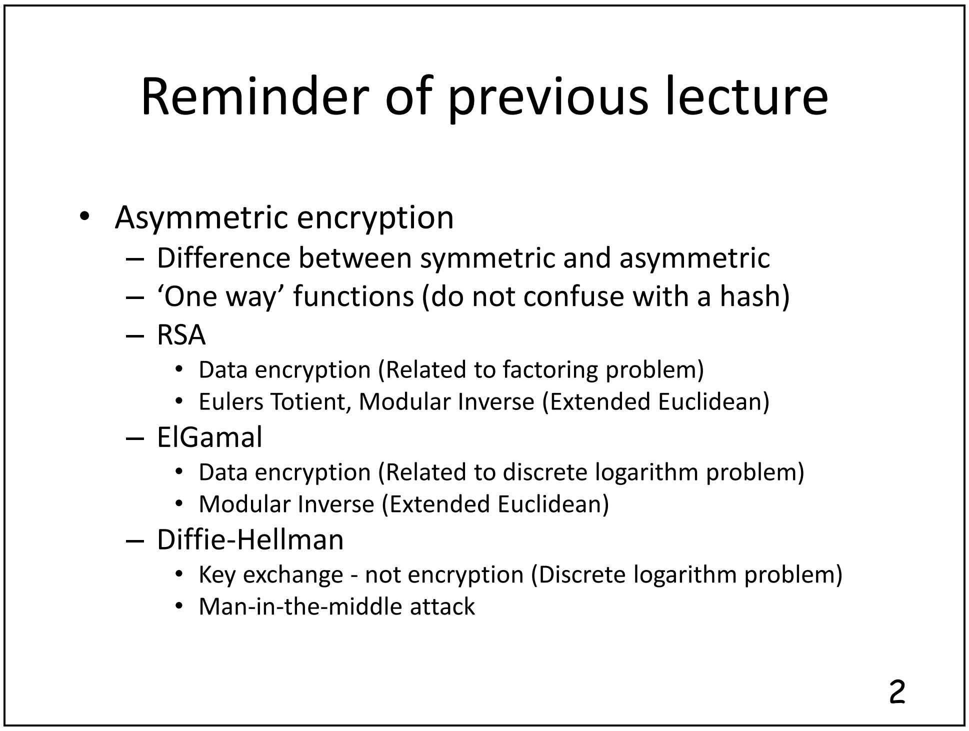
CS5285

# Information Security for eCommerce

Dr. Gerhard Hancke

CS Department
City University of Hong Kong



## Reminder of previous lecture

- Asymmetric encryption
  - Difference between symmetric and asymmetric
  - ‘One way’ functions (do not confuse with a hash)
  - RSA
    - Data encryption (Related to factoring problem)
    - Eulers Totient, Modular Inverse (Extended Euclidean)
  - ElGamal
    - Data encryption (Related to discrete logarithm problem)
    - Modular Inverse (Extended Euclidean)
  - Diffie-Hellman
    - Key exchange - not encryption (Discrete logarithm problem)
    - Man-in-the-middle attack

## Number Theory

- From after Problem Set 1 onwards:
  - Modular exponentiation (RSA/ElGamal/DH)
  - Modular Inverse (RSA/ElGamal)
  - Eulers Totient (RSA – but this is just (p-1)(q-1))
- Focus on being able to do RSA/ElGamal/DH

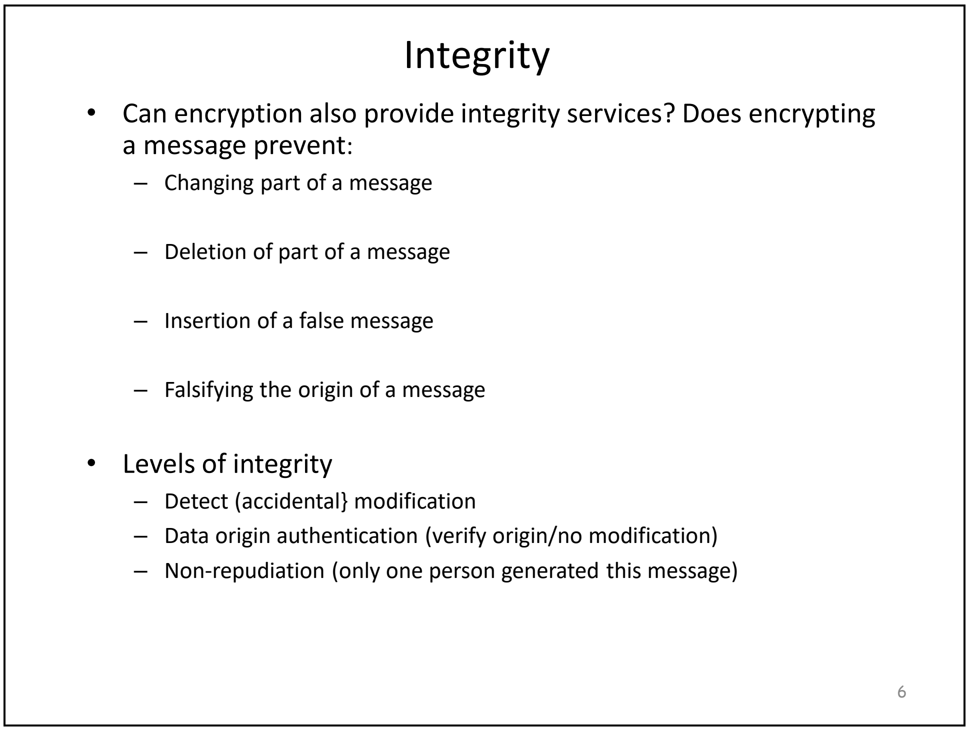
## Today's Lecture

- Data integrity (and non-repudation)
  - Digital Signatures
  - Hash function
  - Message Authentication Codes (MACs)
- CILO2 and CILO5

  (technology that impact systems, and security mechanisms)

4

## Digital Signature
## Hash Function
## Message Authentication Code



## Integrity

- Can encryption also provide integrity services? Does encrypting a message prevent:
  - Changing part of a message
  - Deletion of part of a message
  - Insertion of a false message
  - Falsifying the origin of a message
- Levels of integrity
  - Detect (accidental} modification
  - Data origin authentication (verify origin/no modification)
  - Non-repudiation (only one person generated this message)

6

You must be able to explain why encryption does not prevent data from being modified.

If we encrypt data, the receiver will decrypt it. If someone modified the ciphertext then the plaintext will be wrong – however, the receiver does not know that is it wrong…

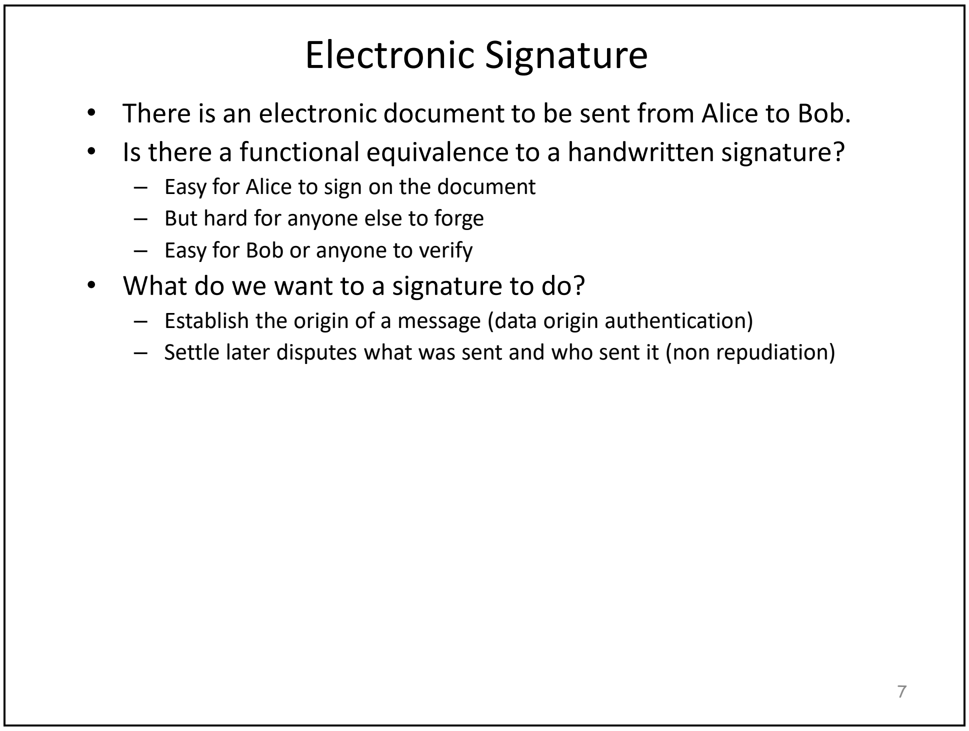
You often hear an argument that says if we decrypt modified data, and it is wrong, then we know it has been modified. That only works if there is a way to see that it is wrong – that is what integrity mechanisms are supposed to do.

Similarly, if we remove or add parts of a message then the receiver does not know there are parts missing or extra (he just has less or more plaintext, which might not make sense but he cannot know it does not make sense)

Also think about the different levels of integrity you would like to provide – and it what practical cases they might be useful.

For example, sending payment information between and ATM and the bank needs only data origin authentication (make sure the message is as it was generates by the ATM). No need for non-repudation as the ATM is not later going to claim it did not send this message (it belongs to the bank)

However, if I would do e-banking and I would like to instruct my bank to make a payment then we might need non-repudiation. The bank would not want me to phone them later and say I did not give this instruction.

