Lecture 3: Authenticity (Data Origin: mac & signature)

3.1. Crypto Primitive: Public Key Cryptography

- 3.1.1 RSA
- 3.1.2 Security of RSA
- 3.1.3 Remarks

3.2 Crypto Primitive: Cryptographic Hash

- 3.2.1 Hash
- 3.2.2 keyed-hash

3.3 Data Authenticity (Hash)

3.4 Data Authenticity (Mac)

3.5 Data Authenticity (Signature)

3.6 Some attacks & pitfalls

- 3.6.1 Birthday attack on hash
- 3.6.2 Design flaw: using encryption for authenticity
- 3.6.3 Time-Space tradeoff

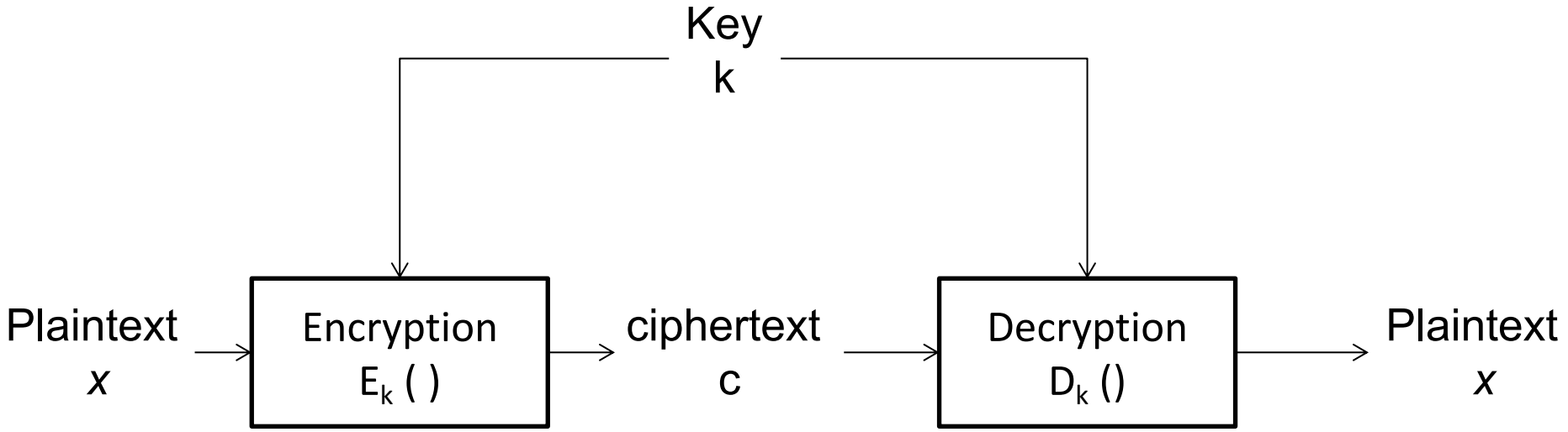
3.7 Application of Hash: password file protection (in Lecture 2)

3.1. Crypto Primitive: PKC

We have seen a crypto primitive: symmetric key encryption. Now, we are going to introduce a few more. *PKC, Hash, Key-ed Hash (mac), Signature.*

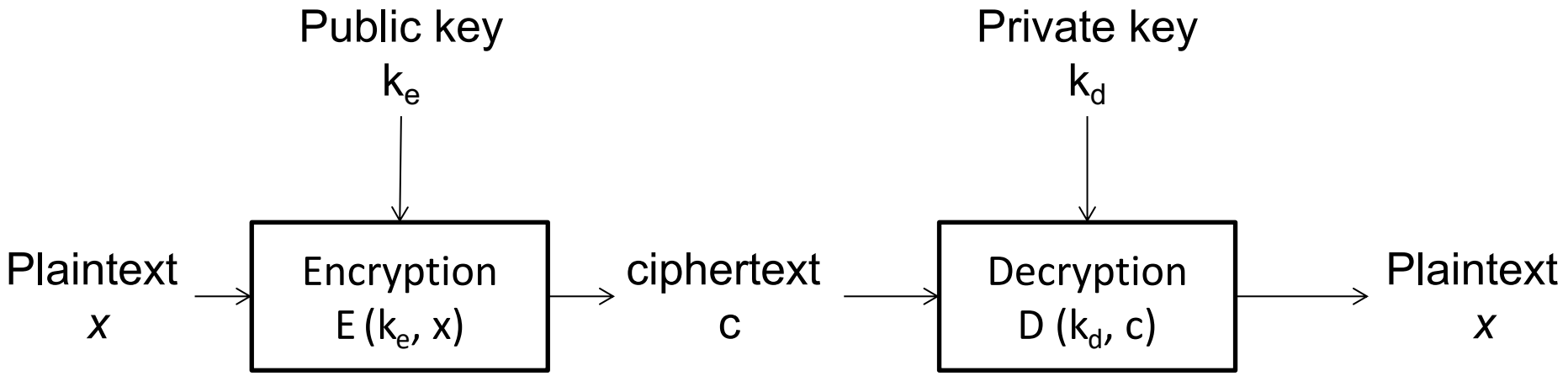
Overview

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



Overview

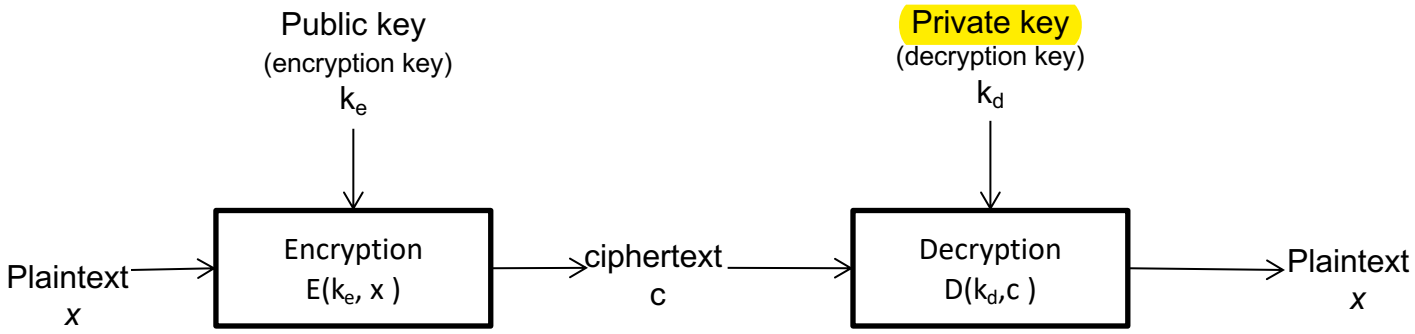
A public key (aka asymmetric-key) scheme uses different keys for encryption and decryption.



Note:

- The private key is typically of the form: $k_d = \langle k_1, k_2 \rangle$ where k_1 is exactly the public key. That is, part of the private key is the public key.
- Some books use this definition: They call k_2 the private key, k_1 the public key and define the input of decryption as public key + private key. Just a different way to define the same the concept.
- We adopt the version as defined above so that the notation $D(k_d, c)$ makes sense. If not, we must write it as $D(\langle k_d, k_e \rangle, c)$ which is too "long-winded".

Why it is called "public key"?



In a typical usage, the owner Alice keeps the private key as a secret but tell everyone the public key (for eg. posts in her Facebook).

An entity, Bob, has a ***plaintext x*** for Alice. Bob can encrypt it (using the ***public key***) and posts the ***ciphertext c*** on Alice's Facebook.

Another entity, Eve, obtains the public key, and ciphertext c from the Alice's Facebook. Without the private key, Eve is unable to derive x.

Alice, with the ***private key***, can decrypt and obtain the plaintext x.

Security Requirements of a Public Key Scheme

- *security requirement*:

  - Given the public key and the ciphertext (but not the private key), it is difficult to determine the plaintext;

- The requirement implies that it must be difficult to get the private key from the public key.

this is an implied requirement

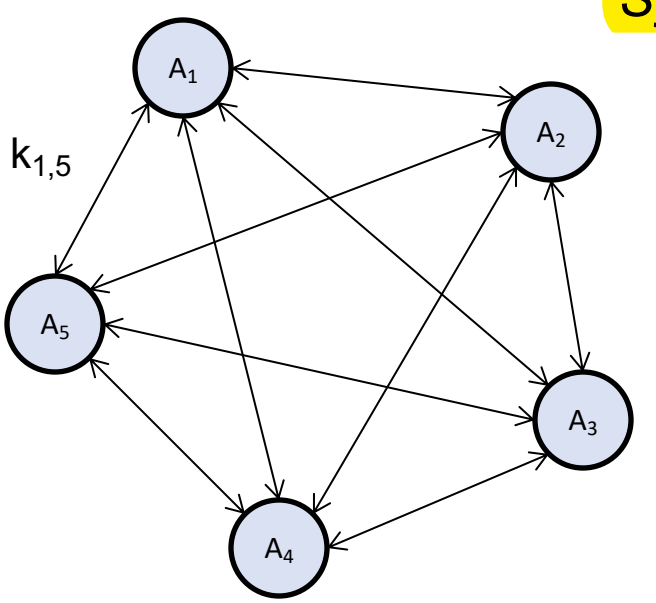
note: A refined notion of "*difficult to determine the plaintext*" is the indistinguishability of the ciphertext from a truly random source, i.e. to an entity who doesn't know the private key, the ciphertext resembles a sequence of random values. Notion of indistinguishability would be covered in CS4236 (crypto). Also note that because everyone (including attacker) can encrypt, in PKC, chosen plaintext attack is a must to be considered.

Advantages in Key Management

- 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 use 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 number of entities is large.
  - Not feasible in scenarios where the two entities haven't met before the session. (e.g. does www.bbc.com know you? Can you establish a symmetric key with BBC before visiting the website?)

Number of keys involved

Symmetric key setting



Every pair of entities requires one key.

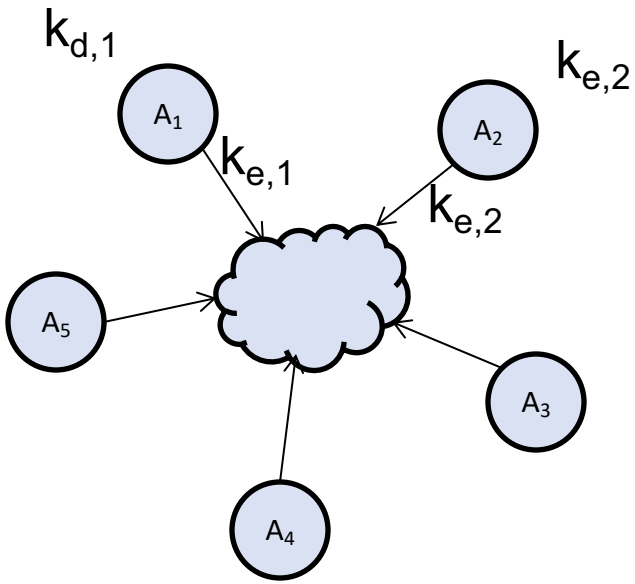
Let $k_{i,j}$ to be the key to be shared by A_i and A_j

$k_{1,2}$, $k_{1,3}$, ...

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

order of magnitude for the number of key is n^2

Public key setting



Each entity publishes its public key

Entity A_i publish $k_{d,i}$ and keep $k_{e,i}$

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

Public keys can be published/broadcasted even before two entities know each other.

Important note

- It is very wrong to say that in PKC, we no longer need a secure channel to distribute the key.

- We still need a secure channel to distribute the public key.

(whole lecture on PKI)

- While we are using encryption to describe PKC, an important application of PKC is for authentication (e.g. HTTPS).

Popular PKC schemes

- RSA Key size ≈ 2048 bits
- ElGamal: ElGamal can exploit techniques in Elliptic Curve Cryptography (ECC), reducing the key size to ≈ 300 bits.
- Paillier: Partial homomorphic with respect to addition.

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

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

Rivest, Shamir and Adleman,
2002 Turing Award.

Integer representations

- We will introduce the "classroom" RSA, i.e. the basic form of RSA. We call it "classroom" because the variant used in practice is different, with padding and special considerations in choosing the primes, etc. Also called "textbook" RSA.

- Many PKC represent the data (plaintext, ciphertext, key) as integers, and the algorithm are some arithmetic operations on the integers. The integers can be represented using binary representations. When we say that a key is of 1024-bit, we mean that the key can be represented using 1024 bits under binary representation. (e.g. a 3-bit integer is a value from 0 to 7)

- Note that the total number of 1024-bit integers are 2^{1024}. So, an algorithm that exhaustively searches 1024-bit numbers is infeasible.

3.1.1 RSA

Classroom/Textbook RSA: algo that taught in many “non-security” modules.

In practice: Padded RSA with strong primes. In some cases, secure implementation to guard against side-channel attack.

