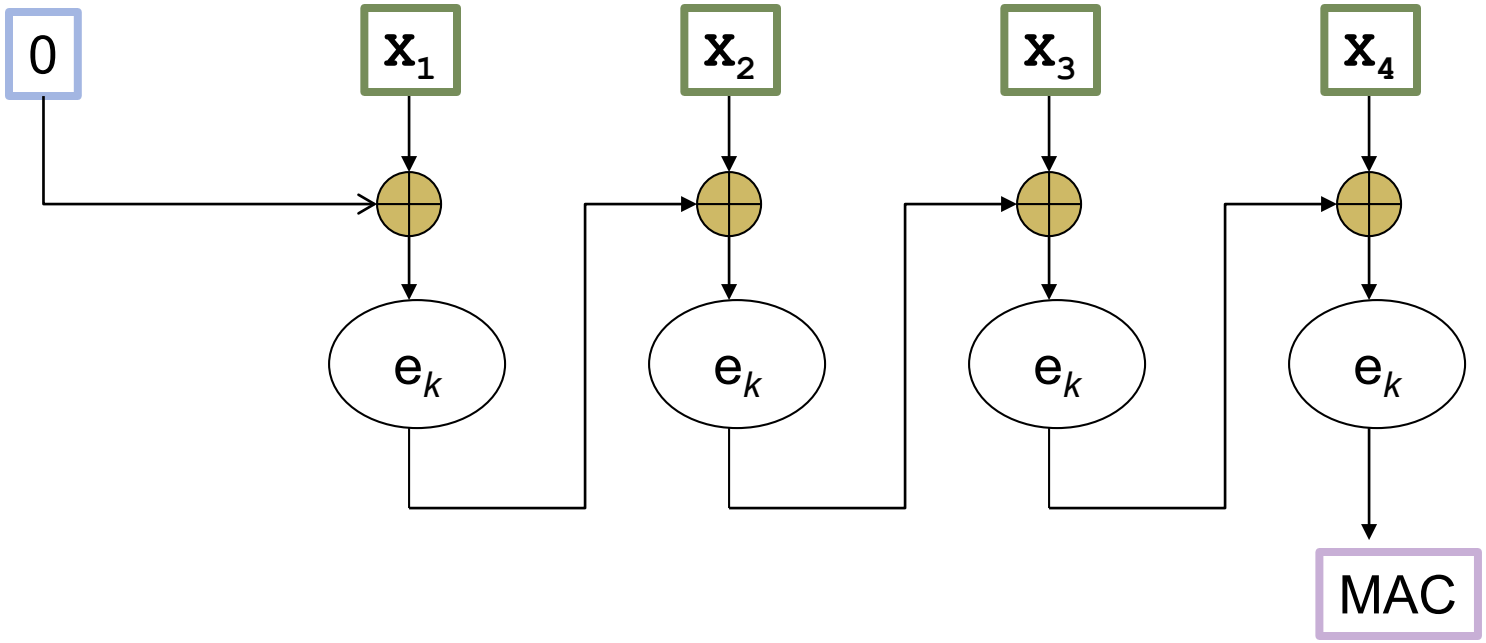
- **CBC-MAC**:
  - Based on AES operated under CBC mode

- **HMAC**:
  - Based on any iterative cryptographic hash function (e.g. SHA)
  - Hashed-based MAC
  - Standardized under RFC 2104 (http://tools.ietf.org/html/rfc2104)

CBC-MAC

Optional

Initial Value (IV)



HMAC

Optional

$HMAC_k(x)$ = **SHA-1** (($K \oplus opad$) || **SHA-1** (($K \oplus ipad$) || x))

where:

opad = 3636…36 (outer pad)

ipad = 5c5c…5c (inner pad)

(Note: the values above are in hexadecimal)

3.3. Data Integrity: Hash (Without Secret Keys)

Authenticity and Integrity Problems

Recall the examples given in last lecture:

- Bob submitted a medical certificate to the lecturer, indicating that he was unfit for exam.
  Was the certificate **authentic**: i.e. was it issued by the clinic?
  Or had Bob altered the date?
- Alice received an email from the lecturer.
  Is the email **authentic**: i.e. sent by the lecturer?

Another **important example**:

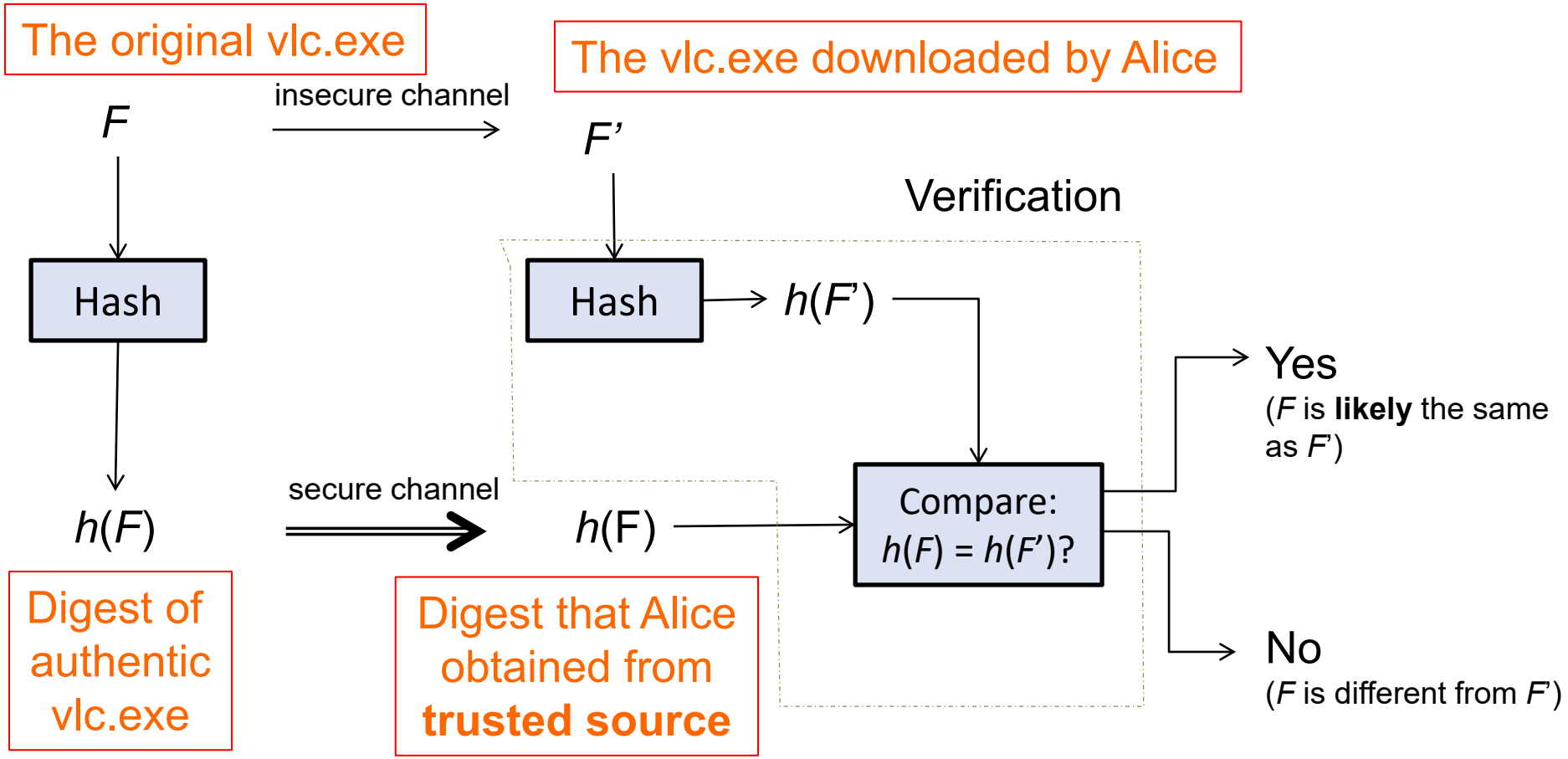
- Alice downloaded **a software** vlc-2.15-win32.exe from the Web.
  Is the downloaded file **authentic**?
  (See the "checksum" in http://get.videolan.org/vlc/2.1.5/win32/vlc-2.1.5-win32.exe)

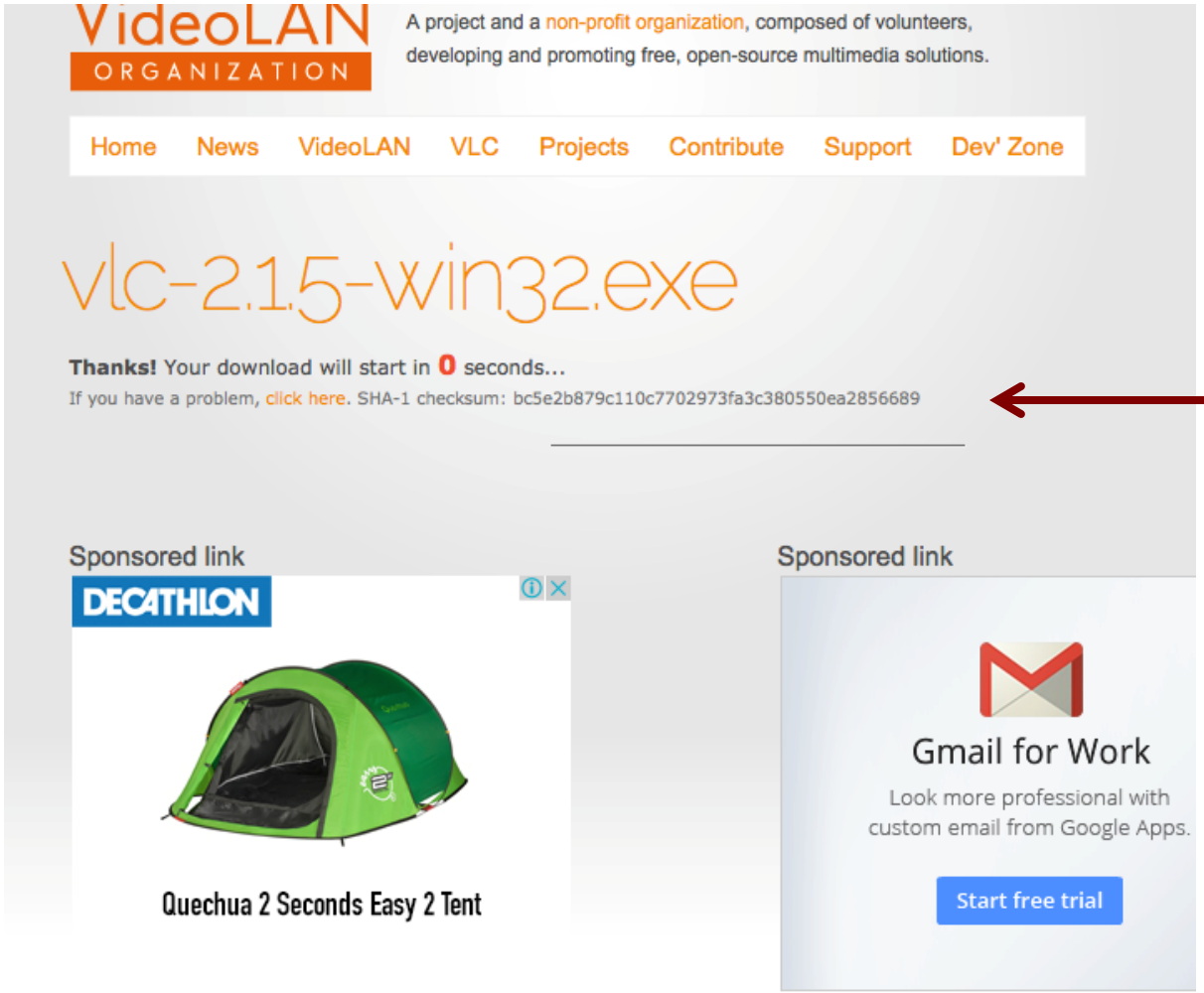
Using (Unkeyed) Hash for *Integrity* Protection