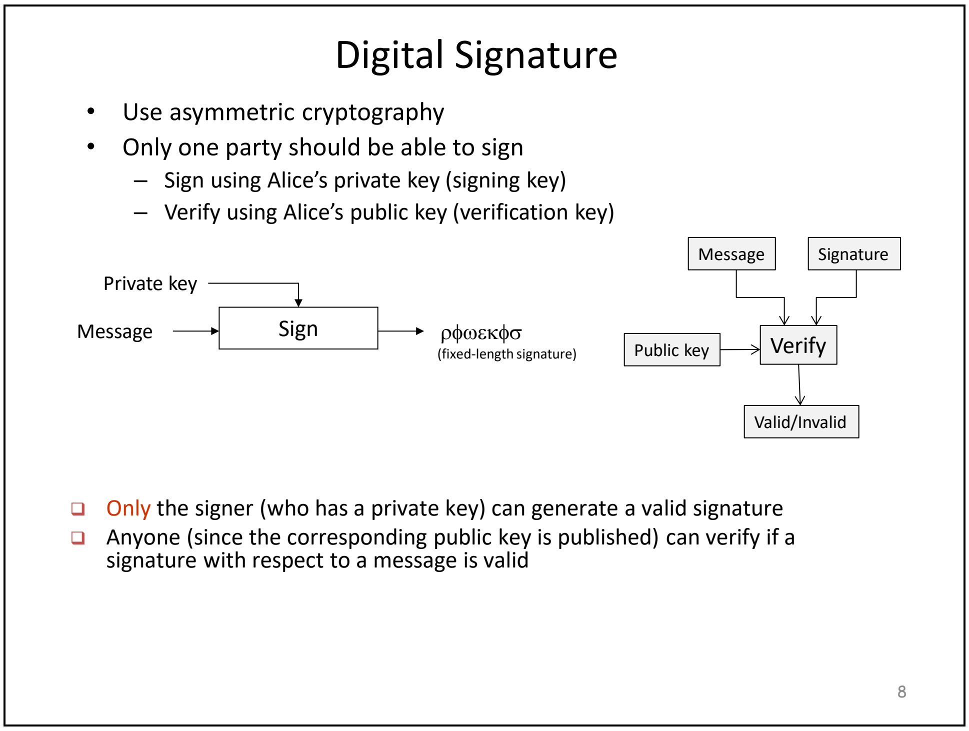


## Electronic Signature

- There is an electronic document to be sent from Alice to Bob.
- Is there a functional equivalence to a handwritten signature?
  - Easy for Alice to sign on the document
  - But hard for anyone else to forge
  - Easy for Bob or anyone to verify
- What do we want to a signature to do?
  - Establish the origin of a message (data origin authentication)
  - Settle later disputes what was sent and who sent it (non repudiation)

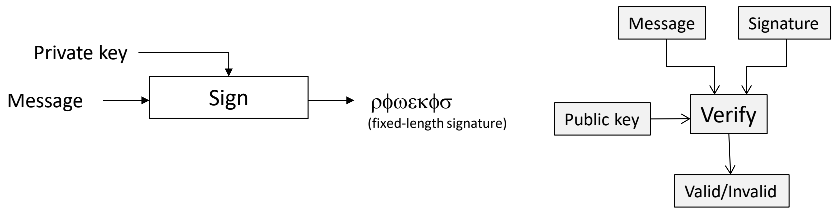
7

Study this slide – be familiar with the properties (i.e. easy to sign, difficult to forge, easy to verify) and the security goals (origin authentication and non-repudiation)



## Digital Signature

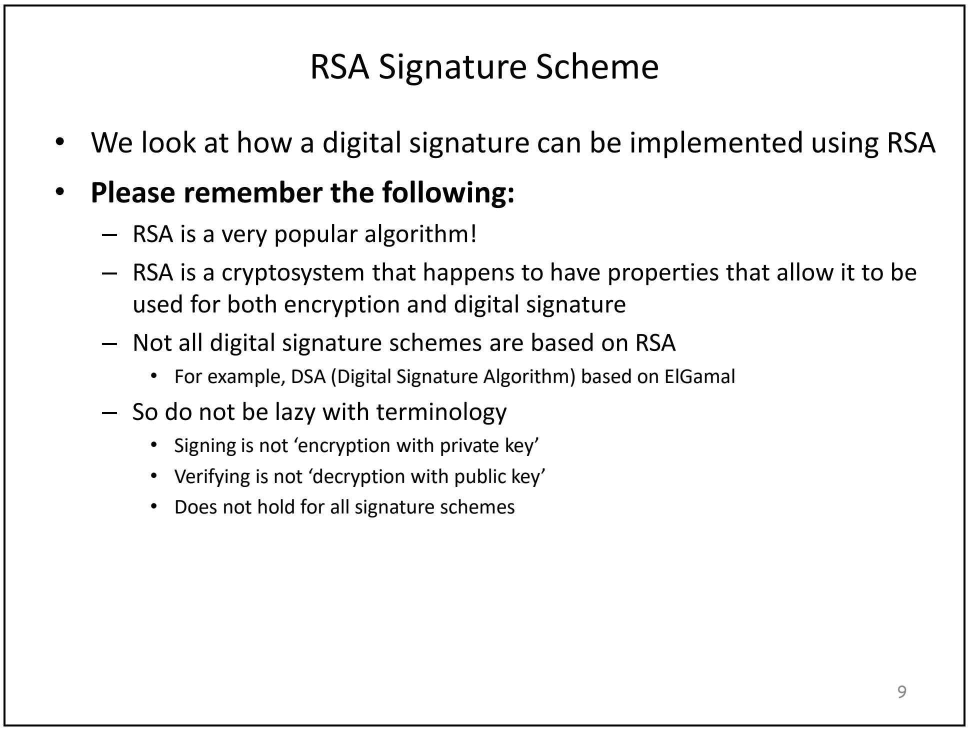
- Use asymmetric cryptography
- Only one party should be able to sign
  - Sign using Alice's private key (signing key)
  - Verify using Alice's public key (verification key)



- ❑ Only the signer (who has a private key) can generate a valid signature
- ❑ Anyone (since the corresponding public key is published) can verify if a signature with respect to a message is valid

8

Study

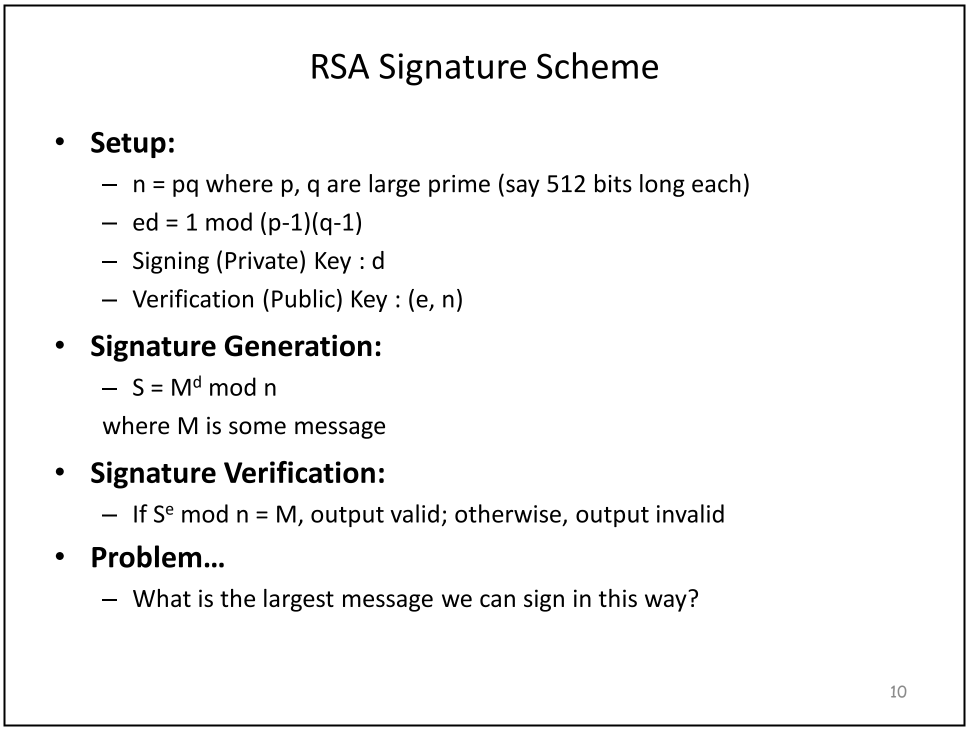


## RSA Signature Scheme

- We look at how a digital signature can be implemented using RSA
- **Please remember the following:**
  - RSA is a very popular algorithm!
  - RSA is a cryptosystem that happens to have properties that allow it to be used for both encryption and digital signature
  - Not all digital signature schemes are based on RSA
    - For example, DSA (Digital Signature Algorithm) based on ElGamal
  - So do not be lazy with terminology
    - Signing is not 'encryption with private key'
    - Verifying is not 'decryption with public key'
    - Does not hold for all signature schemes

9

This is for your interest (but please do not use the terminology that this slides says not to use – if you say in the exam that I sign by encrypting data with private key I will not give you a mark)



## RSA Signature Scheme

- **Setup:**
  - $n = pq$ where p, q are large prime (say 512 bits long each)
  - $ed = 1 \bmod (p-1)(q-1)$
  - Signing (Private) Key : d
  - Verification (Public) Key : (e, n)
- **Signature Generation:**
  - $S = M^d \bmod n$