"Classroom RSA" - setup

1. Owner randomly chooses 2 large primes ***p, q*** and computes ***n*** = ***pq***.
2. Owner randomly chooses an encryption exponent ***e*** s.t. gcd(***e, n***) =1.
   (i.e. e<n, and e is not a multiple of p or q)
3. Owner finds the decryption exponent ***d***

   where ***d*** **e** mod $(p-1)(q-1)$ =1

   There is an algorithm that finds d, when given e, p and q. We won't get into the details,

   The term $(\boldsymbol{p}-1)(\boldsymbol{q}-1) = \Phi(\boldsymbol{n})$ is aka the Euler's totient function, which is the number of co-primes < n
4. Owner publishes (***n***, ***e***) as public key, and safe-keep (***n***, ***d***) as the private key. (note that owner doesn't need to keep p, q)

Encryption, Decryption

public key (n, e) private key (n, d)

- **Encryption,**
  Given *m*, the ciphertext *c* is

  $c = m^e \bmod n$

- **Decryption**
  Given *c*, the plaintext *c* is

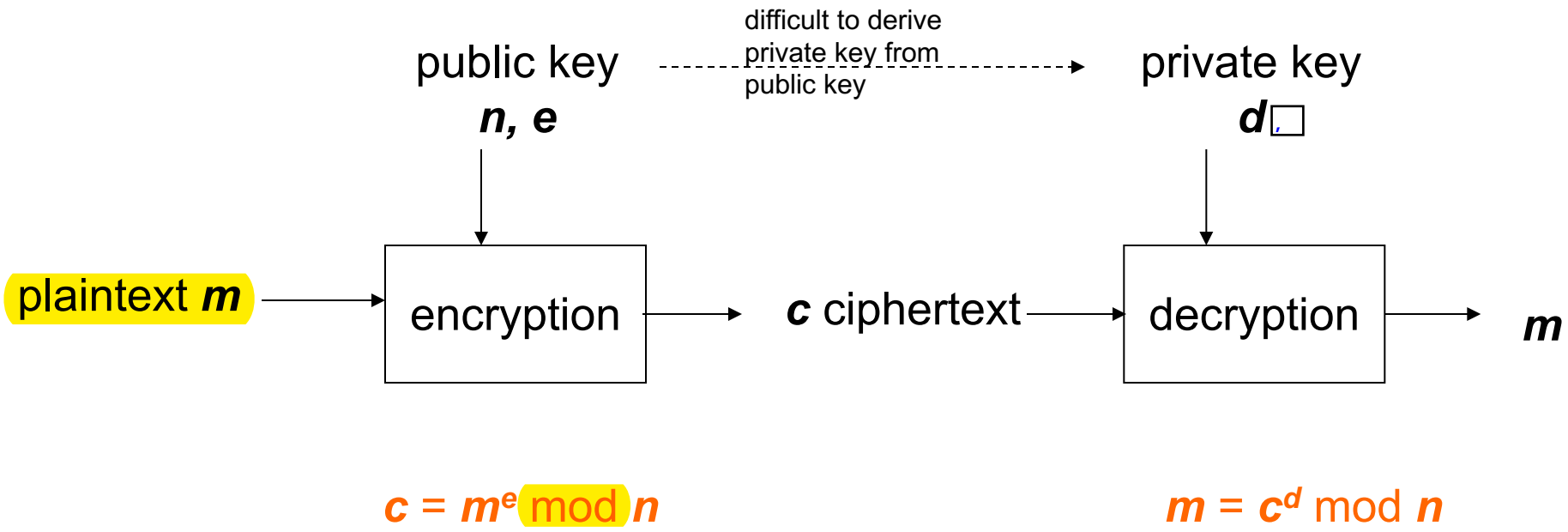
  $m = c^d \bmod n$

RSA

$n = pq$

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

$d = e^{-1} \bmod \Phi(n)$



Optional remark: decryption amounts to, given c, n, e, finding the e-th root of c.
Compare to Discrete log problem: given c, n, m, finding e, the discrete log of m.

Correctness of RSA

- Note that for any positive m <n, and any pair of public/private keys,

Decrypt (Encrypt (m)) = m

That is,

$$(m^e)^d \mod n = m$$

Optional

Proof (sketch): The correctness depends on this property of modulo:

For any m, r, n, we have

$$m^r \mod n = m^{r \bmod \Phi(n)} \mod n$$

(when n is product of two primes, p,q, then $\Phi(n) = (p-1)(q-1)$.)

Now, combining the fact that

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

we have $m^{de} \mod n = m^1 \mod n = m$

Example

- p=5, q=11, n=55
- (p-1)(q-1) = 40
- Suppose e = 3, then d = 27

There is an efficient algorithm
that computes d from p, q, e. Details omitted.

note that indeed $3 \times 27 \bmod 40 = 81 \bmod 40 = 1$

- Suppose m = 9

Encrypt:

$$c = m^e \bmod n = 9^3 \bmod 55 = 14$$

Decrypt

$$c^d \bmod n = 14^{27} \bmod 55$$

$$= 14^{8\times 2 + 8 + 3} \bmod 55$$

$$= (36 \times 16 \times 49) \bmod 55$$

$$= 9$$

There is an efficient algorithm
that computes modulo exponentiation.
Details omitted.

Interchangeable role of encryption and decryption key

- Classroom RSA has this property: we can also use the decryption key ***d*** to encrypt, and then the encryption key ***e*** to decrypt.

- The above property is special in RSA. It usually does not hold in other public key schemes. For e.g. the ElGamal PKC (not covered in this module) doesn't has this property.

Some documents flip the role of encryption/decryption when describing RSA, i.e. using public key to decrypt. This could be very confusing. Take special note.

Algorithmic issues

- (Encryption/decryption) Given n, m, e, there is an efficient algo to compute exponentiation: $m^e \bmod n$. Likewise, there is an efficient algo to compute $c^d \bmod n$

  (in practice, e is intentionally chosen to be small so that encryption can be done very fast.)

- (step 1 in setup: Primality test) How to find a random prime? Randomly pick a number, and test whether it is a prime. Since there are many primes, there is a high chance that a randomly chosen number is prime. (as we shall see later, some primes are weak and lead to attack. Using "strong prime" to avoid those attacks)

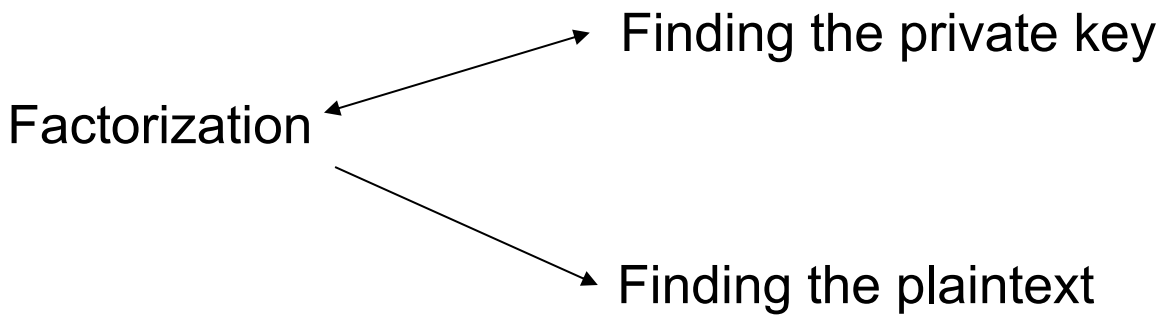
  (step 3 in setup) The value of d can be efficiently computed from e and n using the *extended Euclidean algorithm.*

3.1.2 Security of RSA

Security of RSA

- It is can shown that, the problem of getting the RSA private key from public key is as difficult as the problem of factorizing n.

- However, it not known whether the problem of getting the plaintext from the ciphertext is as difficult as factorization, that is, it could be easier.



State-of-the-art on factorization

- more info: http://en.wikipedia.org/wiki/Integer_factorization
- A 640 bits number was successfully factored in Nov 2, 2005, using approximately 30 2.2GHz-Opteron-CPU years. It is computed over five months using multiple machines.
- A 768 bits number (RSA-768) was factored in Dec 2009, using hundreds of machines over 2 years.
- See NIST recommendation.

- Here is the 640-bits number

  3107418240490043721350750035888567930037346022842727545720161948823206440518081504556346829671723286782437916272838033415471073108501919548529007337724822783525742386454014691736602477652346609

  =

  1634733645809253848443133883865090859841783670033092312181110852389333100104508151212118167511579

  *

  1900871281664822113126851573935413975471896789968515493666638539088027103802104498957191261465571

- Here is the 768-bits number

123018668453011775513049495838496272077285356959533479219732245
215172640050726365751874520219978646938995647494277406384592519
255732630345373154826850791702612214291346167042921431160222124
047927473779408066535141959745985690214341 3

=

347807169895689878604416984821269081770479498371376856891243138
898288379387800228761471165253174308773781446799948 9

*

367460436667995904282446337996279526322791581643430876426760322
838157396665112792333734171433968102700927987363089 17

from

T Kleinjung et al., *Factorization* of a *768-bit* RSA modulus, eprint 2010.

https://eprint.iacr.org/2010/006.pdf

post-Quantum cryptography

- A Quantum computer can factorize and perform "*discrete log*" in polynomial time. In 2001, a 7-*qubits* quantum computer was built to factor 15, carried out by IBM using NMR.

  http://domino.watson.ibm.com/comm/pr.nsf/pages/news.20011219_quantum.html

Hence, both RSA and "discrete log based" PKC will be broken with Quantum computer.

Post-Quantum Cryptography: This generally refers to PKC that are secure against quantum computer.

- Lattice-based cryptography: Based on this hard problem-- Given the "basis" of lattice, it is computationally hard to find the shortest (or approx) lattice point. Many proposals, no clear winner yet.
- Multivariate polynomial: Given a multivariate polynomial, it is difficult to find the solution (under modulo p). No secure construction yet.

Padding of RSA

- Same as symmetric-key encryption, some forms of IV is required so that encryption of the ***same*** plaintext at different time would give different ciphertext. Hence additional mechanisms are added to the "classroom" RSA.

- Classroom RSA has some interesting properties, e.g. "homomorphic property". These properties are useful in applications (e.g. blind signature, encrypted domain processing), but they lead to attacks and information leakages. Such properties can be destroyed using padding.

- The standard (Public-Key Cryptography Standards) PKCS#1, add "optimal padding" to achieve the above
  http://en.wikipedia.org/wiki/PKCS_1

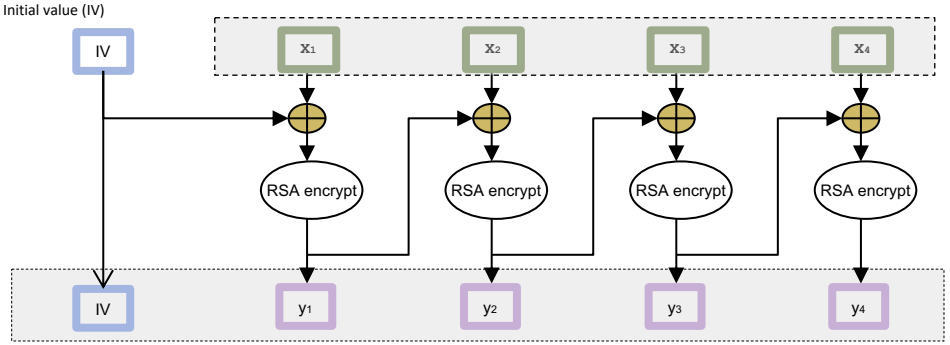
3.1.3. Remarks

Pitfall: using RSA in the symmetric key setting.

It is quite common to see in final year projects where the students use classroom RSA instead of AES, although the applications only require symmetric key. Do not do that, for reason in efficiency and security.

E.g.

- *Bob needs to write an application that provides end-to-end encryption between a digital camera and a mobile phone*
- *Bob feels that RSA is cool. Instead of employing AES, Bob employs classroom RSA as the encryption scheme. The pair of public/private key 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.*
- *To encrypt a large image (say 1MB), Bob divides the file into blocks of 256 bytes, and apply RSA on each block using CBC mode.*



Issue 1 (Efficiency)

- Following NIST recommendation, 128-bit AES and 3072-bit RSA has the equivalent key strength. Larger key might lead to difficulty in key management.
- RSA encryption/decryption is significantly slower than AES.

- See Crypto++ Benchmarks (crypto++ is an C++ library)

https://www.cryptopp.com/benchmarks.html

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

Issue 2 (Security of RSA)

RSA is not necessary "more secure" than AES. As mentioned earlier:

- One common argument that RSA is better than AES is: *RSA is "provably" secure*. While it can be proven that, getting ***the private key from the public key*** is as difficult as factorization, but it is still not known whether the problem of getting ***the plaintext from the ciphertext and public key*** is as difficult as factorization (this is known as the RSA problem). So, in a certain sense, similar to AES, there is no rigorous proof that RSA is "secure" in protecting the ciphertext.

- The "classroom" RSA needs to be modified to prevent some attacks. Many attacks exploit one nice property of RSA: ***homomorphic property***. This property is a double-edged sword: it is useful (e.g. blind signature), but is also a vulnerability.

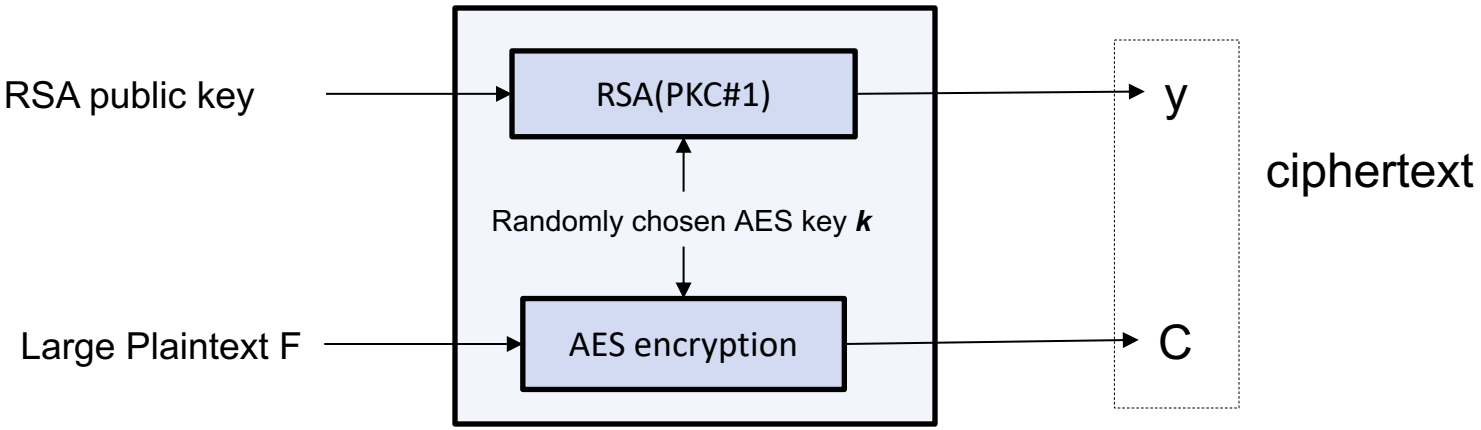
(optional. Warning: heavy math) It turns out that the classroom RSA leaks one-bit of information about the plaintext! If adversary knows the module n, and knows the ciphertext c, the adversary can derive one "bit" of information regarding the plaintext. This is because RSA encryption preserves the "Jacobi symbol", i.e. the Jacobi symbol of the plaintext and ciphertext is the same.