Assuming that **there is a secure channel** to send a short piece of information, the steps are as follows:

- Let F be the *original* data file
- Alice obtains the digest $h(F)$ from the secure channel
- Alice then obtains a file, say F', whose origin claims that the file is F
- Alice computes and compares the two digests $h(F)$ and $h(F')$:
  - If $h(F) = h(F')$, then $F = F'$ (with a very high confidence)
  - If $h(F) \neq h(F')$, then $F \neq F'$, i.e. the file integrity is compromised

Using (Unkeyed) Hash for *Integrity* Protection





The digest (checksum)

Some Remarks

- We may argue that with the digest,
  the verifier can be assured that the data is "**authentic**",
  and thus the "authenticity" of the data origin is achieved
- Nevertheless, in many literatures/documents,
  when there is no secret key involved,
  we refer to the problem as an "**integrity**" issue
- The requirement of *a secure channel*,
  e.g. digest posted on a webpage which is sent over HTTPS
- How can provide (data-origin) authenticity?
  Use **MAC** or **digital signature**

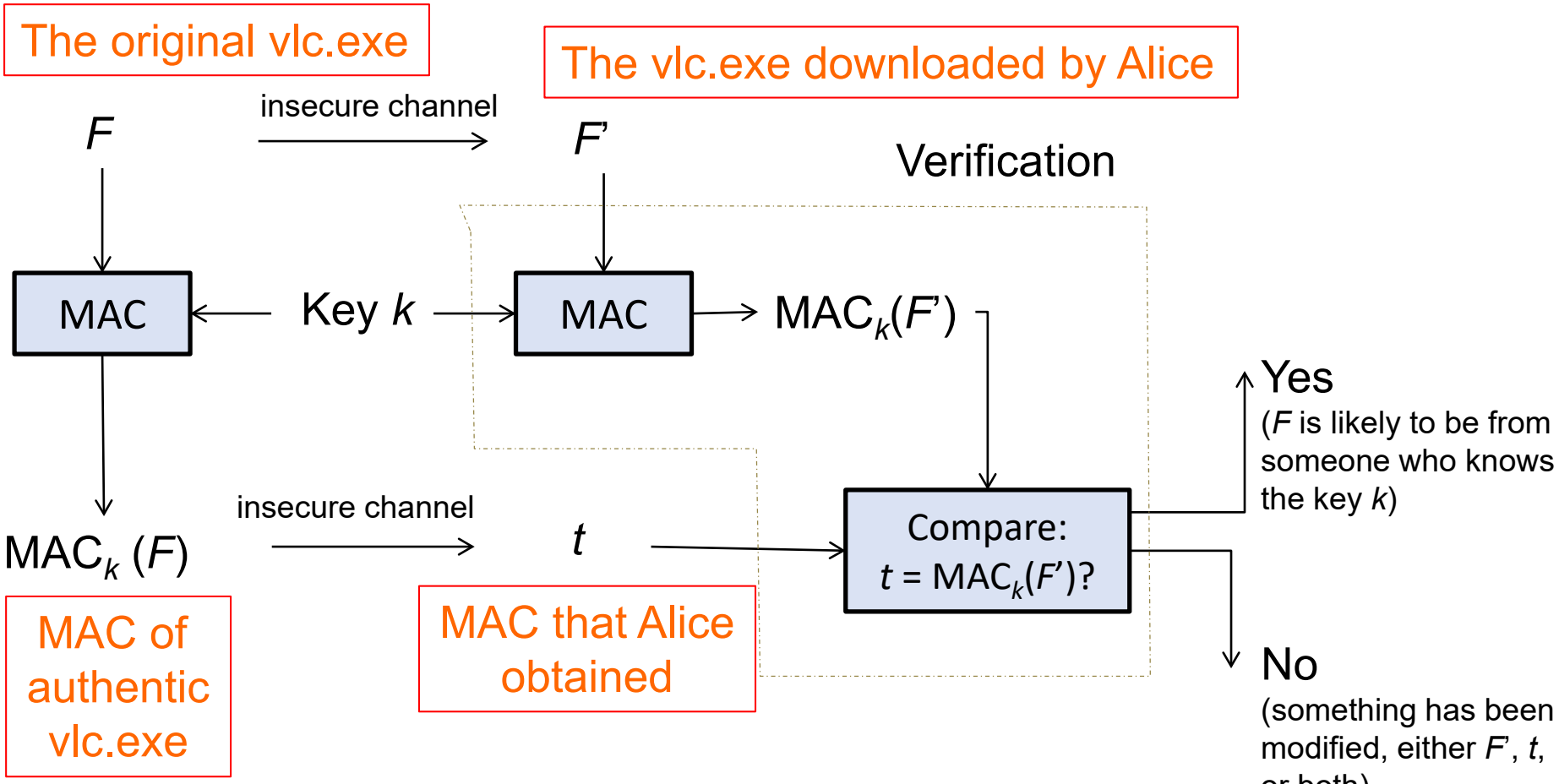
3.4. Data-Origin Authenticity: MAC and Signature

MAC and Signature

- In the previous vlc.exe example, the digest **could be faked**:
  - E.g. the server hosting the web page could be compromised
  - Or in the first place, it was obtained from a “spoofed” webpage
- In other words, we *don’t have* a secure channel to deliver the digest
- In such scenarios, we can protect the digest with the help of some secrets:
  - In the symmetric-key setting: **MAC**
  - In the public-key setting: **digital signature**