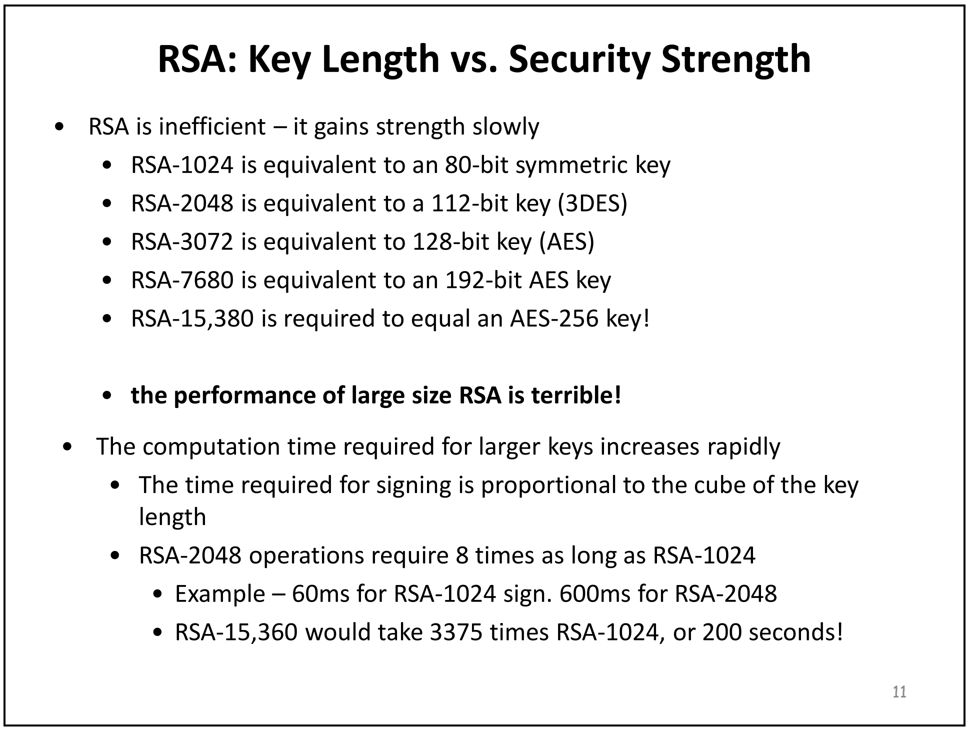
  where M is some message
- **Signature Verification:**
  - If $S^e \bmod n = M$, output valid; otherwise, output invalid
- **Problem…**
  - What is the largest message we can sign in this way?

10

Study – you should already be familiar with RSA from PKE lecture – it is simply a matter of remembering whether the private or public component is used for signing/verification.

The largest message we can sign, is the equivalent question to he largest message we can encrypt with RSA (message must be smaller than n).

Can we break the message into bits and sign each on individually? No, public key crypto is too slow!



## RSA: Key Length vs. Security Strength

- RSA is inefficient – it gains strength slowly
  - RSA-1024 is equivalent to an 80-bit symmetric key
  - RSA-2048 is equivalent to a 112-bit key (3DES)
  - RSA-3072 is equivalent to 128-bit key (AES)
  - RSA-7680 is equivalent to an 192-bit AES key
  - RSA-15,380 is required to equal an AES-256 key!

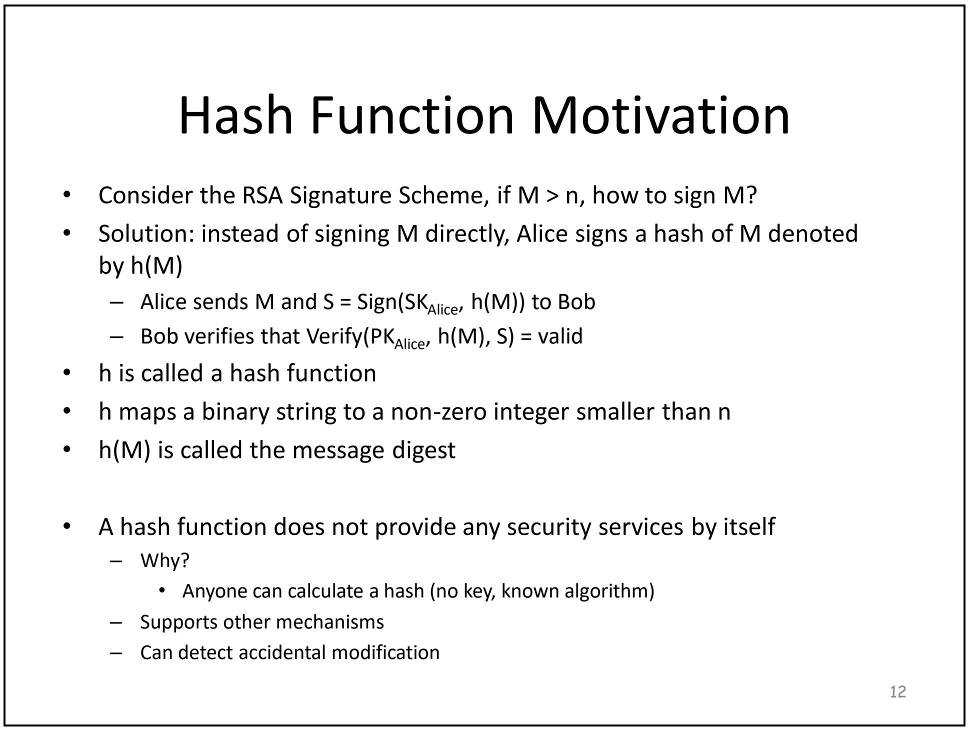
  - **the performance of large size RSA is terrible!**
- The computation time required for larger keys increases rapidly
  - The time required for signing is proportional to the cube of the key length
  - RSA-2048 operations require 8 times as long as RSA-1024
    - Example – 60ms for RSA-1024 sign. 600ms for RSA-2048
    - RSA-15,360 would take 3375 times RSA-1024, or 200 seconds!

11

No need to know the exact numbers – but read slide 11 and 12 together to understand why we need a hash.

What I do expect you to know is that you need a long n to be secure, and computing RSA with long n is slow. So we cannot split up large message into blocks <n or just make n as large as the message (of course there are other issues if you take the second approach like finding really large primes, etc.)

So we need to make a digest (short reduction) of the message first and then sign that…which means we need a hash…



## Hash Function Motivation

- Consider the RSA Signature Scheme, if M > n, how to sign M?
- Solution: instead of signing M directly, Alice signs a hash of M denoted by h(M)
  - Alice sends M and S = Sign($SK_{Alice}$, h(M)) to Bob
  - Bob verifies that Verify($PK_{Alice}$, h(M), S) = valid
- h is called a hash function
- h maps a binary string to a non-zero integer smaller than n
- h(M) is called the message digest

- A hash function does not provide any security services by itself
  - Why?
    - Anyone can calculate a hash (no key, known algorithm)
  - Supports other mechanisms
  - Can detect accidental modification

12

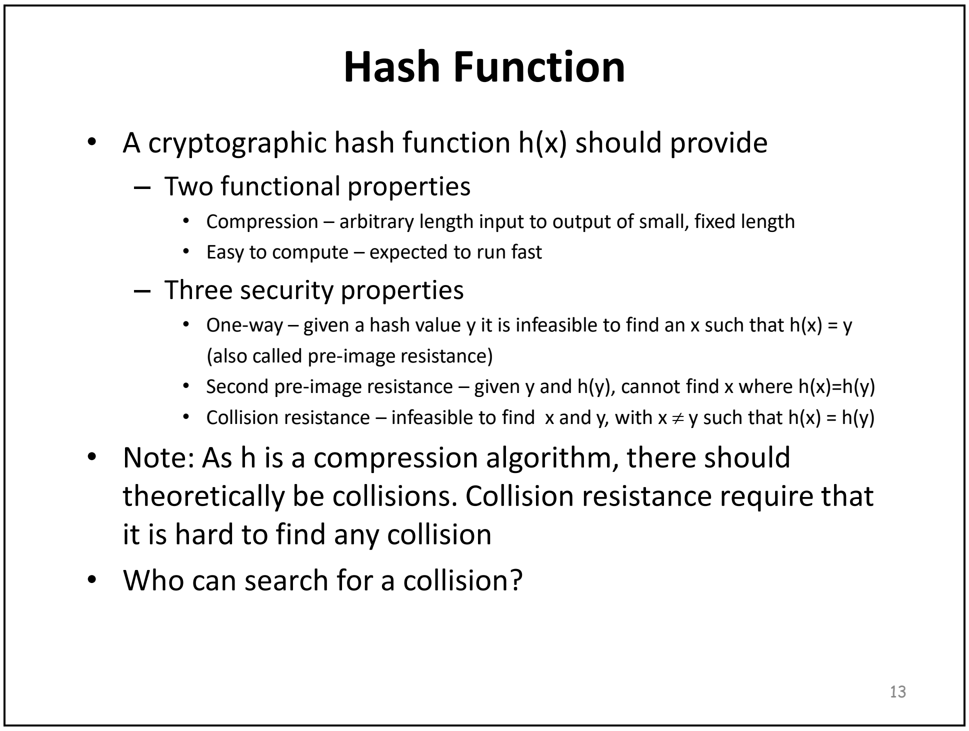
You must be able to explain why a hash is useful (also see previous slide).

You must also realise that a hash does not by itself provide any security service.

1.Anyone can generate it as no key (so attacker can change message, update hash)

2.It only supports others (signature, HMAC)

3.Hash is often used to validate downloads, this is not secure! Anyone can update the file and then calculate a new hash.



## Hash Function

- A cryptographic hash function h(x) should provide
  - Two functional properties
    - Compression – arbitrary length input to output of small, fixed length
    - Easy to compute – expected to run fast
  - Three security properties
    - One-way – given a hash value y it is infeasible to find an x such that h(x) = y (also called pre-image resistance)
    - Second pre-image resistance – given y and h(y), cannot find x where h(x)=h(y)
    - Collision resistance – infeasible to find x and y, with $x \neq y$ such that h(x) = h(y)
- Note: As h is a compression algorithm, there should theoretically be collisions. Collision resistance require that it is hard to find any collision
- Who can search for a collision?

13

Very important slide! You must know the functional and security properties.