- Factorization can be efficiently done by Quantum Computer. So, RSA is broken under Quantum Computer. It is not clear how Quantum Computer can be used to break AES. (Quantum computer can speed up exhaustive search by square root, that is, to exhaustively search 128-bit keys, it only takes 2^{64} under quantum computer. So, to be secure against quantum computer, one can adopt 256-bit AES (exhaustive search would be 2^{128}). This is not true for RSA under quantum computer. Double the length of key doesn't help.)

Using PKC for encryption of large plaintext

- The main strength of RSA and PKC is the "public key" setting, which allows a public entity to encrypt without knowing the secret key. This is very useful in ***authentication.***

- If public key encryption is required to encrypt a large file F, it can be efficiently carried out in this way: choose a random AES 128-bit key ***k***, encrypt ***k*** using PKC, and encrypt F using AES with ***k*** as the key.



Other PKC

- ElGamal Encryption

ElGamal is a “Discrete Log-based” encryption, whereas RSA is “factorization-based”.

There are many choices of Algebraic groups for discrete log-based encryption, e.g. Elliptic Curve. Those using Elliptic Cure are often called Elliptic Curve Cryptography (ECC). Certain choices of ECC reduce the key size. E.g. ~300 bit for equivalent of 2048-bit RSA.

- Paillier Encryption

Paillier Encryption is also discrete-log based.

ElGamal can be easily modified so that it is homomorphic w.r.t. multiplication, whereas Paillier is homomorphic w.r.t. addition.

- If an algorithm can efficiently solve the Discrete Log problem, then the algorithm can easily break the discrete-log based encryption. (In this module, we omit detail of the discrete log problem.)

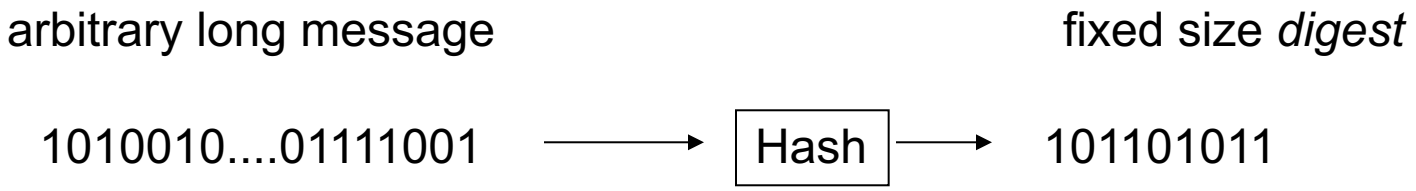
3.2. Crypto Primitive: Hash and keyed-Hash

3.2.1 (unkeyed) Hash

Remarks: We follow Kerckhoff's principle.
Adversary know all the algorithms.

Hash (no secret involved!!!)

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



Security requirement (collision):

- It is difficult for an attacker to find two different messages m_1 , m_2 that "hash" to the same digest. That is,

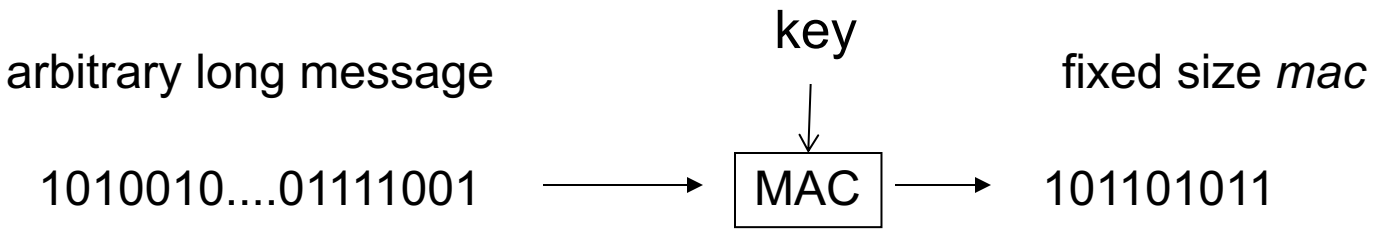
$$h(m_1) = h(m_2)$$

This is known as ***collision-resistant***.

- A hash that is collision-resistant is also ***one-way***, that is, given a digest d, it is difficult to find a message m s.t. h (m)=d.

keyed-Hash (aka MAC) (a secret key is involved)

A keyed-hash is a function that takes an arbitrary large message and a secret key as input, and outputs a fixed size (say 160 bits) ***mac (message authentication code)***.



- **Security requirement (forgery)**: After seen multiple valid pairs of messages and their corresponding mac, it is difficult for the attacker to forge the mac of a message not seen before. (The precise formulation is quite involved, and with many variants. Higher level crypto module CS4236 would cover this)

Example of hash algorithms that are not secure (collision resistant)

- Taking selected bits from the data.
- CRC checksum.

See https://en.wikipedia.org/wiki/Cyclic_redundancy_check

Popular Hash

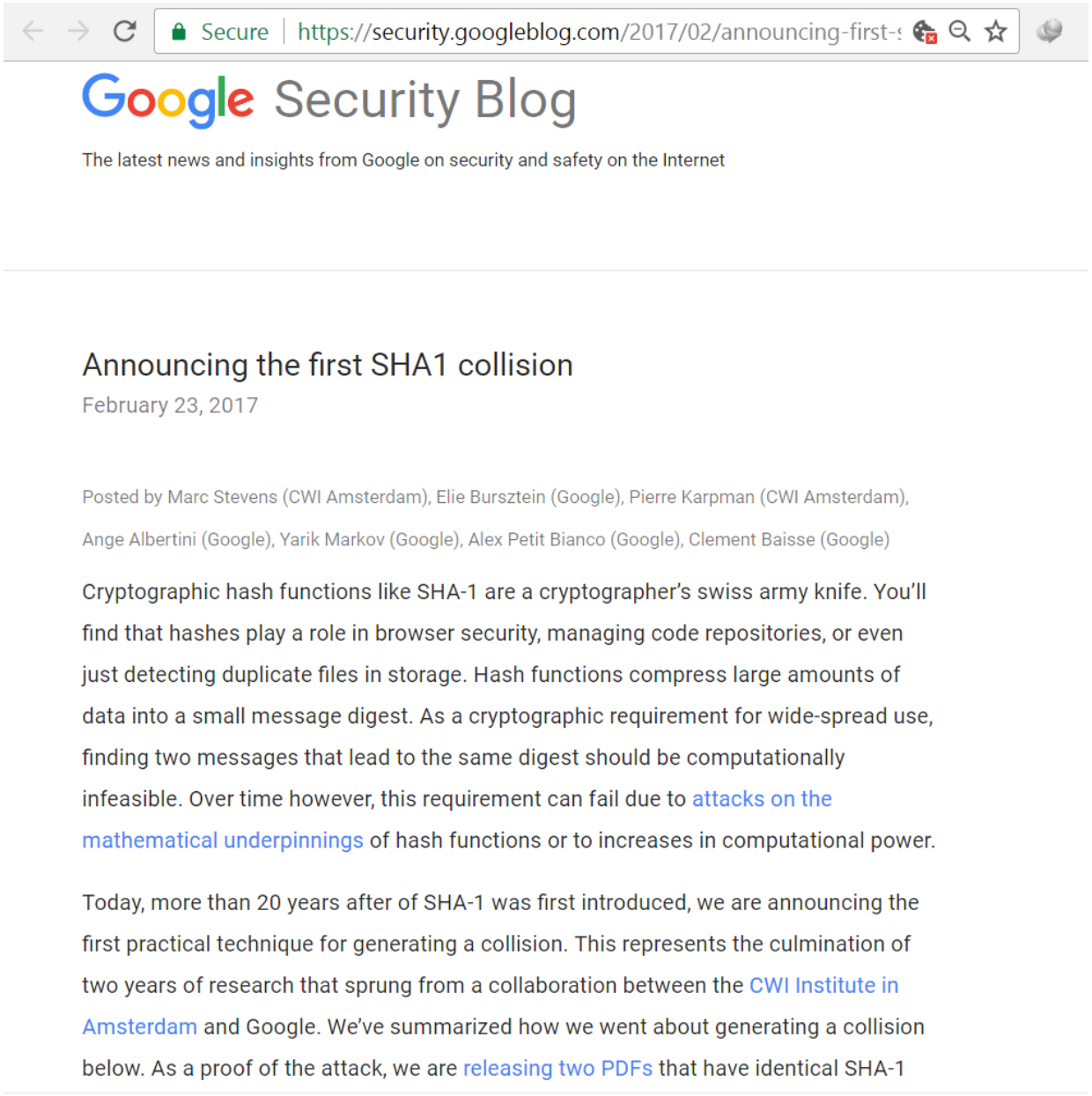
- SHA-0, SHA-1, SHA-2, SHA-3

- 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.
- SHA-1 is a popular standard. It produces 160-bits message digest. It is employed in SSL, SSH, etc.
- In 1998, an attack that finds collision of SHA-0 in 2^{61} operations was discovered. (Using the straight forward birthday attack, collision can be found in $2^{160/2} = 2^{80}$ operations). In 2004, a collision was found, using 80,000 CPU hours. In 2005, Wang Xiaoyun et al. (Shandong University) gave attack that can finds collision in 2^{39} operations.
- 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-rounds SHA-256 (the full SHA-256 takes 64 rounds)
- In 2005, Xiaoyun Wang et al gave a method of finding collision of SHA-1 using 2^{69} operations, which was later improved to 2^{63} . A collision was found in 2017. It took 110 GPU years, completed 2^{63} SHA1 operations. https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html
- In Nov 2007, NIST called for proposal of SHA-3. In Oct 2012, NIST announced the winner, Keccak (pronounced "catch-ack").

(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)

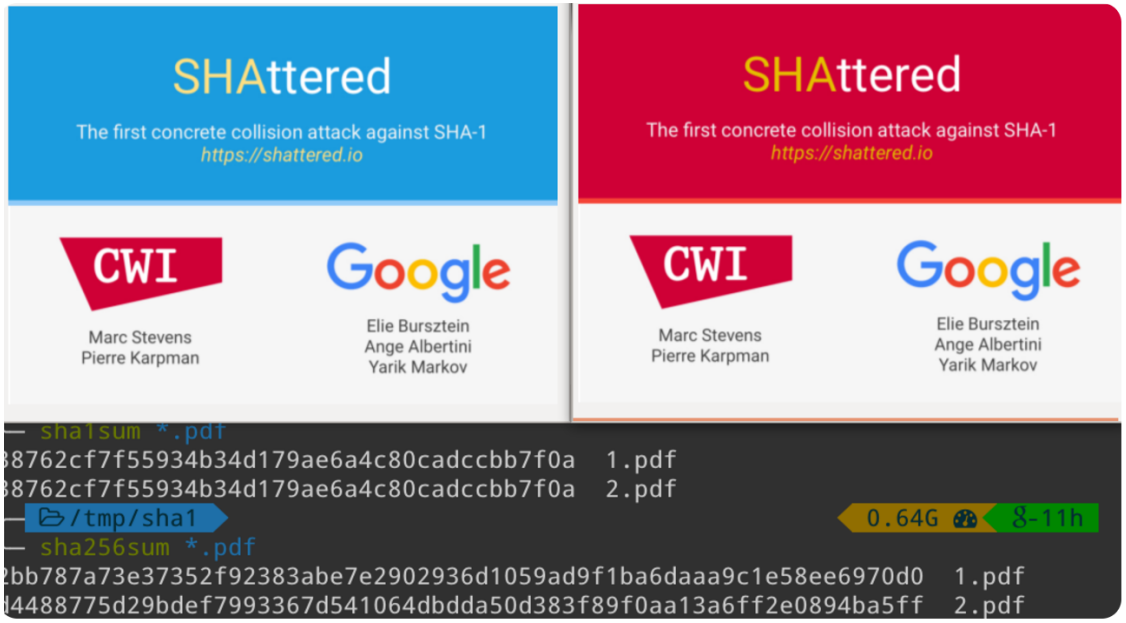
Cryptographic hash functions like SHA-1 are a cryptographer's swiss army knife. You'll find that hashes play a role in browser security, managing code repositories, or even just detecting duplicate files in storage. Hash functions compress large amounts of data into a small message digest. As a cryptographic 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/

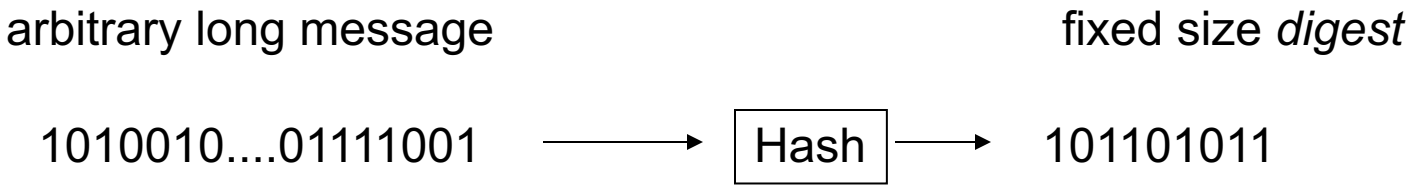


- MD5

- Designed by Rivest. MD, MD2, MD3, MD4, MD5, 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.

3.2.1 keyed Hash

Recap: Hash (no secret involved)



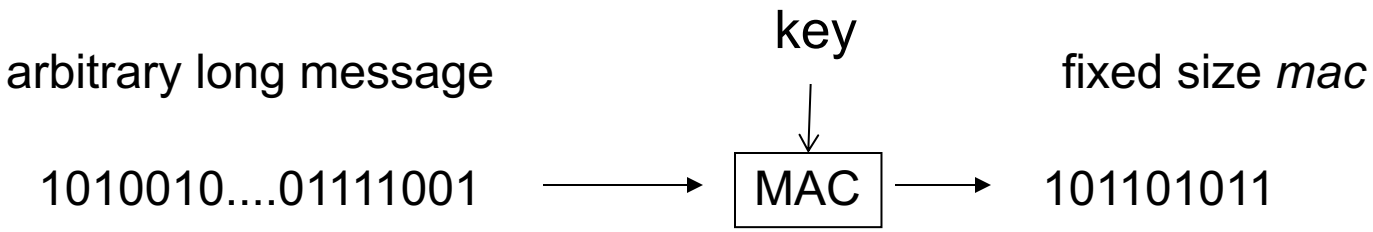
Security requirement (collision):

- **(*Collision-resistant*)** It is difficult for an attacker to find two different messages m_1 , m_2 that "hash" to the same digest. That is,

$$h(m_1) = h(m_2)$$

Recap: keyed-Hash (aka MAC) (a secret key is involved)

A keyed-hash is a function that takes an arbitrary large message and a secret key as input, and outputs a fixed size (say 160 bits) ***mac (message authentication code)***.