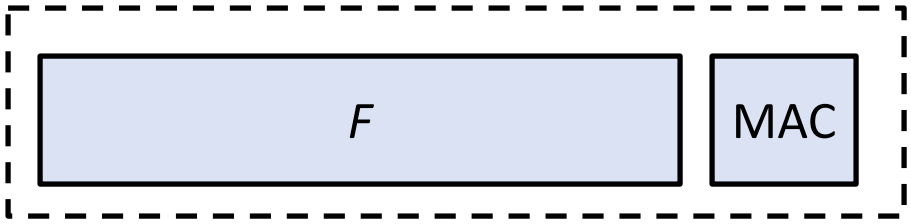
MAC (Message Authentication Code)

- In this setting, the MAC might also be modified by attacker
- If such a case happened, it can be detected with a high probability



Some Remarks

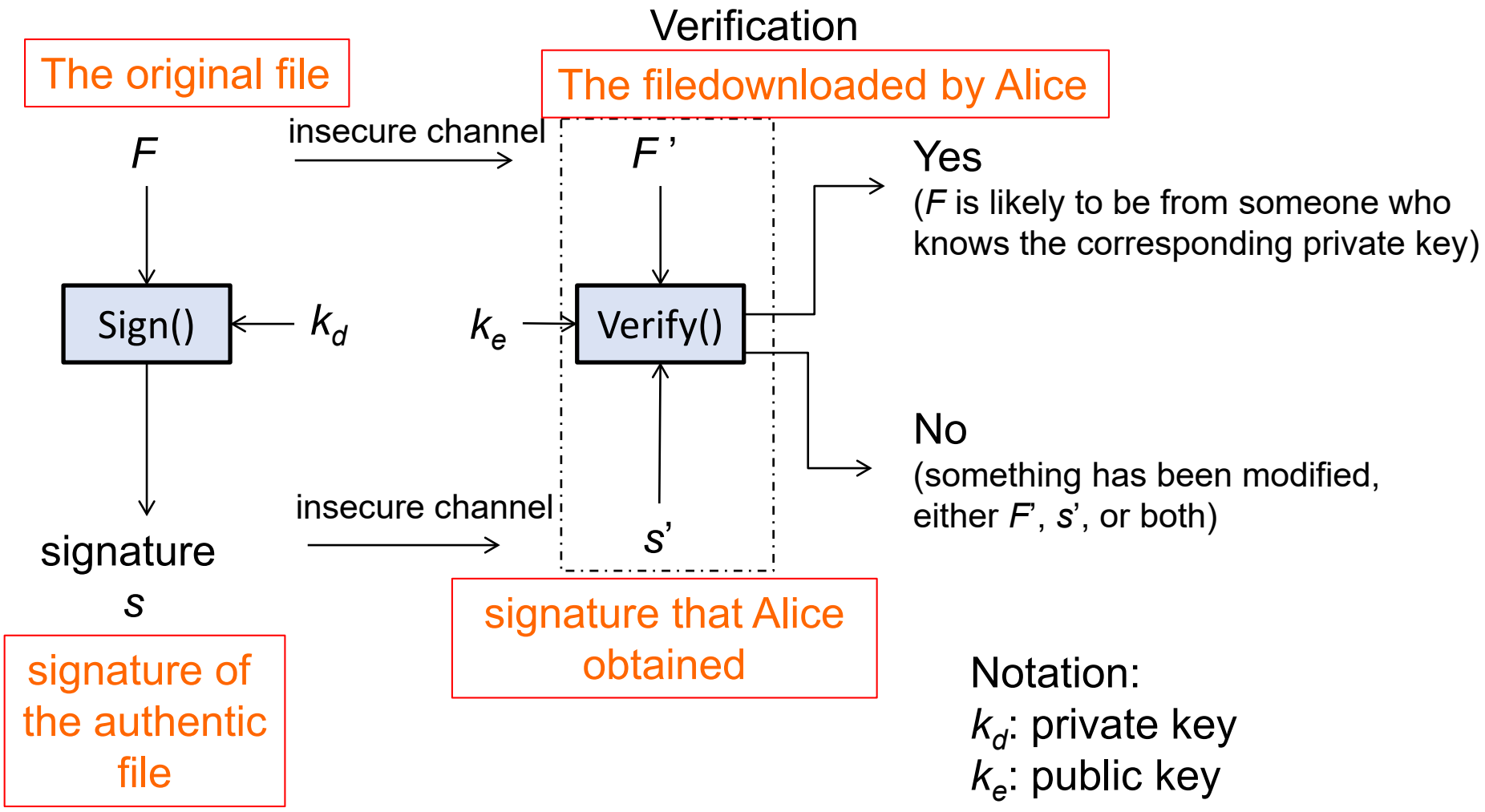
- Note that there is no issue with confidentiality:
  the **data *F*** can be sent **in clear**
- Typically, the *MAC* is **appended** to *F*,
  then they are stored as a single file,
  or transmitted together through a communication channel
  (Hence, MAC is also called the *authentication tag*)



- Later, an entity who wants to verify the *authenticity* of *F*,
  can carry out the verification process using the secret key

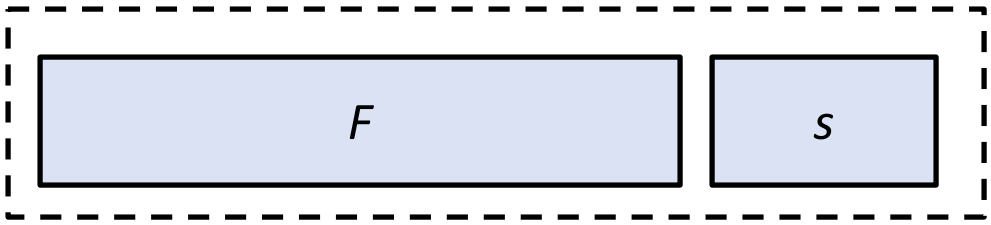
Digital Signature

This is essentially the asymmetric-key version of MAC



Some Remarks

- Note that the signature is computed using the private key and *F*
- Likewise, the computed signature is typically **appended** to *F* and stored as a single file



- When we say that ***"Alice signs the file F"***, we mean that Alice computes the signature *s*, and then appends it to *F*
- Later, the authenticity of *F* can be verified by ***anyone*** who knows the signer's public key
- Why? The valid signature can be computed only by someone knowing the private key:
  if the signature is valid, then *F* must be authentic

What's so Special about Signature (Compared to MAC)?