Understand the difference between second pre-image resistance and general collision resistance.

Given a message and its hash you cannot find another message with same hash (this is second pre-image resistance)

For collision resistance you are not given any message – you can choose any two messages you want but they must have same hash value.

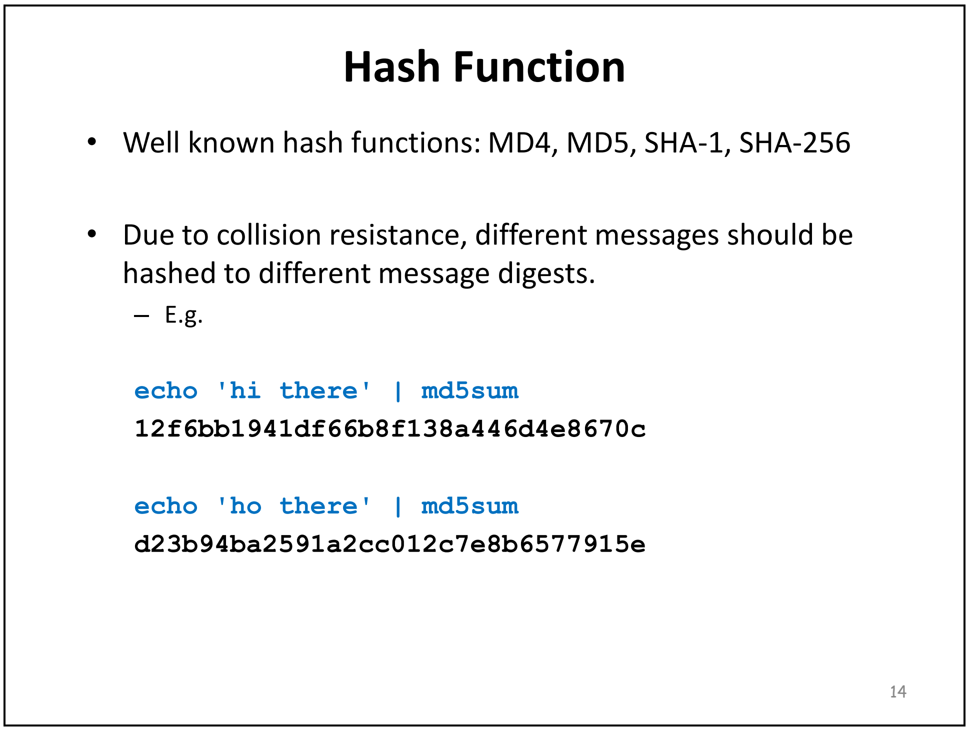
Realise that if you take a message of length x (x>n) and put it into a hash that maps it to a value of length n then you will always have mathematical probability of collisions – it must just be infeasible to compute these.

**Which of these security properties is strongest? i.e. the most demanding property to satisfy?**

**One-way is the simplest, collision resistance is the hardest to**

implement.

If you think about it the other way finding a collision is easier for an attacker than breaking the one-wayness - finding *the* input that gives a specific output.



## Hash Function

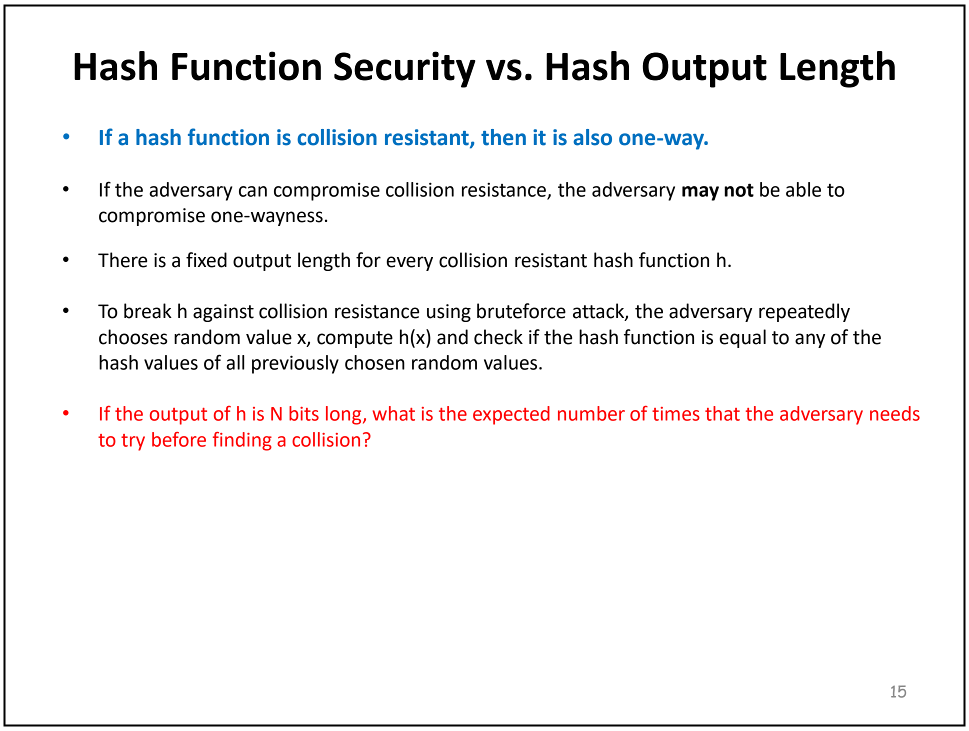
- Well known hash functions: MD4, MD5, SHA-1, SHA-256
- Due to collision resistance, different messages should be hashed to different message digests.
  - E.g.

```
echo 'hi there' | md5sum
12f6bb1941df66b8f138a446d4e8670c

echo 'ho there' | md5sum
d23b94ba2591a2cc012c7e8b6577915e
```

14

You need to understand that a good hash will give a completely different result even if very small part of the input changes.



## Hash Function Security vs. Hash Output Length

- **If a hash function is collision resistant, then it is also one-way.**
- If the adversary can compromise collision resistance, the adversary **may not** be able to compromise one-wayness.
- There is a fixed output length for every collision resistant hash function h.
- To break h against collision resistance using bruteforce attack, the adversary repeatedly chooses random value x, compute h(x) and check if the hash function is equal to any of the hash values of all previously chosen random values.
- If the output of h is N bits long, what is the expected number of times that the adversary needs to try before finding a collision?

15

Read 15-19 together. You must know basic idea of how to evaluate the collision resistance of a hash.

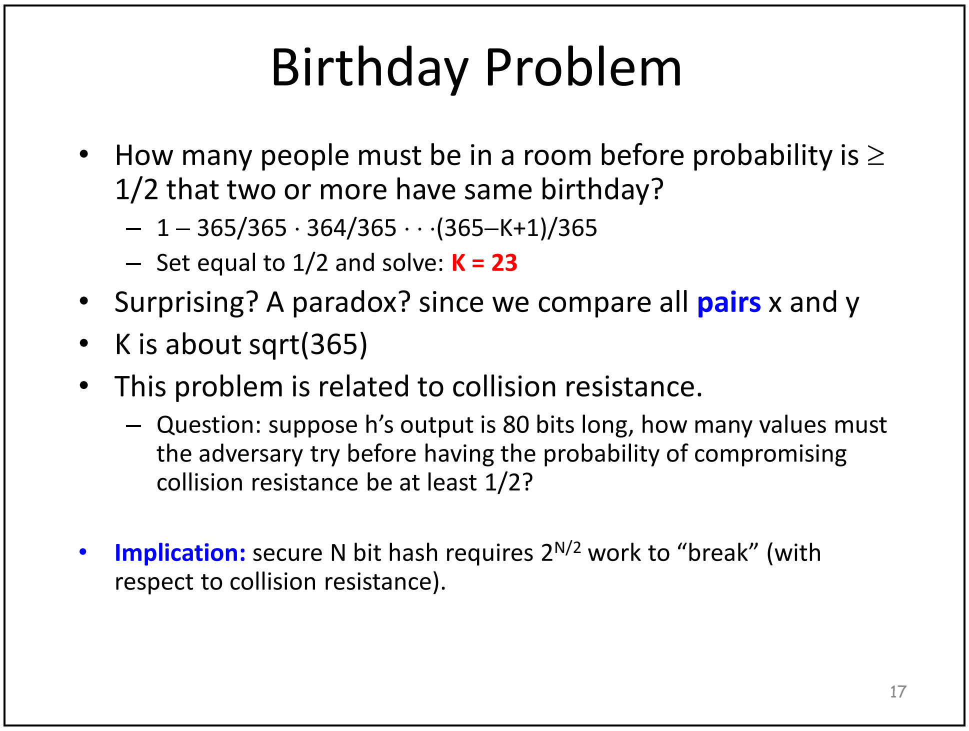
You must know when the birthday problem is relevant and what the resultant computational requirement to find a collision is 2^(n/2)

Do not confuse this with brute forcing keys, or even finding second pre-image collisions. If you are given x and h(x) and you need to find y so h(y)=h(x) then birthday problem not relevant.

## Pre-Birthday Problem

- Suppose K people in a room
- How large must K be before the probability that someone has the same birthday as me is $\geq 1/2$
  - Solve: $1/2 = 1 - (364/365)^K$ for K
  - Find K = 253
- This problem is related to collision resistance but not the same.

16



## Birthday Problem

- How many people must be in a room before probability is ≥ 1/2 that two or more have same birthday?
  - 1 – 365/365 · 364/365 · · ·(365–K+1)/365
  - Set equal to 1/2 and solve: **K = 23**
- Surprising? A paradox? since we compare all **pairs** x and y
- K is about sqrt(365)
- This problem is related to collision resistance.
  - Question: suppose h's output is 80 bits long, how many values must the adversary try before having the probability of compromising collision resistance be at least 1/2?