- **Security requirement (forgery)**: After seen multiple valid pairs of messages and their corresponding mac, it is difficult for the attacker to forge the mac of a message not seen before. (The precise formulation is quite involved, and with many variants. Higher level crypto module CS4236 would cover this)

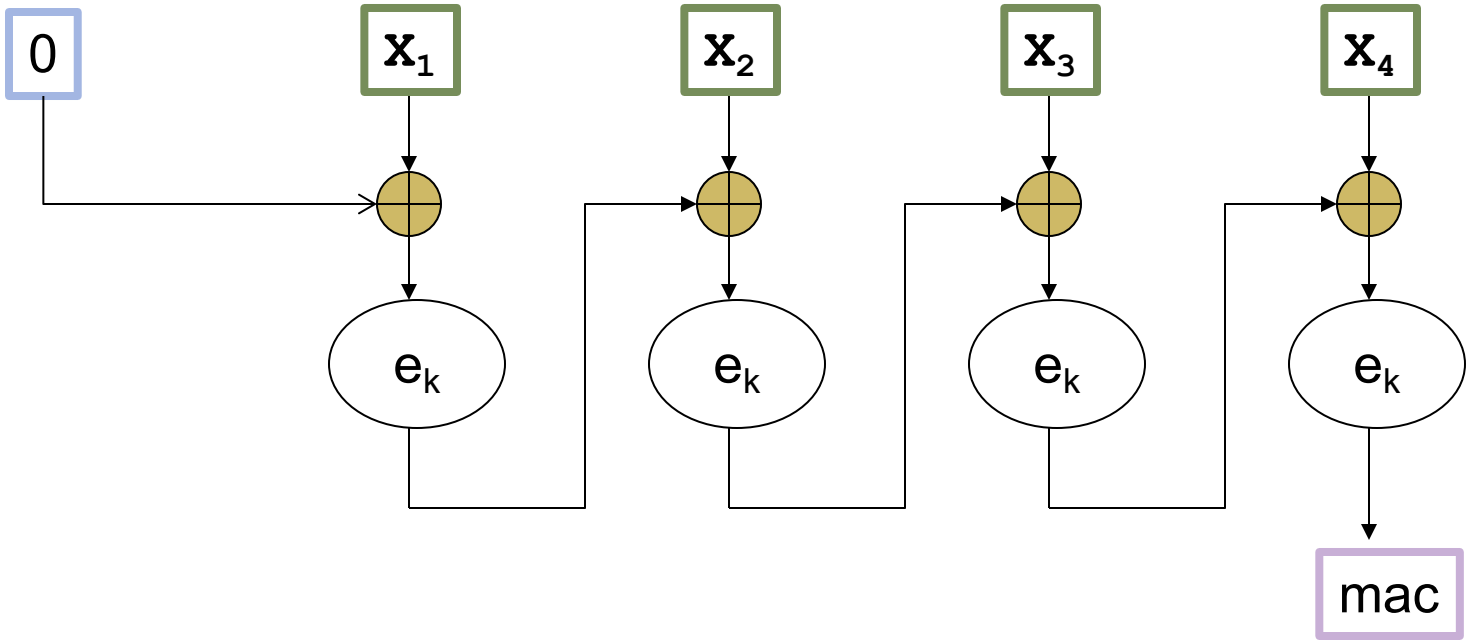
Popular keyed-hash (MAC)

- CBC-MAC (based on AES operated under CBC mode)
- HMAC (based on SHA)
  Hashed-based MAC

standard: RFC 2104. http://tools.ietf.org/html/rfc2104

CBC-mac

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)

(the above are in hexadecimal)

3.3. Data Integrity: Hash (without secret keys)

Application of (unkeyed) Hash for integrity

Consider this example

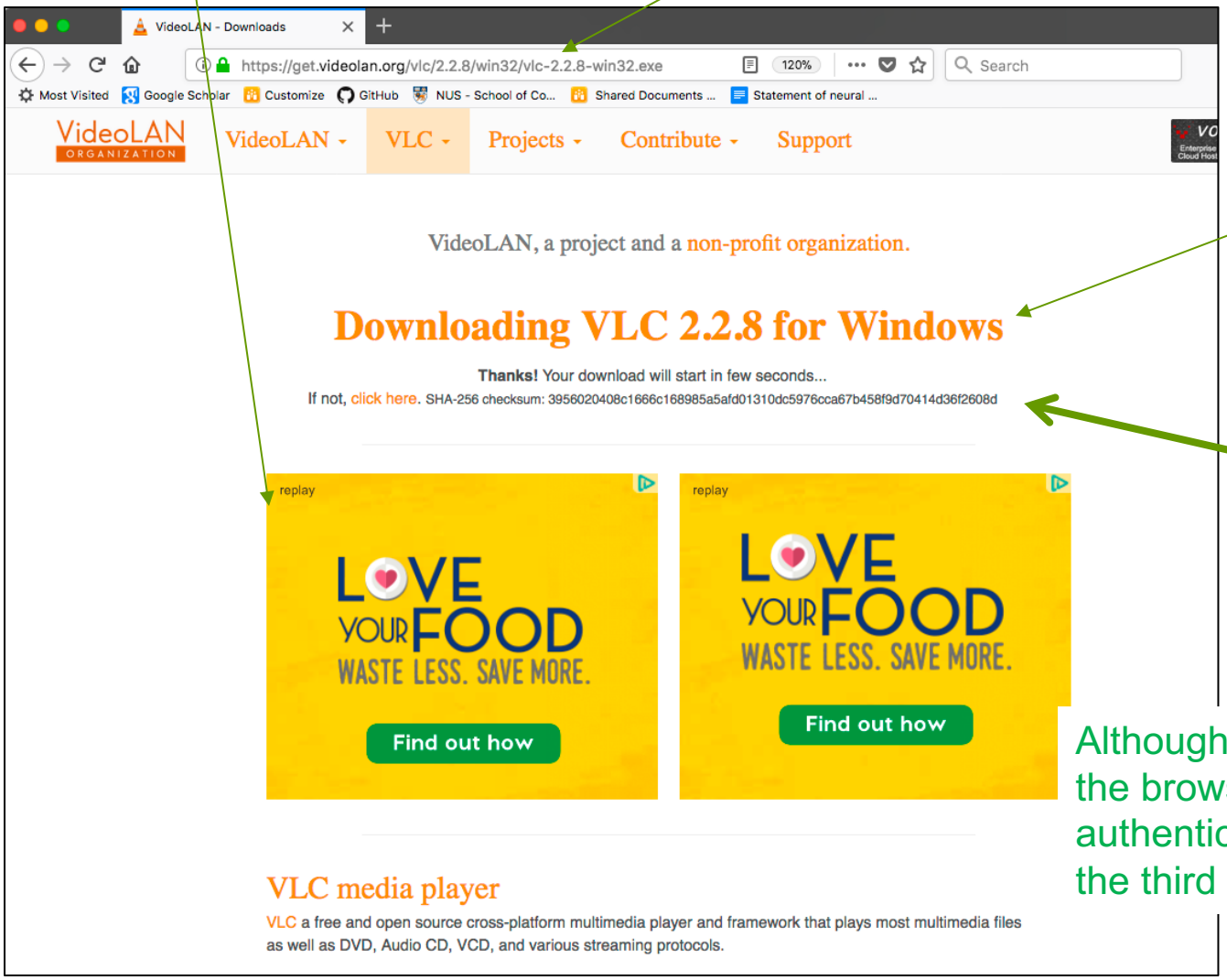
- Alice downloaded a software vlc-2.2.8-win32.exe from the web. Is the downloaded file authentic?
  (see the "checksum" in next slide)

Specifically,

1. Alice visits the website of VLC.
2. Since the website is hosted with HTTPS protocol, Alice is being assured that the content displayed on the browser is from VLC and authentic (we will discuss https later).
3. However, the downloading site is 3rd-party, i.e. the actual file vlc-2.2.8-win32.exe is hosted in another website. The communication channel to the 3rd party website is not secure, and there is a possibility that the 3rd party website is malicious, giving out virus infested software.
4. After Alice downloaded the file, she can check the integrity of the file by matching the "hash" of the file, with the "SHA-256 checksum" listed in the website. If it matches, then Alice is very sure that the file is intact. If not, certainly the file is corrupted.

Ads from 3rd party.

https. So the content displayed is from **videolan.org** and is authentic.



Actual file is hosted in a third party site

the digest (aka checksum).

Although the information displayed in the browser is verified to be authentic, what about the file from the third party site?

Question: why ”videolan.org” in the url is boldfaced? (to be discussed later)

Application of (unkeyed) Hash for integrity

In this scenario, we assume that there is a secure channel to send short piece of information. (in VLC example, this channel is the https)

Let ***F*** be the original data. Alice obtains the digest h(***F***) from the secure channel.

Alice obtains a file, say ***F'***, whose origin claims that the file is F. Alice computes and compares the digests h(***F***), h(***F'***).

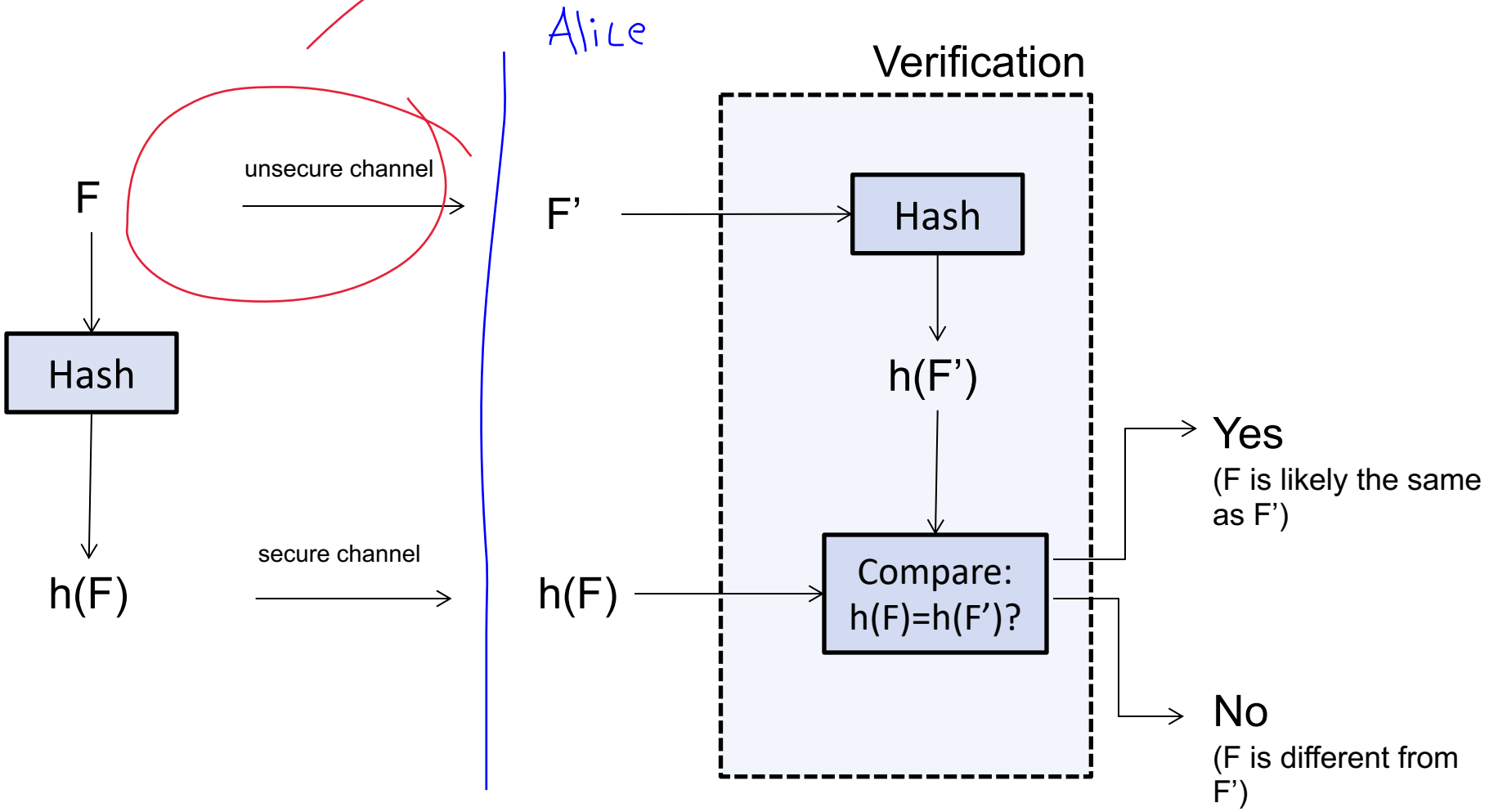
If they are the same, ***F'*** is indeed same as ***F*** with very high confidence. If they are different, then F' must be different from F, i.e. integrity compromised.