- We can view the *digital signature* as the counterpart of the *handwritten signature* in a legal document:
  - A legal document is authentic or certified if it has the **correct** handwritten signature
  - **No one**, except the authentic signer, can forge the signature
- In addition to authenticity, signature scheme also achieves ***non-repudiation***:
  - *Assurance that someone cannot deny his/her previous commitments or actions*

Two Scenarios of Non-Repudiation Provisioning

Using only **MAC** (non-repudiation is ***not*** achieved):

- Suppose Alice sent Bob a message appended with a MAC, which is computed with a secret key **shared by** Alice and Bob
- Later Alice denied that she had sent the message
- When confronted by Bob, Alice claimed that Bob generated the MAC

Using **digital signature** (non-repudiation ***is*** achieved):

- Suppose Alice sent Bob a message signed using her private key
- Later, she wanted to deny that she had sent the message
- However, she was unable to so: only the person who knows the **private key** (i.e. Alice) can sign the message
- Hence, the signature is **a proof** that Alice is aware of the sent message

Popular Signature Schemes

- Many signature schemes incorporate both PKC and cryptographic hash in their designs
- The algorithm ***sign()*** usually corresponds to decryption using private key, and the algorithm ***verify()*** corresponds to encryption using public key: the details are omitted here
- However, do refer to the operations as ***sign()*** and ***verify()***

- For examples:
  - RSASSA-PSS, RSASSA-PKCS1:
    a signature scheme based on RSA
  - DSA (Digital Signature Algorithm):
    a standard by FIPS that is based on ElGamal

3.5. Some Attacks and Pitfalls

3.5.1 Birthday Attack on Hash

Hash and Birthday Attack

- Hash functions are designed to make a **collision** difficult to find (i.e. to be **collision resistant**)
- Recall that a *collision* involves two different messages m_1, m_2 that give the same digest, i.e.:

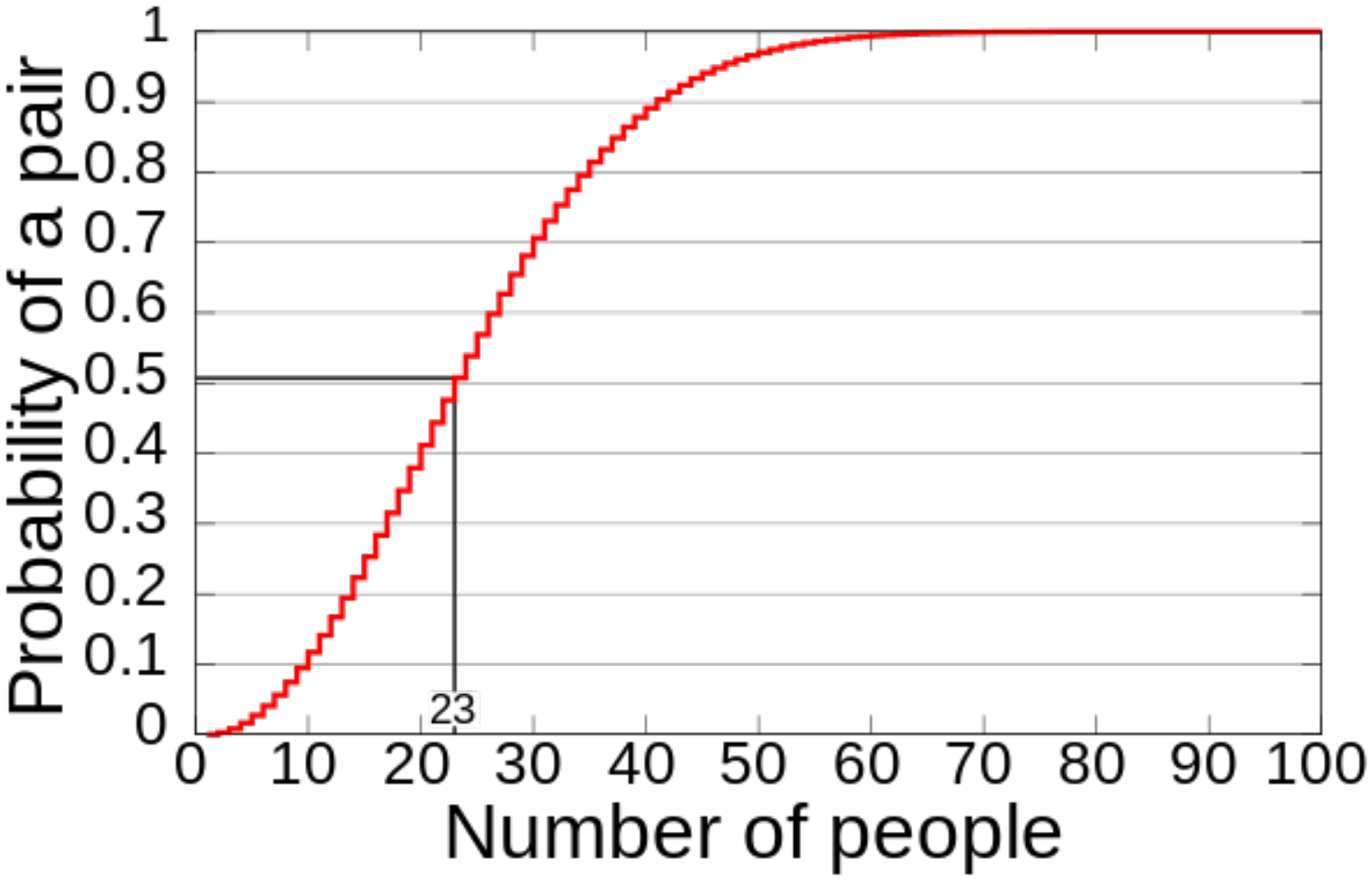
$$h(m_1) = h(m_2) \quad \text{and } m_1 \neq m_2$$