- **Implication:** secure N bit hash requires $2^{N/2}$ work to "break" (with respect to collision resistance).

17

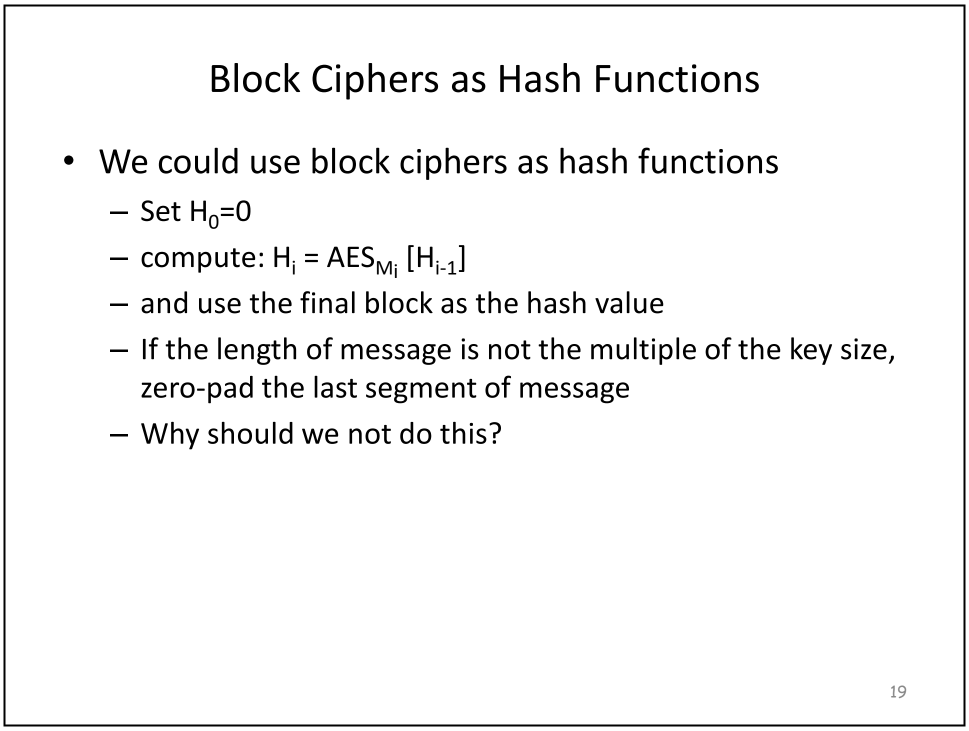
You do not need to study the probability maths behind this idea, just remember the final implication and where it is applied.

Total space of 2^80, sqrt(2^80)=2^40

## Bruteforce Attack Against the Collision-resistance of a Hash Function

- Finding collisions of a hash function using Birthday Paradox.
  1. randomly chooses K messages, $m_1$, $m_2$, …, $m_k$
  2. search if there is a pair of messages, say $m_i$ and $m_j$ such that
     $h(m_i) = h(m_j)$.

     If so, one collision is found.
- This birthday attack imposes a lower bound on the size of message digests.
- E.g. 10-bit message digest is very insecure, since one collision can be found with probability at least 0.5 after doing slightly over $2^5$ (i.e. 32) random hashes.
- E.g. 40-bit message digest is also insecure, since a collision can be found with probability at least 0.5 after doing slightly over $2^{20}$ (about a million) random hashes.



## Block Ciphers as Hash Functions

- We could use block ciphers as hash functions
  - Set $H_0$=0
  - compute: $H_i = AES_{M_i} [H_{i-1}]$
  - and use the final block as the hash value
  - If the length of message is not the multiple of the key size, zero-pad the last segment of message
  - Why should we not do this?

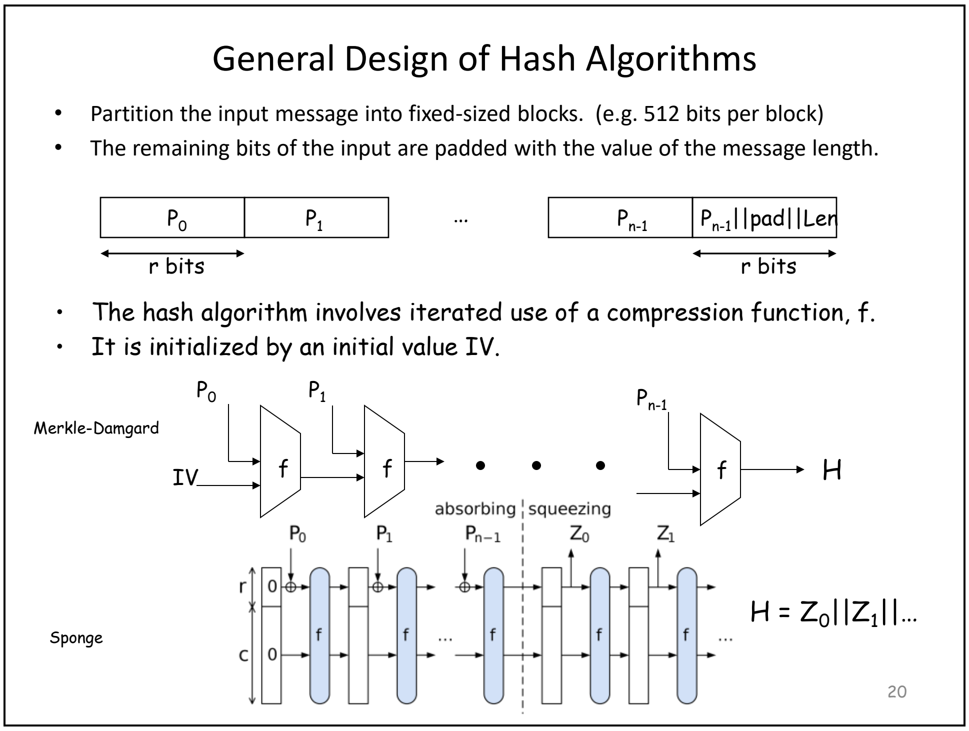
19

If you use a block cipher as a has then Hash length = block size…

Generally block sizes are not that large – block cipher security is more proportional to key length. Even AES has blocksize 128 bits.

This is much shorter than most of you hash function lengths (generally 160 and up 256/384/512).

If we did a block cipher hash the birthday problem would mean that finding collisions are much easier than for normal hash functions.



## General Design of Hash Algorithms

- Partition the input message into fixed-sized blocks. (e.g. 512 bits per block)
- The remaining bits of the input are padded with the value of the message length.

$P_0$
$P_1$
…
$P_{n-1}$
$P_{n-1}$||pad||Len
r bits
r bits

- The hash algorithm involves iterated use of a compression function, f.
- It is initialized by an initial value IV.

Merkle-Damgard
$P_0$
$P_1$
$P_{n-1}$
IV
f
f
f
H
absorbing
squeezing
$P_0$
$P_1$
$P_{n-1}$
$Z_0$
$Z_1$
r
c
0
0
f
Sponge
H = $Z_0$||$Z_1$||…
20

You must be able to explain in general approaches to designing a hash function.

Take message and split into input block, pad last block if message if not a multiple of the input length.

Merkle-Damgard (SHA-1/SHA-2)

We first calculate the hash if IV and M1 to give us intermediate hash value H1 (f being a compression function). We then use H1 and M2 to generate H2, and so we continue….H_n=h(H_(n-1),M_n)

The final hash function output is then used as the hash value.

Sponge (SHA-3)

Internal state of length r+c bits.

Aborbing

r-bits input each round, PRF f mix state.

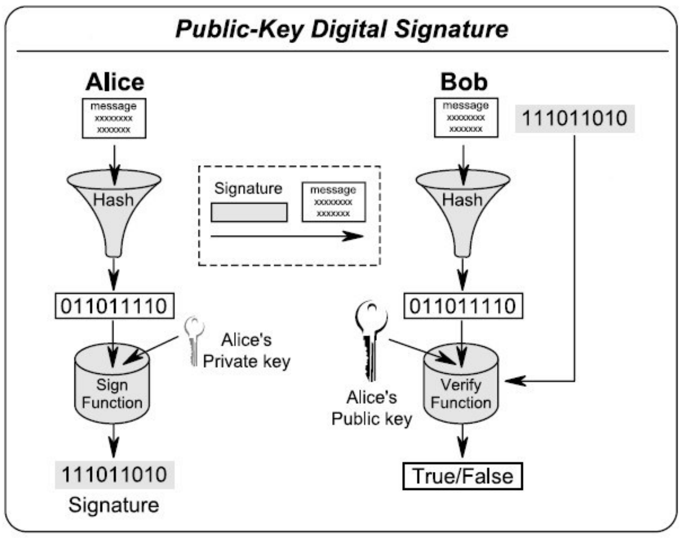
Squeezing

Take out current state r (then run though PRF until output Z is desired length).

Sponge is very flexible in performance – can adjust input block length (r) and output length.

Internal state is larger than output (believed better security)

What is the main application of cryptographic hash functions?