h(***F***) = h(***F'***) ⇒ with high probability, ***F***=***F'***

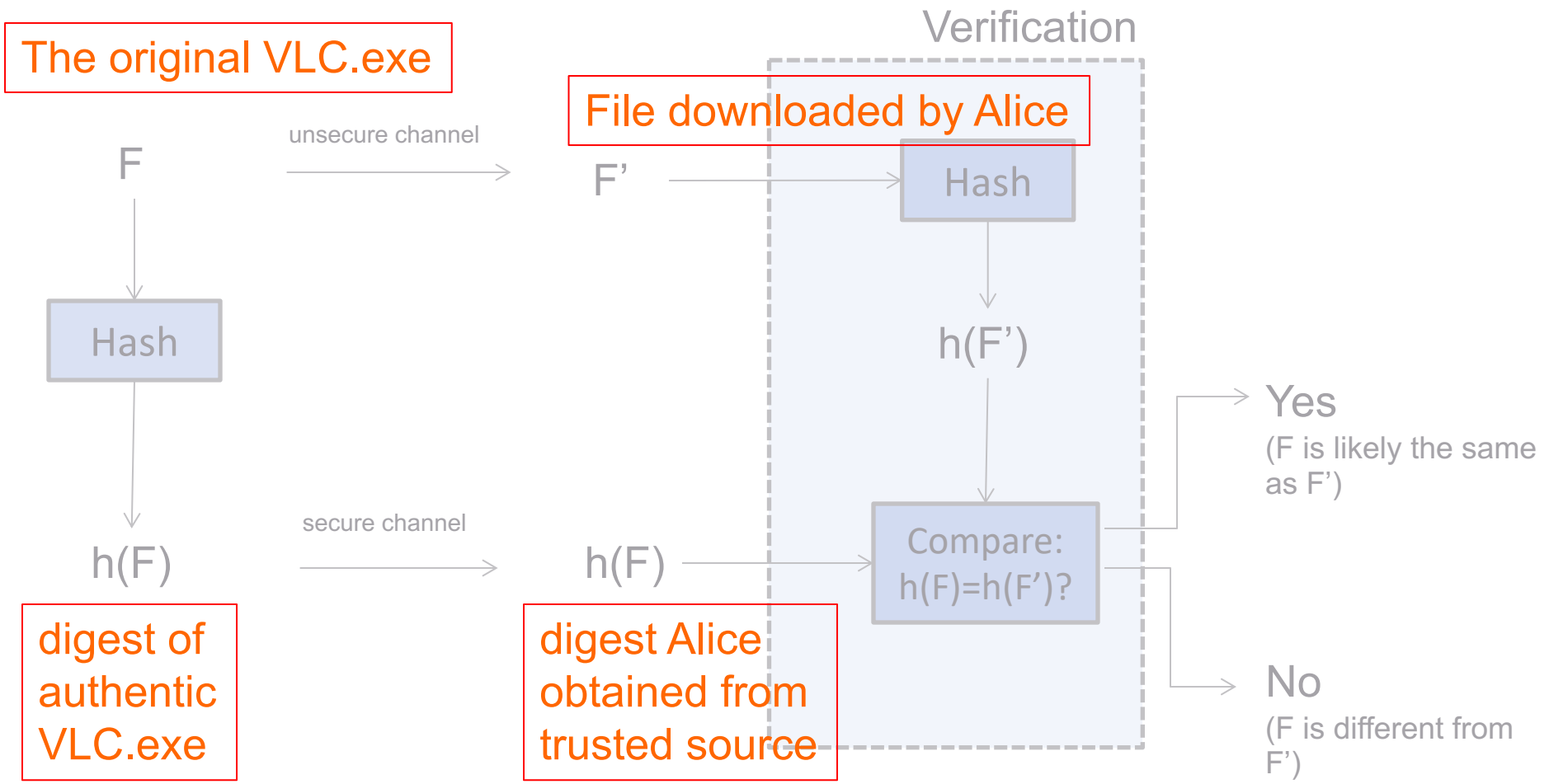
h(***F***) ≠ h(***F'***) ⇒ ***F*** ≠ ***F'***

(unkeyed) hash

an attacker can tap here, so will know the original file and the digest
so just need to upload a F" where digest is the same



(unkeyed) hash



Remarks

- What would an attacker who wants to pass an infected VLC.exe do? *(find a collision)*

$$h(F_1) = h(F_2)$$

- We may argue that with the digest, the verifier can be assured that the data are authentic, and thus 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.

3.4 Data Origin Authenticity (mac)

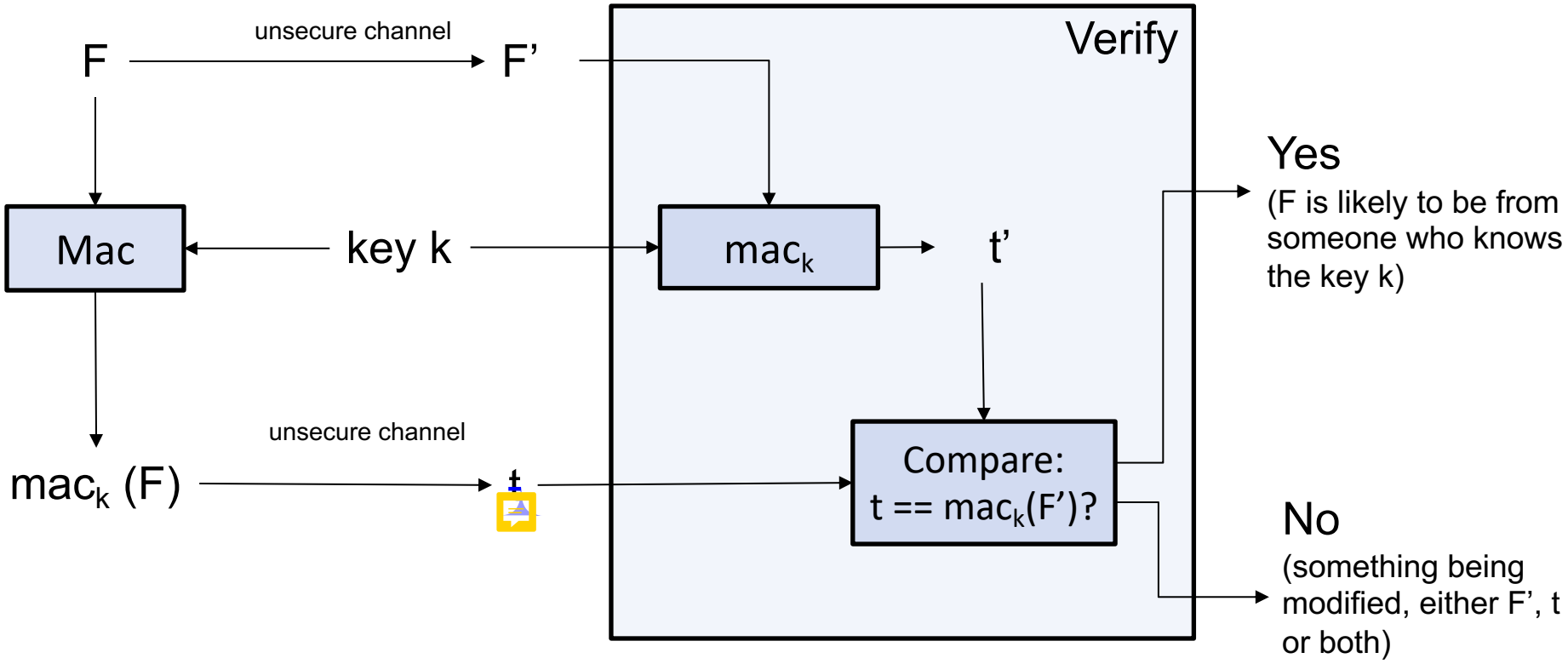
- In the previous example (on vlc), we assume that there is a secure channel to send the digest.

- There are scenarios that we don't have a secure channel to deliver the digest. (in the vlc example, it relies on the fact that we have https. What if we don't have https?)

- In such scenarios, we can protect the digest with the help of some secrets.
  - In the symmetric key setting, it is called the **mac.**
  - In the public key setting, it is called the **digital signature.**

mac (Message Authentication Code)

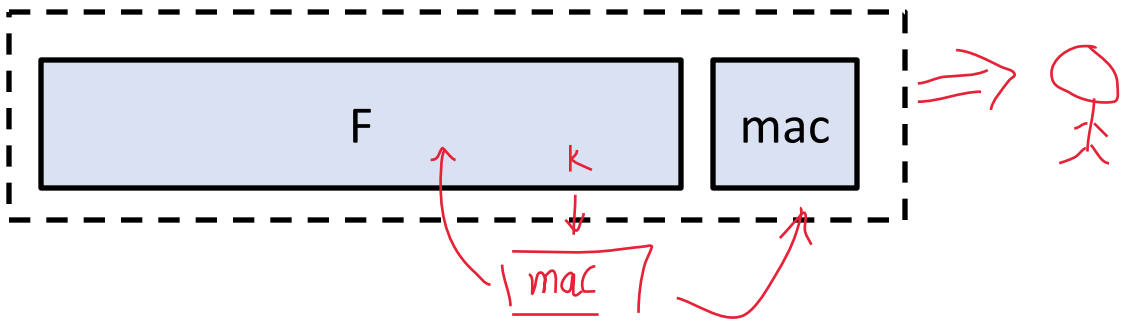
In this setting, the mac might be modified by attacker. If such case happened, it can be detected with high probability.



Security Requirement: Without knowing the key k, it is difficult to forge a mac.

Remark

- What would an attacker do? (*forge a valid pair of (message,mac)*)
- Note that there is no issue on confidentiality. In fact, the data F can be sent in clear.
- Typically, the *mac* is appended to F. They are then stored as a single file, or transmitted through the communication channel together. (Hence, mac is also called the *authentication tag,* or *authentication code*)
- Later, an entity who wants to verify the *authenticity* of F, can carry out the verification process using the secret key.



3.5 Data Origin Authenticity (Signature)

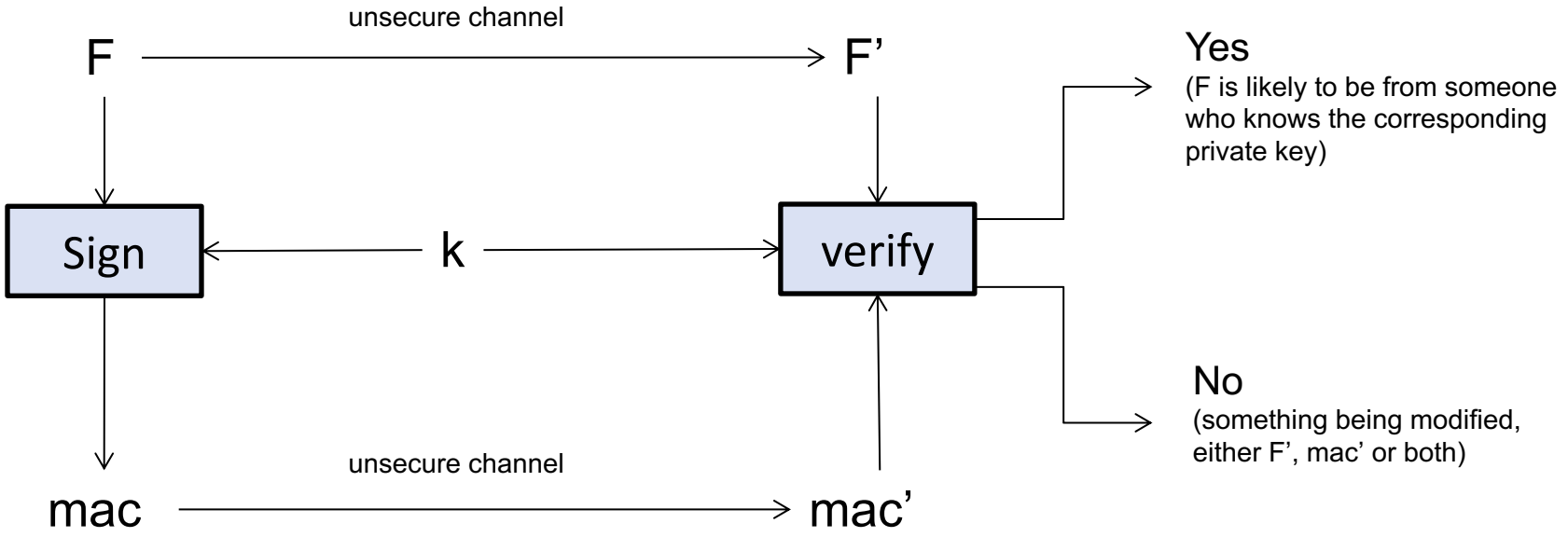
Signature

- The public key version of MAC is called Signature.

- Here, the owner uses the ***private key*** to generate the signature. The public can use the ***public key*** to verify the signature.

- So, anyone can verify the authenticity of the data, but only *the person who know the private key* can generate the signature.

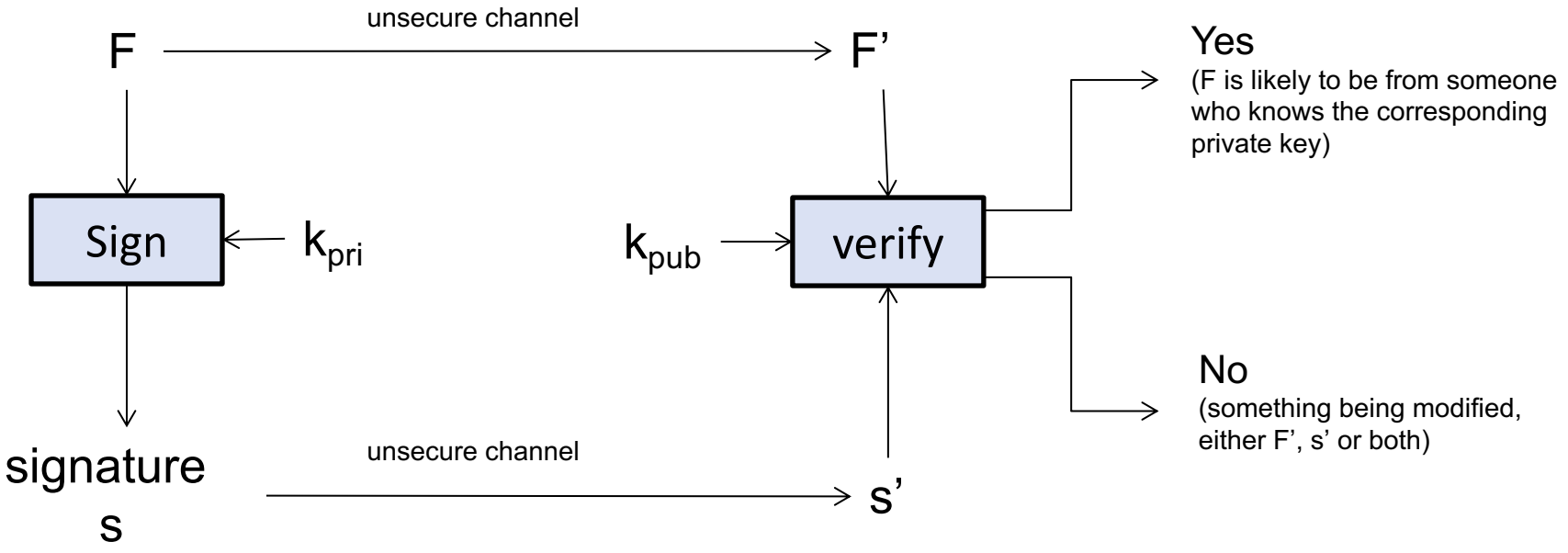
mac


unsecure channel
F
F'
Yes
(F is likely to be from someone who knows the corresponding private key)
Sign
k
verify
No
(something being modified, either F', mac' or both)
unsecure channel
mac
mac'

Signature

Verifier and Signer using different key.