- Nevertheless, all hash functions are subjected to **birthday attack** (similar to exhaustive search on encryption schemes)

Birthday Paradox/Problem

- "How many students need be randomly selected so that, with probability more than 0.5, there is *a pair of students* having the same birthday"
- Answer: **23**
- The (low) number needed may surprise many people!
- See: https://en.wikipedia.org/wiki/Birthday_problem
- *Why is that possible?*
- There are **366** possible birthdays (including 29 February)
- By **the pigeonhole principle**, the probability reaches 100% when the number of students reaches 366+1=367
- This is *not* about fixing on one individual and comparing his/her birthday to everyone else's birthday
- But about comparing between *every possible pair* of students = 23×22/2 = 253 comparisons

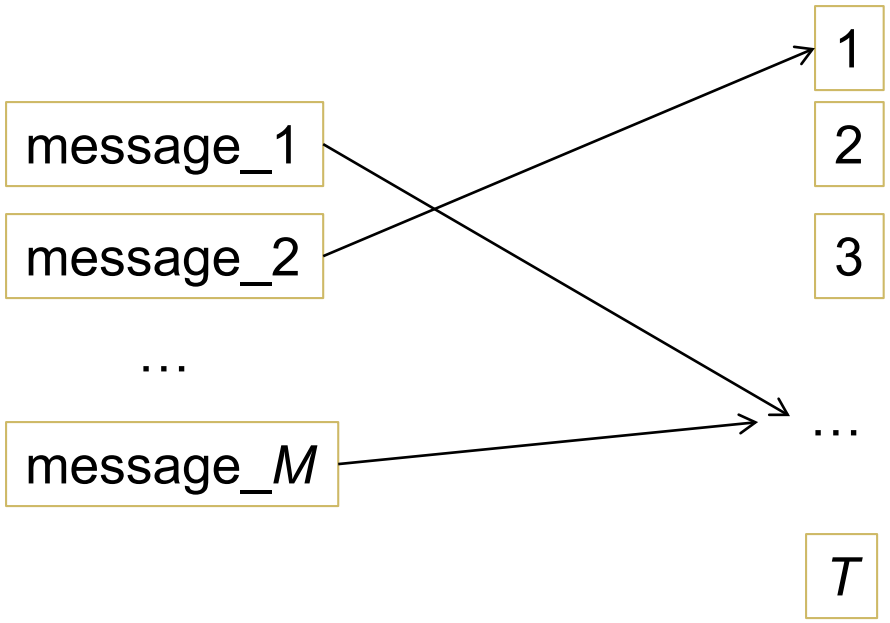
Birthday Paradox/Problem



Ref: https://en.wikipedia.org/wiki/Birthday_problem

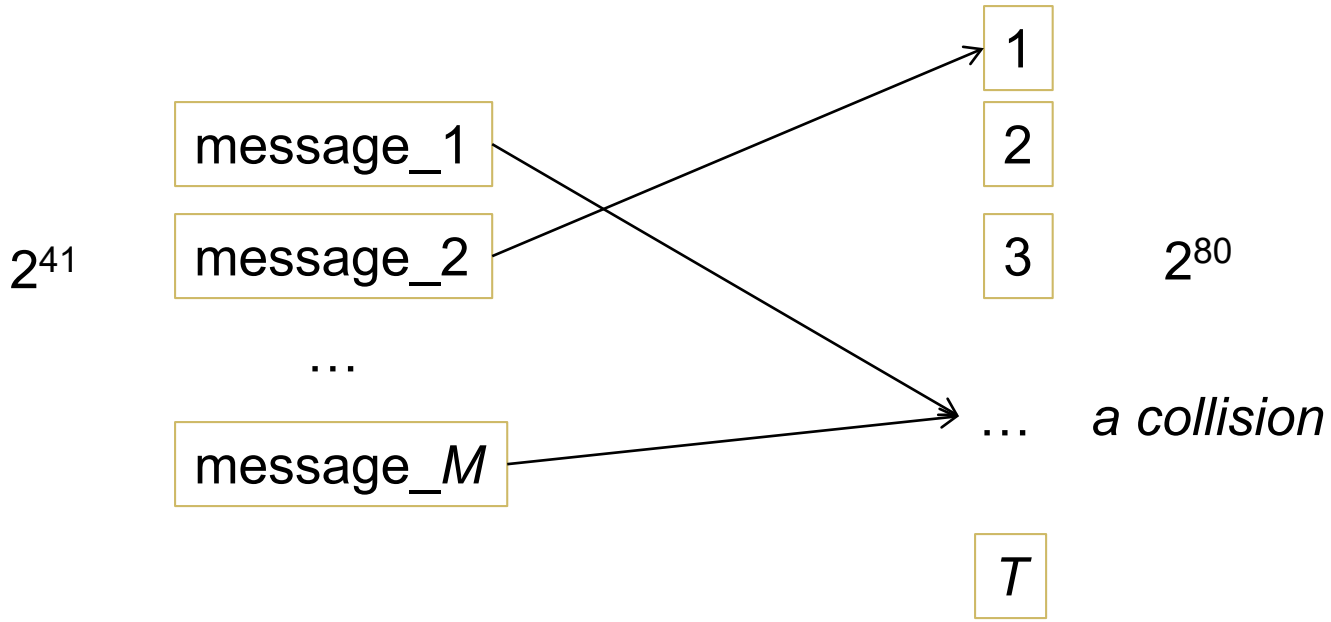
Birthday Paradox/Problem

- Birthday paradox can be applied to hash function
- Suppose we have *M* messages, and each message is tagged with a value randomly chosen from {1,2,3,…,*T*}
- If $M > 1.17\ T^{0.5}$, then with probability more than 0.5, there is *a pair of messages* tagged with the same value
- This result leads to **birthday attack** on a hash function: to reduce the complexity/work-factor of finding a collision