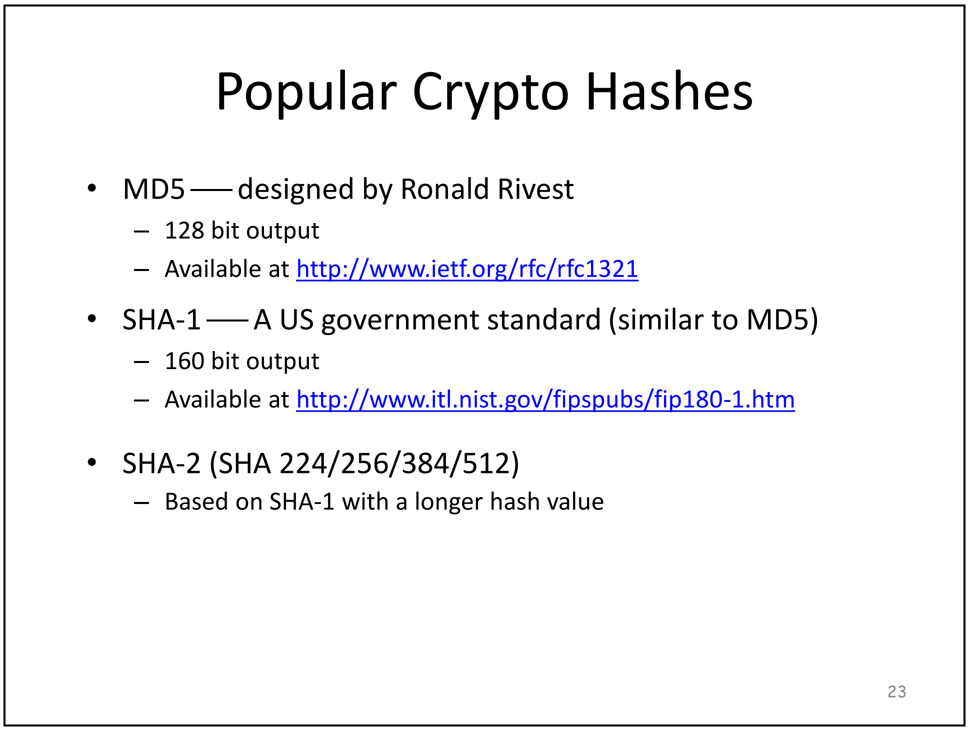


You must be able to explain how a hash is used in digital signature scheme.

The signer will calculate the hash of the message and then in the case of RSA signs the hash (not the message). All signature schemes use the hash in the signature generation.

Upon receiving the message and signature the receiver will also compute the hash of the received message. This hash is then used during signature verification.

If we were using RSA, the signature verification with the public key will result in the hash the sender calculated. We then compare this verification hash with the has we calculated over the received message and if it they are the same we consider the signature valid.

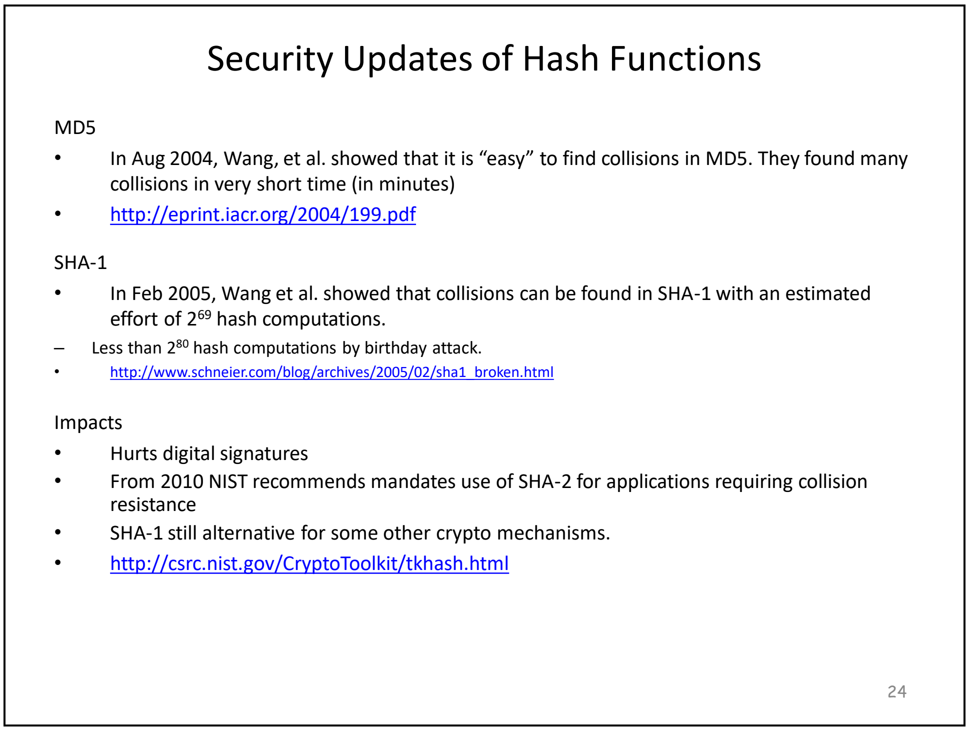


## Popular Crypto Hashes

- MD5 —— designed by Ronald Rivest
  - 128 bit output
  - Available at http://www.ietf.org/rfc/rfc1321
- SHA-1 —— A US government standard (similar to MD5)
  - 160 bit output
  - Available at http://www.itl.nist.gov/fipspubs/fip180-1.htm
- SHA-2 (SHA 224/256/384/512)
  - Based on SHA-1 with a longer hash value

23

For interest – however, you should at least be able to name a hash algorithm and generally be aware of their lengths.



## Security Updates of Hash Functions

MD5

- In Aug 2004, Wang, et al. showed that it is “easy” to find collisions in MD5. They found many collisions in very short time (in minutes)
- http://eprint.iacr.org/2004/199.pdf

SHA-1

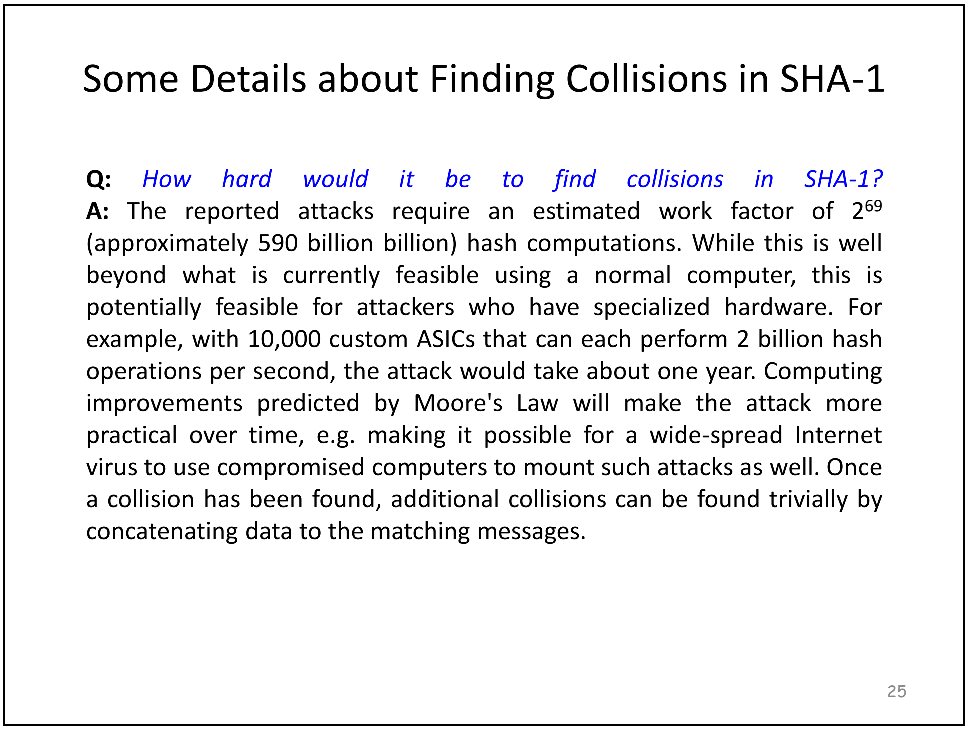
- In Feb 2005, Wang et al. showed that collisions can be found in SHA-1 with an estimated effort of $2^{69}$ hash computations.
  – Less than $2^{80}$ hash computations by birthday attack.
- http://www.schneier.com/blog/archives/2005/02/sha1_broken.html

Impacts

- Hurts digital signatures
- From 2010 NIST recommends mandates use of SHA-2 for applications requiring collision resistance
- SHA-1 still alternative for some other crypto mechanisms.
- http://csrc.nist.gov/CryptoToolkit/tkhash.html

24

For interest



## Some Details about Finding Collisions in SHA-1

**Q:** *How hard would it be to find collisions in SHA-1?*
**A:** The reported attacks require an estimated work factor of $2^{69}$ (approximately 590 billion billion) hash computations. While this is well beyond what is currently feasible using a normal computer, this is potentially feasible for attackers who have specialized hardware. For example, with 10,000 custom ASICs that can each perform 2 billion hash operations per second, the attack would take about one year. Computing improvements predicted by Moore's Law will make the attack more practical over time, e.g. making it possible for a wide-spread Internet virus to use compromised computers to mount such attacks as well. Once a collision has been found, additional collisions can be found trivially by concatenating data to the matching messages.

25

For interest

Why are hash collisions bad? Signatures

Is the impact real?