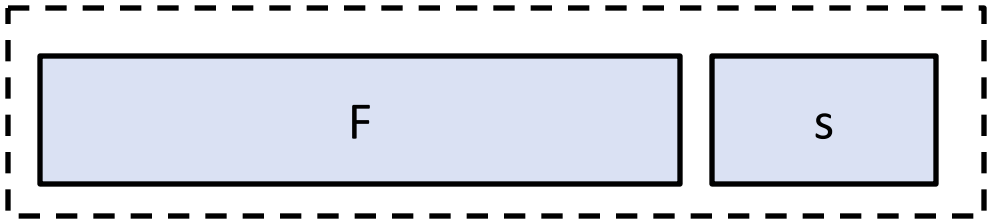
k_{pri}: private key
k_{pub}: public key



Security Requirement: Without knowing the private k_{pri}, it is difficult to forge a signature.

Remark

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



- 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 public key. The valid signature ***can only be*** computed by someone who knows the private key, so, if it is valid, then F must be authentic.

- Anyone can verify using the public key.

What so special of signature compare to mac?

- Beside the ease of key management compare to mac, signature achieve an additional security requirement.

- We can view the digital signature as the counterpart of the handwritten signature in legal document. A legal document is authentic or certified, if it has the correct handwritten signature. No one, except the authentic signer, can generate the signature.

- Signature scheme achieves ***Non-repudiation.***

Non-Repudiation: *Assurance that someone cannot deny previous commitments or actions.*

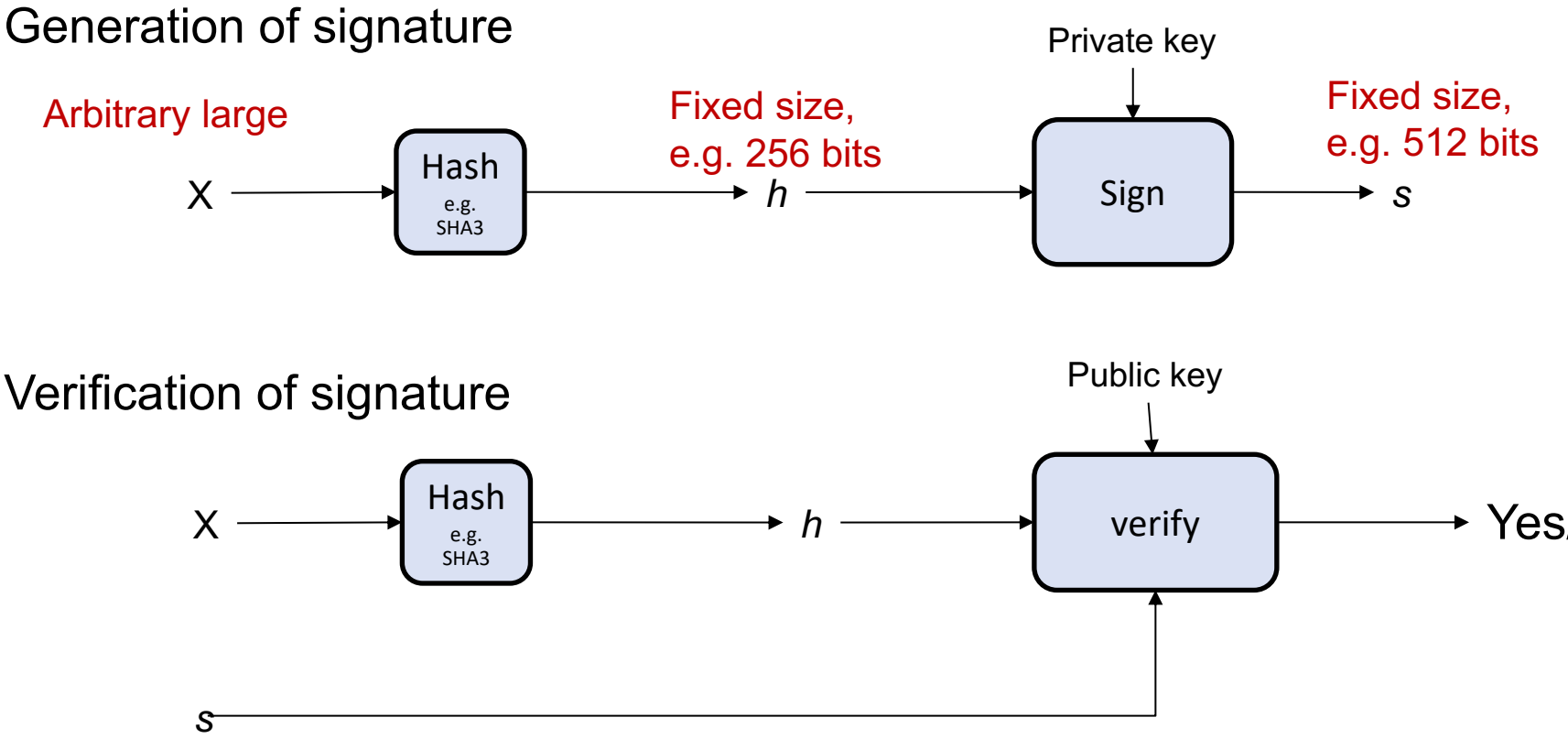
Example on non-repudiation

- (using mac). Suppose Alice sent Bob a message appended with a mac. The mac is computed with a 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 signature). 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. This is because only the person who knows the private key can sign the message and only Alice knows the private key. Hence, the signature is a proof that Alice generated the message.

Design of Signature scheme

- Most signature scheme consists of two components. An unkeyed hash, and the sign/verify algorithm .



Popular Signature scheme

- A popular group of schemes use RSA for the sign/verify component. :

  RSASSA-PSS, RSASSA-PKCS1: signature scheme based on RSA

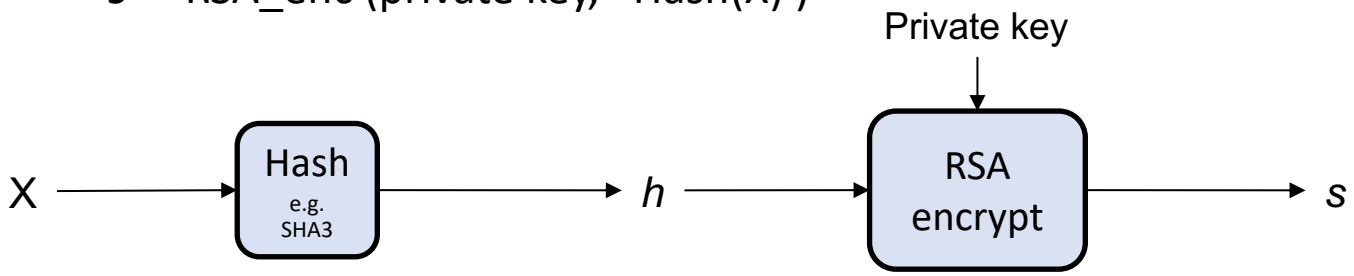
- DSA (Digital Signature Algorithm) is another popular standard whose security depends on discrete log.

  https://en.wikipedia.org/wiki/Digital_Signature_Algorithm#Implementations

RSA-based signature

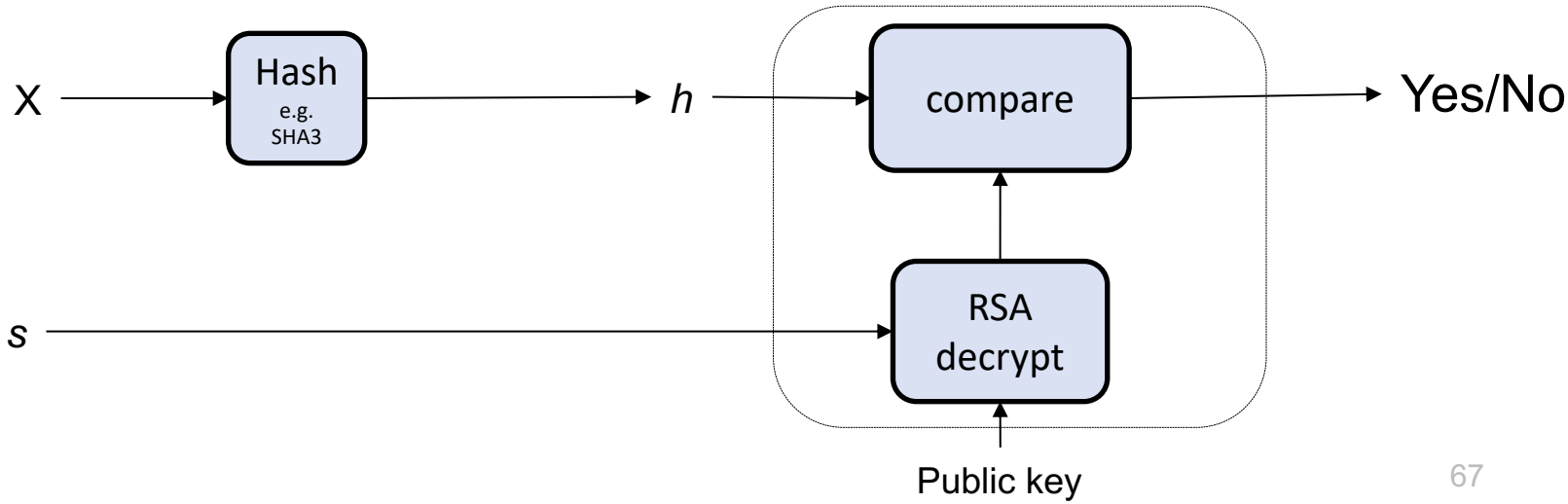
- Use RSA for signing and verification. Essentially, the signature is the "encrypted" digest. During verification, decrypt to obtain the digest and compare. *A messy notation issue: Previously, we use public key to encrypt. Here, we use private key to encrypt. Recall that for RSA, we can flip the role.*

s = RSA_enc (private key, Hash(X))



- Verification of (X,***s***) is done by using the public key.

If Hash (X) = RSA_dec (public key, s) then accept, else reject.



Not necessary to have “encryption” in signature scheme

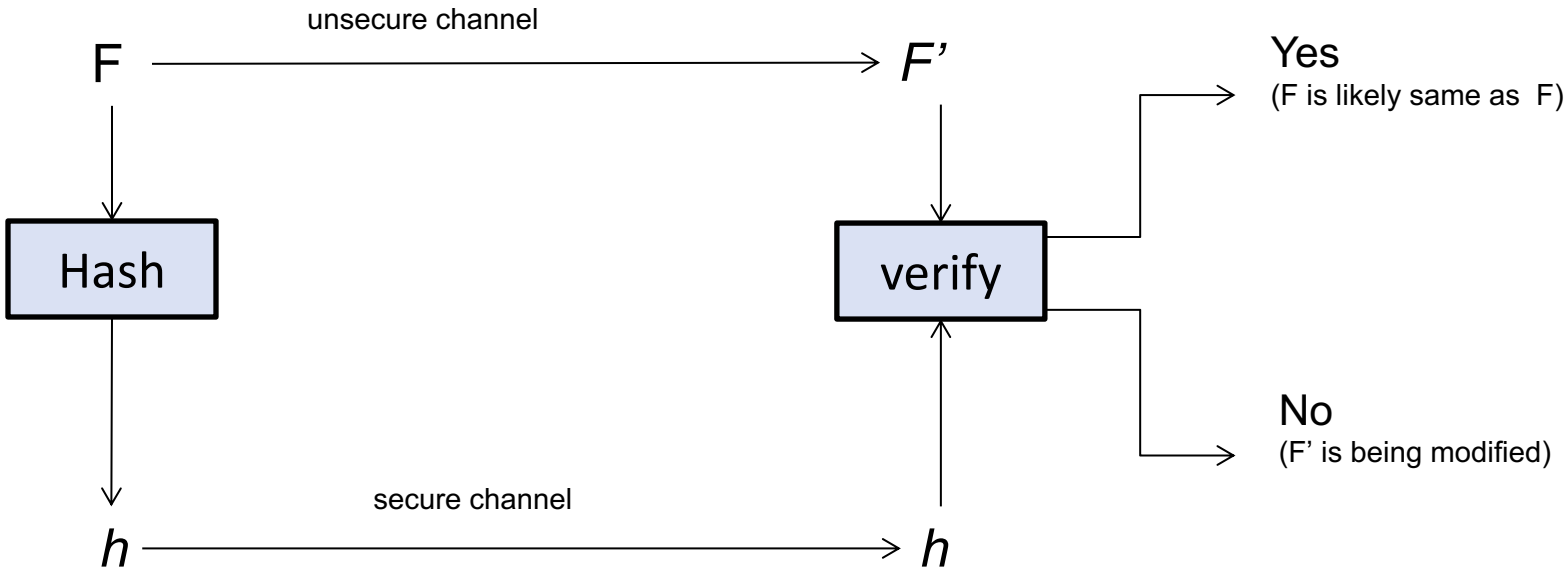
- Many books/documents describe a signature as the encryption of the hash. This is not true.
- Popular schemes such as DSA employ do not “encrypt” (there isn’t a way to decrypt).
- So, a more accurate way is to say: “hash-and-sign”

The method of hash-and-encrypt is a special case of hash-and-sign.