Birthday Attack on Hash

- Suppose the digest of a hash is 80 bits: $T = 2^{80}$
- Now, an attacker wants to find a collision
- If the attacker randomly generates $\mathbf{2^{41}}$ messages ($M = 2^{41} = 2^{1 + 80*0.5}$), then $M > 1.17\ T^{0.5}$
- Hence, with probability more than 0.5, among the $\mathbf{2^{41}}$ messages, ***two of them*** *give the same digest*!



In general, the **probability that a collision occurs** ≈ 1 - exp ($-M^2/(2T)$)

Implication of Birthday Attack for Digest Length

- Birthday attack has a serious consequence on the **digest length** required by a hash function to be collision resistant
- Read the NIST cryptographic key-length recommendation, including on several popular hash functions: http://www.keylength.com/en/4/
- When the key length for a symmetric-key is **112**, the corresponding recommended length for digest is at least **224**
- Why must the digest length be significantly larger? (Answer: birthday attack, of course)

3.5.2 Design Flaw: Using Encryption for Authenticity

Misuse of Encryption: A Case Study

- Encryption schemes may provide *false sense* of security
- Consider the following case involving **the design of a mobile app** from company XYZ:
- The mobile phone and a server share a secret 256-bit key *k*. The server can send **instructions** to the mobile phone via SMS. *(Note that an SMS consists of only readable ASCII characters. We assume, however, that there is a way to **encode** a binary string using the readable characters).*
- Suppose **the format** of the instruction is:

  X P

  where: *X* is a 8-bits string specifying the type of operation, and *P* is a 248-bits string specifying the parameter.
- If an operation doesn't take in parameter, *P* will be ignored

Misuse of Encryption: A Case Study

- There is a total of **16 valid instructions**, such as:

  00000000 *P* : adds *P* into the contact list

  11110000 *P* : rings for *P* seconds

  10101010 : self-destruct (brick) the phone!

- An instruction is to be **encrypted** using as 256-bit AES, encoded to readable characters, and then sent as an SMS
- After a mobile phone received the SMS, it **decrypts** it. If the instruction is invalid, it ignores the instruction. Otherwise, it executes the instruction.
- The company XYZ claims that: "*256-bits AES provides high level of security, and in fact is classified as Type 1 by NSA. Hence the communication is secure. Even if the attackers have compromised the base station, they are still unable to break the security*".
- **What's wrong with this?**
  And what cryptographic technique should be used instead?

Misuse of Encryption: A Case Study (Concluding Remarks)

- Encryption is designed to provide **confidentiality**
- It does **not**, however, guarantee integrity and authenticity
- In the case above, the XYZ company actually wants to achieve “authenticity” (or “integrity”??)
- But it wrongly employs encryption to achieve that
- A secure design could use **MAC** instead of encryption

Notes:

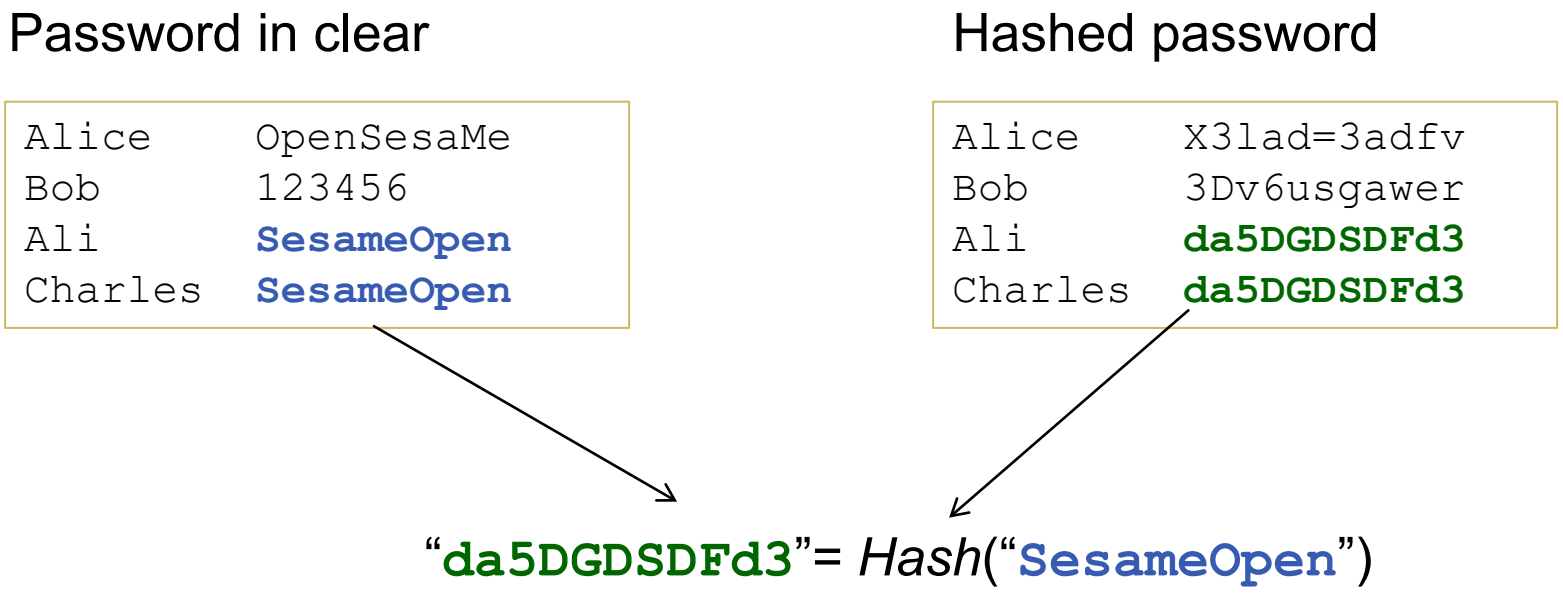
- There are still some details being omitted in this case
- Simply adding MAC to the instructions is not secure, for example against “replay attacks”
- To prevent replay attacks, a “nonce” is required