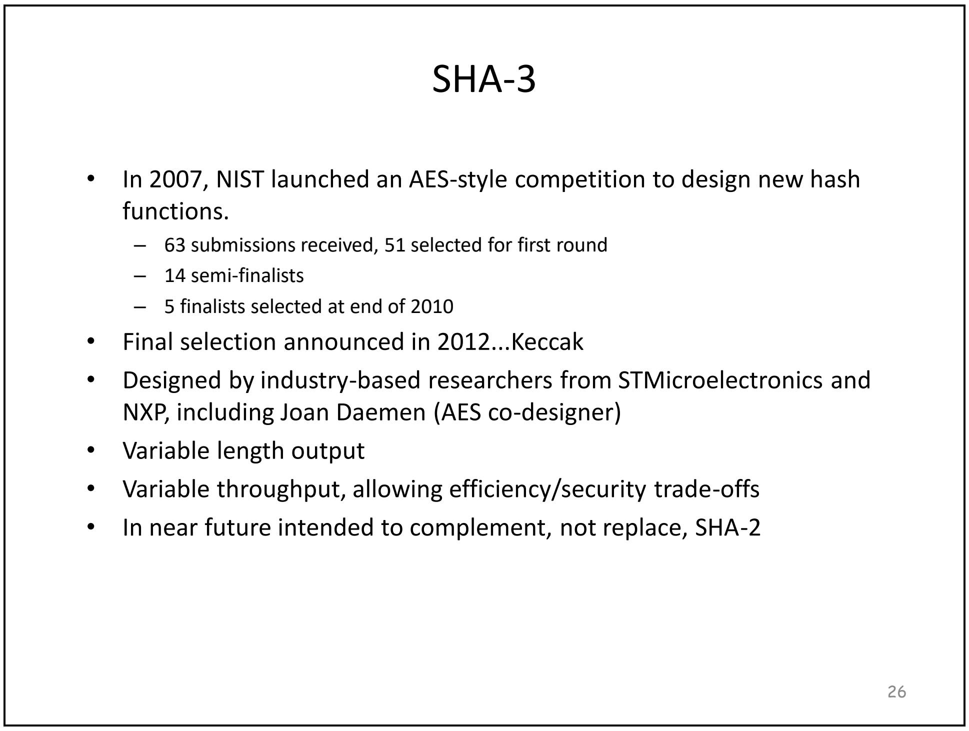
See speed camera story in extra reading

About 50 times more processing power than 25 modern GPU cards. 1250 modern GPU, can do in a year.

As, example NVidia V100 (90 Mhash/s)

Total bitcoin system 2^65 hash per second. Every single miner worked on collision then couple of seconds.

Bitcoin used Sha-256, and not looking for collisions but for specific answer (more a pre-image problem).



## SHA-3

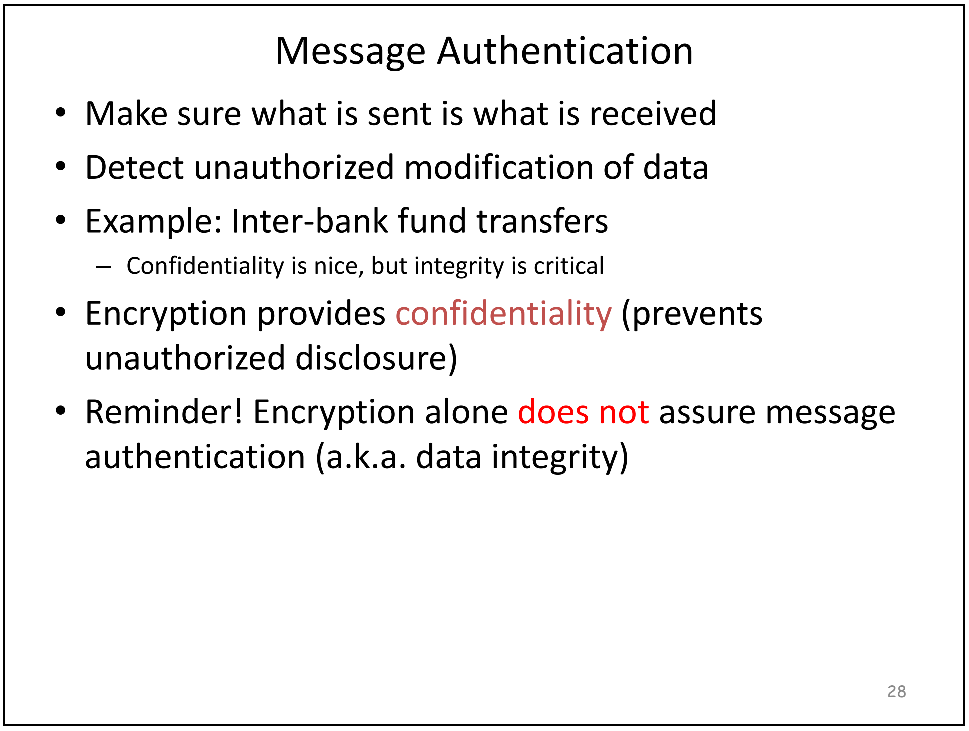
- In 2007, NIST launched an AES-style competition to design new hash functions.
  - 63 submissions received, 51 selected for first round
  - 14 semi-finalists
  - 5 finalists selected at end of 2010
- Final selection announced in 2012...Keccak
- Designed by industry-based researchers from STMicroelectronics and NXP, including Joan Daemen (AES co-designer)
- Variable length output
- Variable throughput, allowing efficiency/security trade-offs
- In near future intended to complement, not replace, SHA-2

26

For interest

## Message Authentication

27

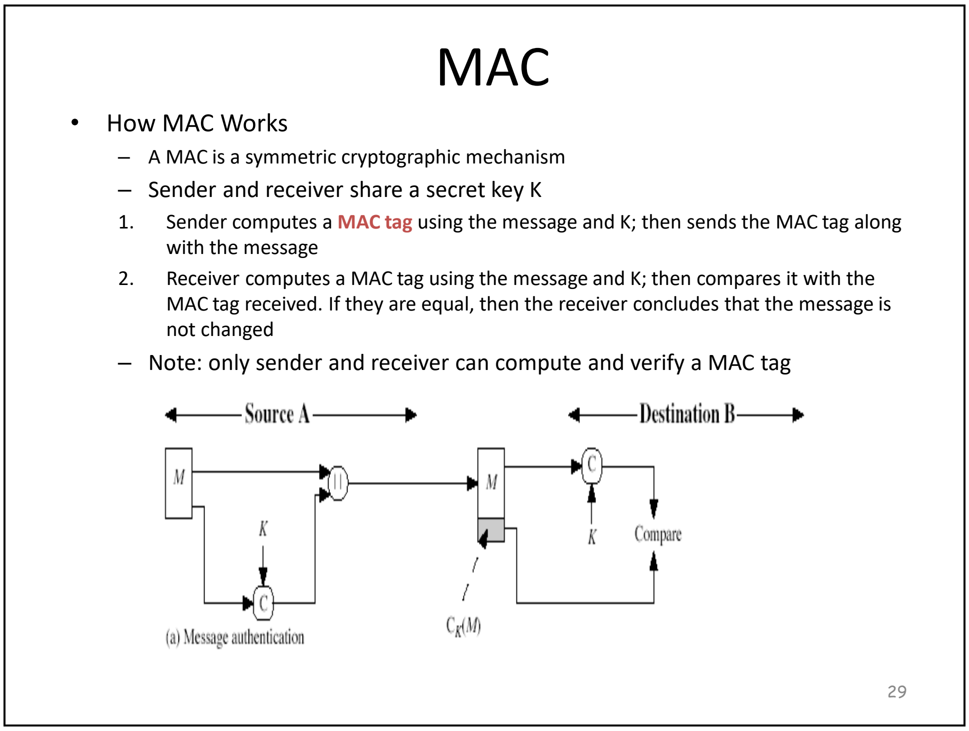


## Message Authentication

- Make sure what is sent is what is received
- Detect unauthorized modification of data
- Example: Inter-bank fund transfers
  - Confidentiality is nice, but integrity is critical
- Encryption provides confidentiality (prevents unauthorized disclosure)
- Reminder! Encryption alone does not assure message authentication (a.k.a. data integrity)

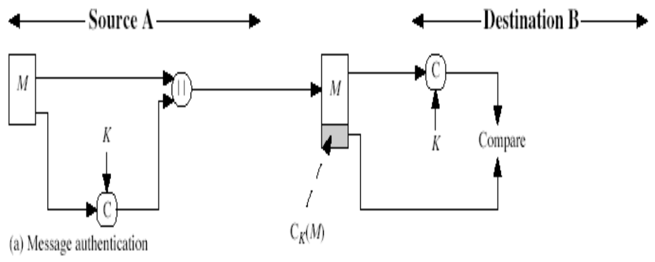
28

This slide revises some comments from start of the lecture. You must be able to give me an example where a MAC is useful (where we only need data origin authentication)



## MAC

- How MAC Works
  - A MAC is a symmetric cryptographic mechanism
  - Sender and receiver share a secret key K
  1. Sender computes a **MAC tag** using the message and K; then sends the MAC tag along with the message
  2. Receiver computes a MAC tag using the message and K; then compares it with the MAC tag received. If they are equal, then the receiver concludes that the message is not changed
  - Note: only sender and receiver can compute and verify a MAC tag

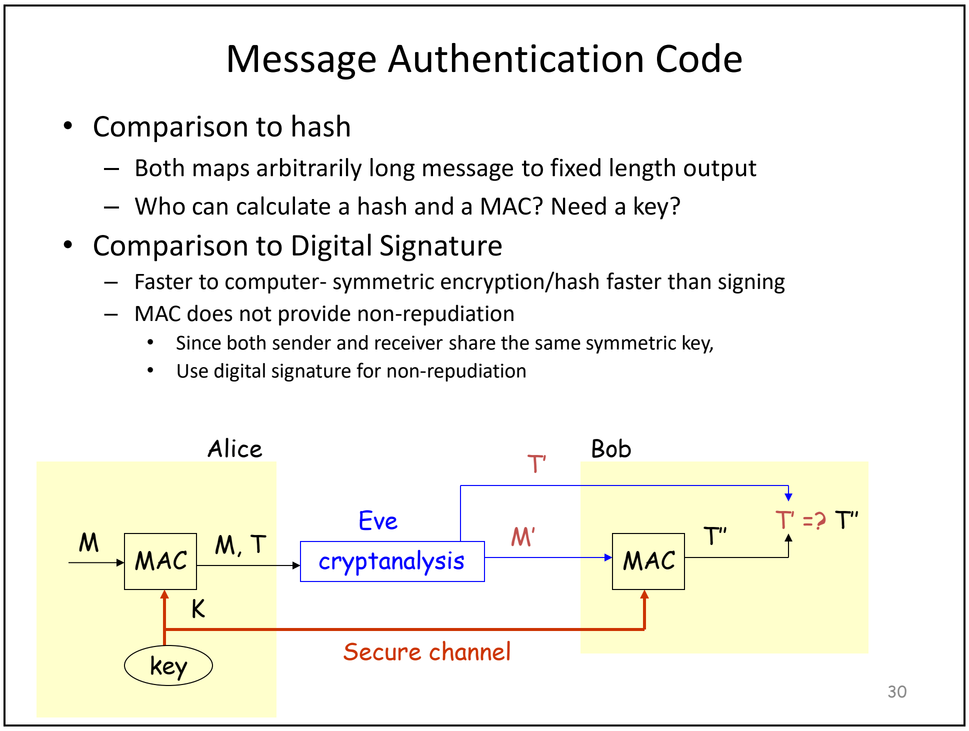


(a) Message authentication

29

Study – a MAC is a little like a symmetric key 'signature' – it is used in the same way.