Summary

3.3 Digest (Hash)

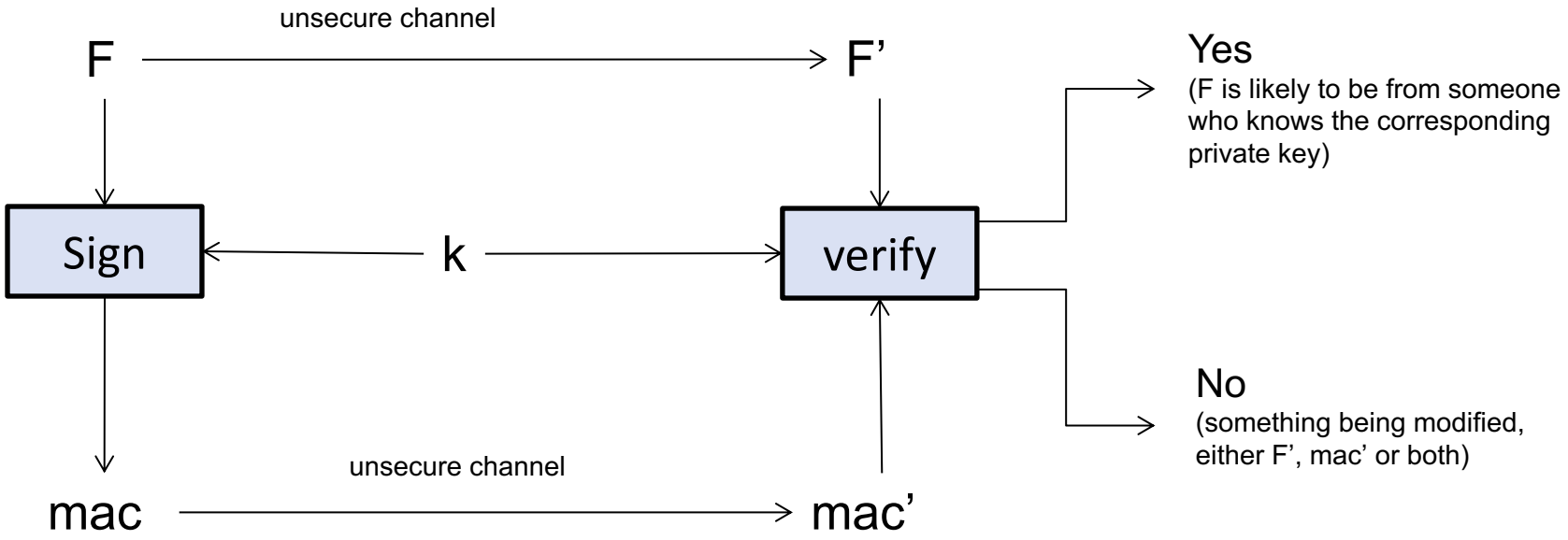
The digest must be sent through secure channel



Security Requirement: Different to find a F' with the same digest

3.4 MAC (Message Authentication Code)

The mac can be sent through an unsecure channel. Both

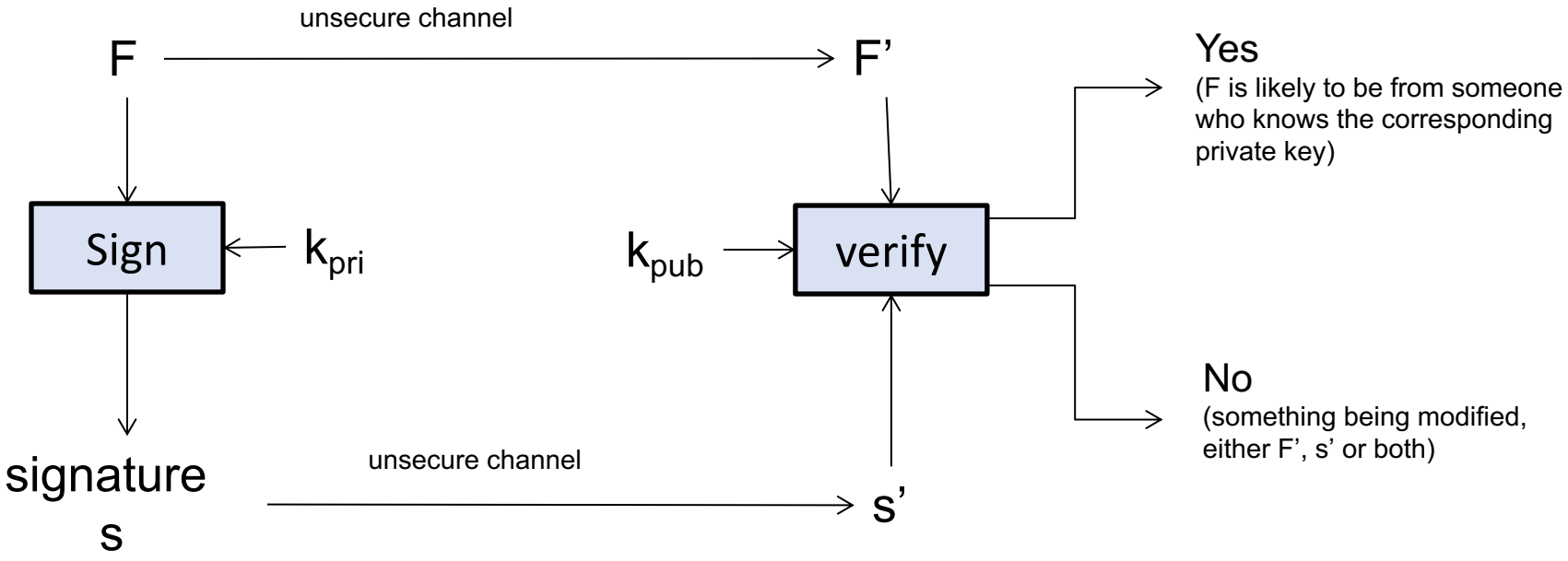


Security Requirement: Without knowing *k*, it is difficult to forge a mac.

3.5 Signature

Verifier and Signer using different key.

k_{pri}: private key
k_{pub}: public key



Security Requirement: Without knowing the private k_{pri}, it is difficult to forge a signature.

3.6. Some attacks and pitfalls

3.6.1 Birthday attacks

- Hashes are designed to make collision difficult to find. Recall that a collision consists of two different messages x_1, x_2 that give the same digest, i.e.

$$h(x_1) = h(x_2) \quad \text{and } x_1 \neq x_2$$

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

```
for i = 1 to inf:
        randomly pick 2 x1,x2
        check for collision, break
end
```

128 bit

$\therefore P = \frac{1}{2^{128}} \Rightarrow \frac{1}{P} = 2^{128}$

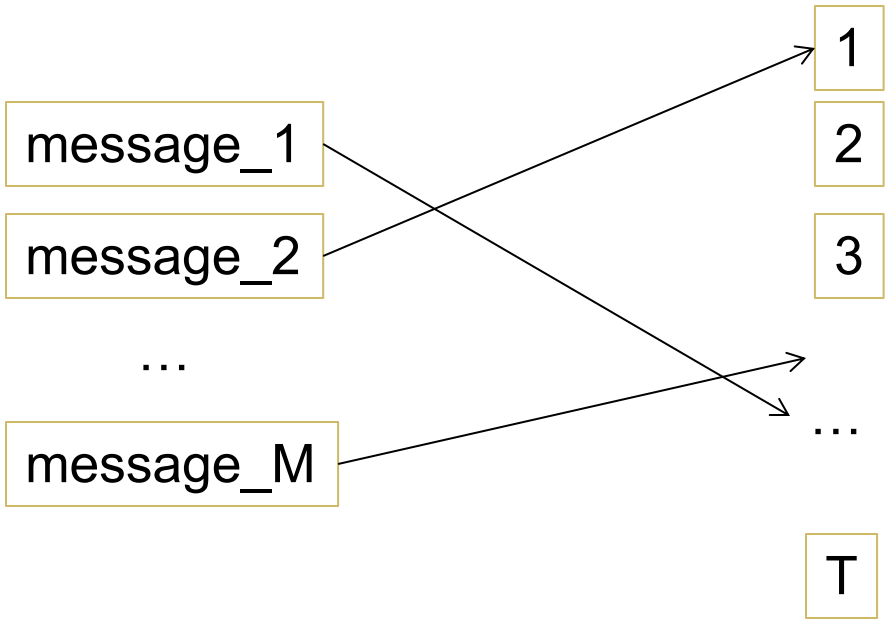
```
for i=1 to inf:
        randomly pick 2000000 messages
        check if there are any collision, break
end
```

- In a class of 25 students, with probability more than 0.5, there is a pair of students having the same birthday.

- Suppose we have M messages, and each message is tagged with a value randomly chosen from {1,2,3,…,T}. If

$$M > 1.17\ T^{0.5}$$

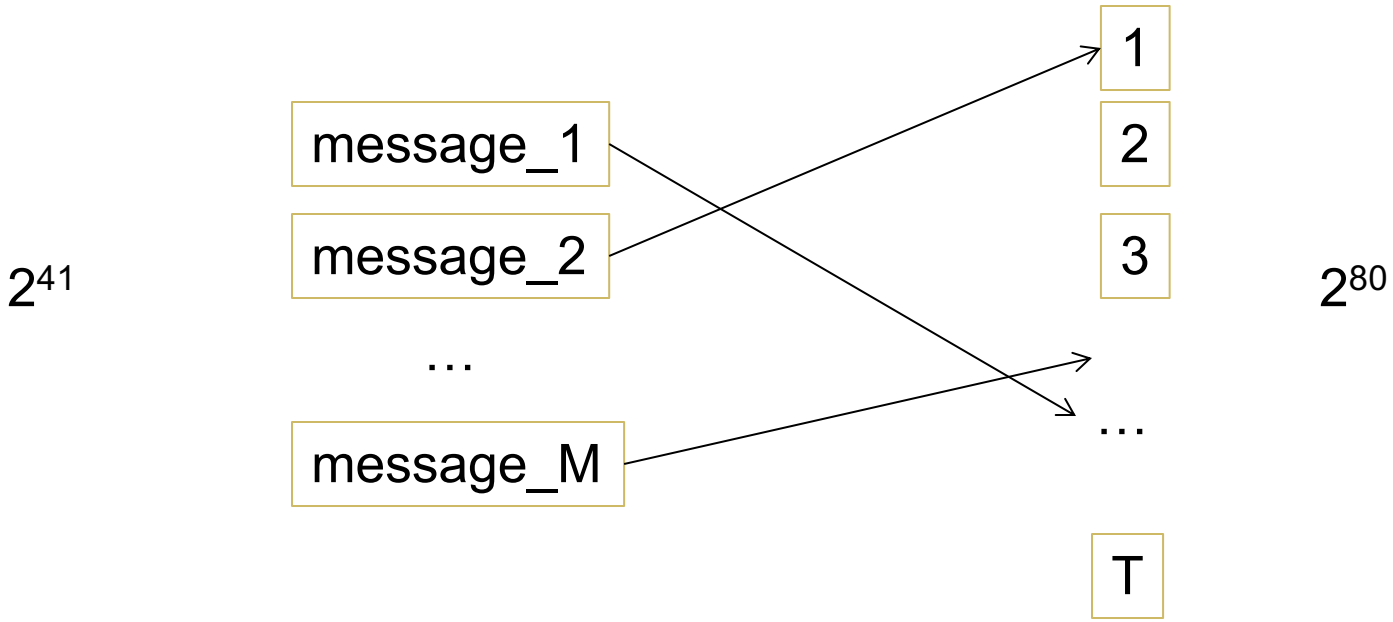
then, with probability more than 0.5, there is a pair of messages tagged with the same value.



a lot of T's
but few messages

- Suppose the digest of a hash is 80 bits (T= 2^{80}). An attacker wants to find a collision.

  If the attacker randomly generates 2^{41} messages (M=2^{41}), then M> 1.17 $T^{0.5}$. Hence, with probability more than 0.5, among the 2^{41} messages, two of them give the same digest.



in general, the probability is approximately 1- exp (- M^2/ (2T))

Length of digest

Read the NIST recommendation of key-length

(http://www.keylength.com/en/4/)

When key length for symmetric-key is 112, the corresponding recommended length for digest is at least 224. Why the digest length has to be significantly larger?

(birthday attack)

Question

- Back to the example of ECB mode-of-operation. (penguin)

- One remedy is as follow: each block is to be encrypted with an IV. Although secure, it increases the ciphertext size. This is because the IV for each block has to be included in the ciphertext. Suppose each IV is 128 bits, there will be an overhead of 128 bits per block. Hence it is desirable to keep the size of IV small.

- What about setting the IV to 16 bits?

Suppose it is decided to use 16 bits for the IV, and each image has around $2^{10} = 1024$ blocks. The IVs are Independently and randomly chosen.

1. What is the probability that the first two blocks have the same IV? 2^{-16}
2. What is the probability that, among the 1024 blocks, there exists two blocks having the same IV? *Extremely high, certainly > 0.5*

3.6.2 Using encryption for the purpose of authentication

Encryption schemes may provide false sense of security. Consider this design of a mobile apps from a 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 sms only consist of readable ascii characters. We assume that there is a way to encode binary string using the readable characters). The format of the instruction is:

X P

where X is an 8-bit string specifying the operation, and P is a 120-bit string specifying the parameter. So, an instruction is of size 128 bits. If an operation doesn't need a parameter, P will be ignored. There is a total of 15 valid instructions.

E.g.

00000000 P : send the GPS location to phone number P via sms. If P is not a valid phone number, ignore.

11110000 P : rings for P seconds. If P>10, ignore.

10101010 : self-destruct now!

- An instruction is to be encrypted using AES CBC-mode with 256-bit key, encoded to readable characters and sent as sms. (recap: block size of AES is 128 bits).

- After a mobile phone received a sms, it decrypts it. If the instruction is invalid, it ignores the instruction. Otherwise, it executes the instruction.

- The company XYZ claims that "*256-bit 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*".