3.6 Password File Protection (Revisited)

Hashed Password (From Lecture 2)

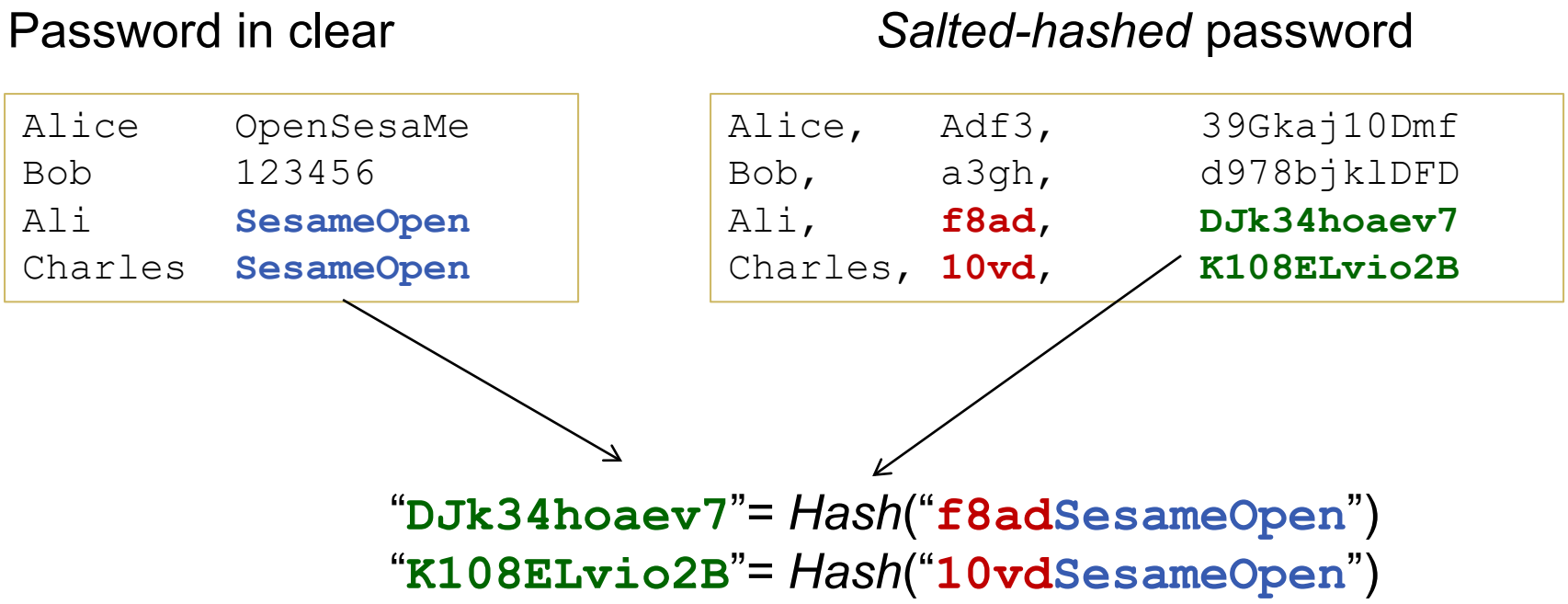
- Passwords should be "hashed" and stored in the password files.
  (Textbook ([PF]pg 46) uses the term "encrypted". Note that this is inaccurate. For encryption, there is a way to recover the password from the ciphertext. For cryptographically secure hash, it is infeasible to recover the password from its hashed value.)

- During authentication, the password entered by the entity is hashed, and compared with the the value stored in the password file



Hashed Password (From Lecture 2)

- It is desired that the same password will be hashed into *two different values* for two different userid.
  Why? (see tutorial)
- This can be achieved by using ***salt***



Interesting Sample Question:

- Back to the example given on Slide 17
  (the attack on a penguin-image encryption)
- One remedy is as follow:
  "**each block** is to be encrypted with an IV"
- Although this is secure, it **increases the ciphertext size**, since the IV for each block has to be included in the ciphertext
- Suppose each IV is 16 bits, there will be an overhead of 16 bits **per block**
- Hence, with the remedy, it is desirable to keep the size of IV small

Interesting Sample Question:

- Now, suppose it has been still decided to use 16 bits for the IV, and each image has around 2^{10} = 1,024 blocks
- Below are two ways of choosing the IVs:
  1. Start from a randomly-chosen number for the first block, and increment it by one for subsequent blocks.
     Hence, the IVs for the 1^{st}, 2^{nd}, 3^{rd}, .. block are:
     r, $(r+1) \bmod 2^{10}$, $(r+2) \bmod 2^{10}$, …
     where r is a randomly chosen value.
  2. Independently and randomly choose an IV for each block
- Which one would you prefer?
- Would you still prefer AES working in Cipher Block Chaining (CBC) mode?