- Something is wrong here.

(If an attacker sends a randomly chosen message to the mobile phone, what is the probability that the mobile phone self-destruct?)

Remarks

- Encryption is designed to provide confidentiality. It does not guarantee integrity and authenticity.
- In the previous example, XYZ wants to achieve "authenticity", but wrongly employing encryption to achieve that.
- A secure design could use mac instead of encryption.

[optional] Also note that a lot of details are omitted. Simply adding mac to the instructions is not sufficiently secure. To prevent "replay" attack, "nonce" is required.

3.6.3 Time-space Tradeoff

Reference:

- See http://en.wikipedia.org/wiki/Rainbow_table
  Section “Precomputed hash chains”.

The above wiki page describe “Rainbow table” which is an improved variant of time-memory tradeoff. The original basic variant is described in the section “Precomputed hash chain”.

3.6.3 Time-memory tradeoff for Dictionary Attack

- Let consider a hash H( ) which is collision resistant (e.g. SHA3() is believed to be so). Since it is collision resistant, it is also one-way. Thus, given a digest y, it is difficult to find a x s.t. H(x)=y.

- Suppose we know that the message x is chosen from a relatively small set of *dictionary D*. For illustration, let's assume x is a randomly and uniformly chosen 50-bits message.

- Even if H() is one-way, given the digest y, it is still feasible to find a x in D s.t. H(x) =y. This can be done by exhaustively searching the 2^{50} messages in D. Although feasible, this would take days of computing time. As the attacker, we want to speedup the inverting process to support "realtime" attack.

Using a large table.

- We are allowed to perform pre-processing. One straightforward method is to build a dataset with 2^{50} elements (H(x), x) for x = 0, 1, 2, …, 2^{50}-1

  (here, we view the 50 bits message as an integer)

  and store these elements in a data structure ***T*** that supports fast lookup (for instance, hash table that facilitates constant-time lookup)

Now, given a digest y, we can query the data structure and readily find the associated x.

However, such table is too large. (2^{50} entries = 2^{10} "Tera" entries)

The time-memory tradeoff is a technique to "tradeoff" time (i.e. slower lookup time) for memory (i.e. smaller storage)

Time-memory Tradeoff:

- The main idea is to use a precomputed hash-chain. (The term "hash-chain" appears in different context and refer to different techniques.)

- Define a ***reduce function*** R() that maps a digest y to a word w in the dictionary ***D***. For illustration, if ***D*** consists of all 50-bit messages, and each digest is 320 bits, then a possible reduce function simply keeps the first 50 bits of input.

$$R(b_0 b_1 \ldots b_{320}) = b_0 b_1 \ldots b_{49}$$

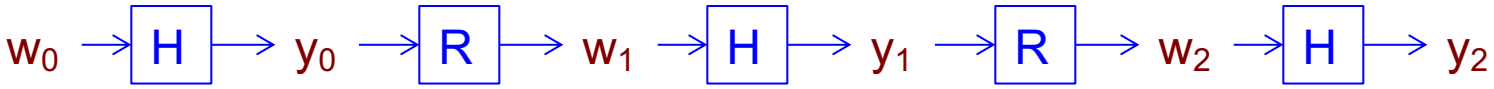
Here is a pre-computed chain. It starts from a randomly chosen word w_0 in ***D***.

$w_0 \rightarrow y_0 = H(w_0) \rightarrow w_1 = R(y_0) \rightarrow y_1 = H(w_1) \rightarrow w_2 = R(y_1) \rightarrow y_2 = H(w_2)$

e.g

"hello" → **A0C0…20** → "qwert1" → **03F0…50** → "Pikachu" → **77FF…3A**

$w_0 \rightarrow y_0 = H(w_0) \rightarrow w_1 = R(y_0) \rightarrow y_1 = H(w_1) \rightarrow w_2 = R(y_1) \rightarrow y_2 = H(w_2)$



Call w_0 the starting-point, and y_2 the ending-point. Store the pair (w_0, y_2) in the data-structure ***T***. The process is repeated with other randomly chosen starting points.

During the query, given a digest y. First search for y in the data-structure ***T***.

- Suppose y is in ***T***. That is, y is one of the ending-point. Let's assume that w_0 is the corresponding starting point (hence $y = y_2$ in the above chain). Note that a pre-image of y is w_2, but at this point, we don't know the value of w_2. Nevertheless, the fact that y is the ending-point implies that w_2 is within the chain starting from w_0. So, we can start to construct the chain w_0, y_0, w_1, y_1, w_2, y_2. When the process hits y_2, we have found the w_2.

- Suppose y is not in ***T***. Compute $y' = H(R(y))$. Search the data-structure for y'. Suppose y' is in ***T***. Let's assume that the starting-point be w_0 (hence $y' = y_2$). With high chances, $y = y_1$. So, a pre-image of y is w_1 (i.e. $H(w_1)=y$). At this point, we don't known w_1. But by constructing the chain from w_0, the pre-image w_1 can be found.

  If y' is not in T, compute $y'' = H(R(y'))$ and repeat the process.

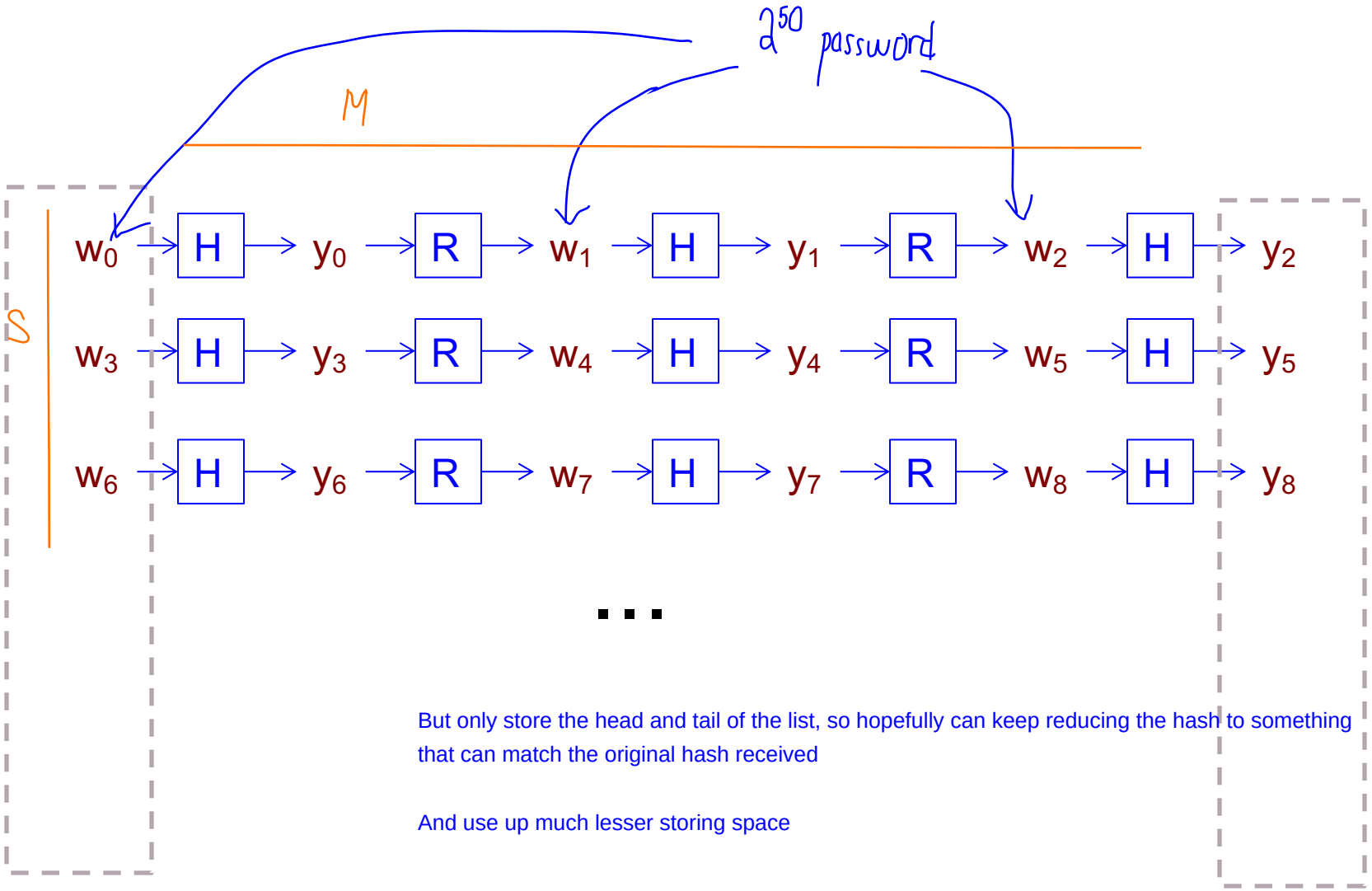
Without hash chain (tutorial 1)

On input y, find w s.t. H(w) = y

Receive this and work backwards

Try : $S \times M = 2^{50}$

On input y, find w s.t. H(w) = y



But only store the head and tail of the list, so hopefully can keep reducing the hash to something that can match the original hash received

And use up much lesser storing space

Analysis of time and space required.

Let's compare the space and time required compare to the simple method of using the full table.

Space: Reduction of space by a factor of 3. (why?)

Time per query: Number of hash operations increases by a factor of 3. Also require 2 reduce operations. (why?)

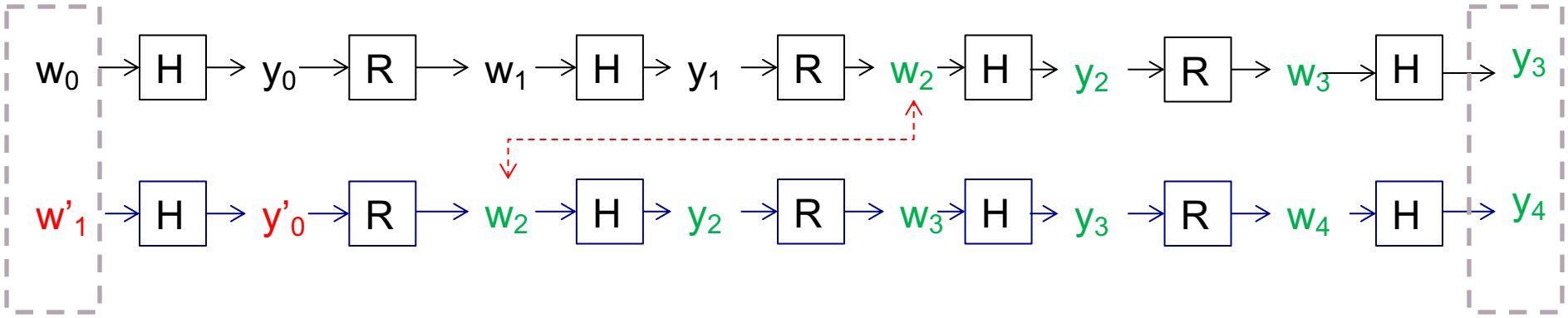
Accuracy: The chains contain repetitions. (why?)

In general, we can choose the length of the chain so that the reduction of space is a factor of k, with the penalty of increase the search time by a factor of k, where k is a parameter.

In our 50-bits example, if we choose $k=2^{15}$, and the number of entries in the table to 2^{35}, then the total number of entries covered in the "virtual table" is 2^{50} (as we shall see later, these 2^{50} entries are not unique). The query time increase to 2^{15} hash operations, which still can be computed in realtime, and the storage reduced to 2^{35} entries.

Limitation: Collisions due to reduce function (frequent)

- Collisions of the reduce function may happen frequently, i.e. two different digests being mapped to the same word.



When given y_0', the algorithm is unable to find w'_1.

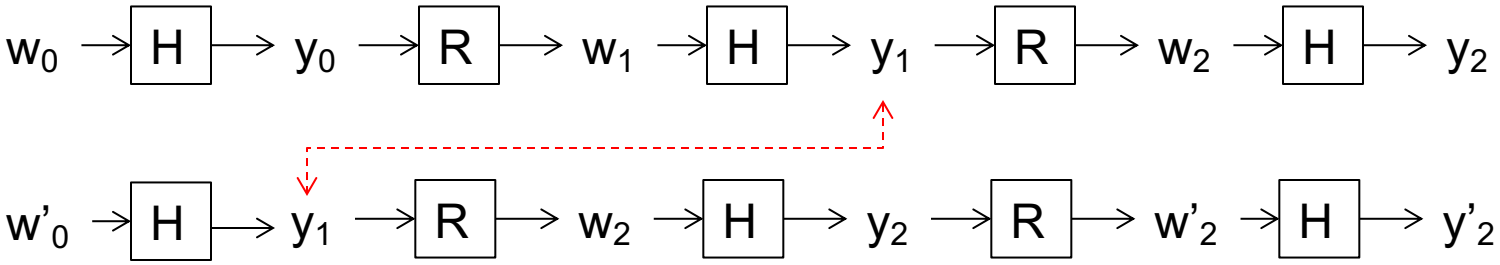
On input y_0', the algo performs these: (1) lookup y_0', (2) lookup y_2 (3) lookup y_3 (4) search the chain starting from w_0

This causes two issues:

- Efficiency in storage: Part of the chain is duplicated. (w_2, w_3 being stored twice)
- Efficiency in search: lead to searches in the wrong chains, before hitting the right chain. (for the query y_0', the lookup process would transverse both chains).

Remark: Collisions dues to Hash function (unlikely)

- The following collision dues to H() is extremely unlikely (since we assume that H() is "secure") and thus can be omitted in our design consideration.



Improved variant: Rainbow Table

Rainbow table gives a simple but effective method to address the collision issue in time-memory tradeoff. (method is simple but analysis quite complex).

Rainbow table utilizes multiple "reduce" functions. Details not included in this module. To find out more more:

Reference:
http://en.wikipedia.org/wiki/Rainbow_table
http://kestas.kuliukas.com/RainbowTables/

The original research paper:
P. Oechslin. *Making a Faster Cryptanalytical Time-Memory Trade-Off,* CRYPTO 2003.
http://lasec.epfl.ch/~oechslin/publications/crypto03.pdf