We calculate the MAC using a key K. We then send the message and the MAC. The receiver takes the received message and also calculates its MAC (since it has the same key K). It the compares it to the sent MAC. If they are the same then message is unmodified.



## Message Authentication Code

- Comparison to hash
  - Both maps arbitrarily long message to fixed length output
  - Who can calculate a hash and a MAC? Need a key?
- Comparison to Digital Signature
  - Faster to computer- symmetric encryption/hash faster than signing
  - MAC does not provide non-repudiation
    - Since both sender and receiver share the same symmetric key,
    - Use digital signature for non-repudiation

Study this slide.

Why do we not worry about collisions on a MAC?

Attackers can search for collisions as they would need the secret key to do so.

Why does not MAC provide non-repudiation?

Non repudiation needs there to be undeniable proof that one person generated the message. In the case of a signature only one person has the private key so only one person can generate the signature.

In MAC two people have the key needed to generate the MAC (it uses a symmetric key). This means the receiver could modify the message and generate a new valid MAC (or the receiver could claim that it is possible and deny that he sent a message).

## A MAC Algorithm

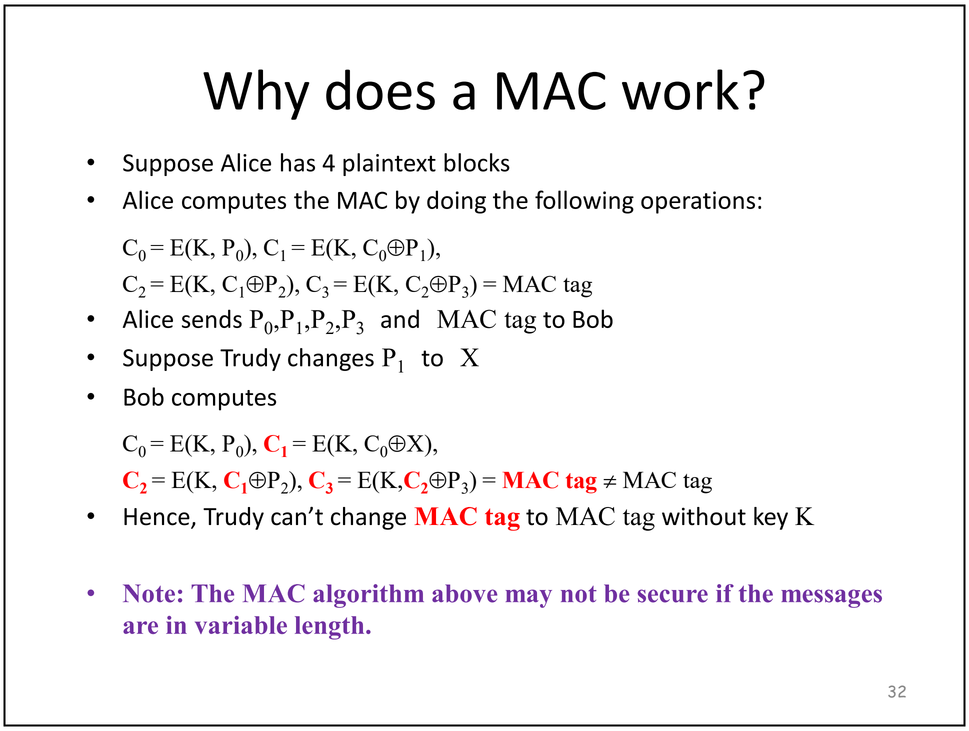
- MAC can be constructed from a block cipher operated in CBC mode (with IV=0).
- Suppose a plaintext has 4 plaintext blocks $P=P_0, P_1, P_2, P_3$
- Suppose K is the secret key shared between sender and receiver.

$C_0 = E(K, P_0)$,

$C_1 = E(K, C_0 \oplus P_1)$,

$C_2 = E(K, C_1 \oplus P_2)$,…

$C_{N-1} = E(K, C_{N-2} \oplus P_{N-1})$ = MAC tag



## Why does a MAC work?

- Suppose Alice has 4 plaintext blocks
- Alice computes the MAC by doing the following operations:

  $C_0 = E(K, P_0)$, $C_1 = E(K, C_0 \oplus P_1)$,
  $C_2 = E(K, C_1 \oplus P_2)$, $C_3 = E(K, C_2 \oplus P_3) = \text{MAC tag}$
- Alice sends $P_0, P_1, P_2, P_3$ and MAC tag to Bob
- Suppose Trudy changes $P_1$ to $X$
- Bob computes

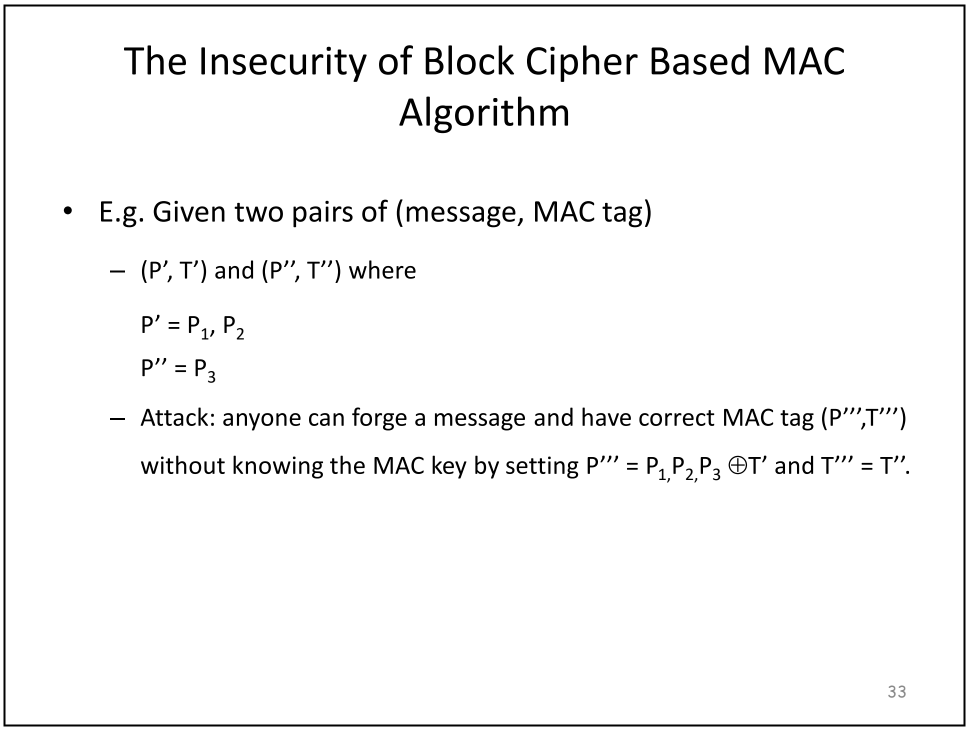
  $C_0 = E(K, P_0)$, $\mathbf{C_1} = E(K, C_0 \oplus X)$,
  $\mathbf{C_2} = E(K, \mathbf{C_1} \oplus P_2)$, $\mathbf{C_3} = E(K, \mathbf{C_2} \oplus P_3) = \mathbf{MAC\ tag} \neq \text{MAC tag}$
- Hence, Trudy can't change **MAC tag** to MAC tag without key K

- **Note: The MAC algorithm above may not be secure if the messages are in variable length.**

32

Study

The most common MAC is CBC-MAC – essentially we 'encrypt' the message using CBC mode but we throw all the ciphertext blocks away apart from the last one. The last one is our MAC.


## The Insecurity of Block Cipher Based MAC Algorithm

- E.g. Given two pairs of (message, MAC tag)
  - (P', T') and (P'', T'') where

    $P' = P_1, P_2$

    $P'' = P_3$
  - Attack: anyone can forge a message and have correct MAC tag (P''',T''') without knowing the MAC key by setting $P''' = P_1, P_2, P_3 \oplus T'$ and $T''' = T''$.

33

Look at slides 33 and 34 together – you must understand this attack. If we just used plain CBC mode and the recipient did not know how much data to expect we could stitch some messages together – but still attach a valid MAC without knowing the key (reuse a MAC).

In practice, we modify the CBC-MAC to prevent such attacks (we have some additional initial and final permutations)

## How?

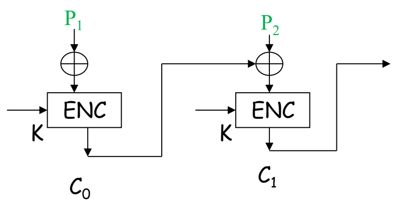
P\`

$C_0 = E(K, IV(00) \oplus P_1)$,

$T' = MAC = C_1 = E(K, C_0 \oplus P_2)$

P\`\`

$T'' = MAC = C_0 = E(K, IV(00) \oplus P_3)$



P\`\`\`

$C_0 = E(K, IV(00) \oplus P_1)$,

$T' = C_1 = E(K, C_0 \oplus P_2)$

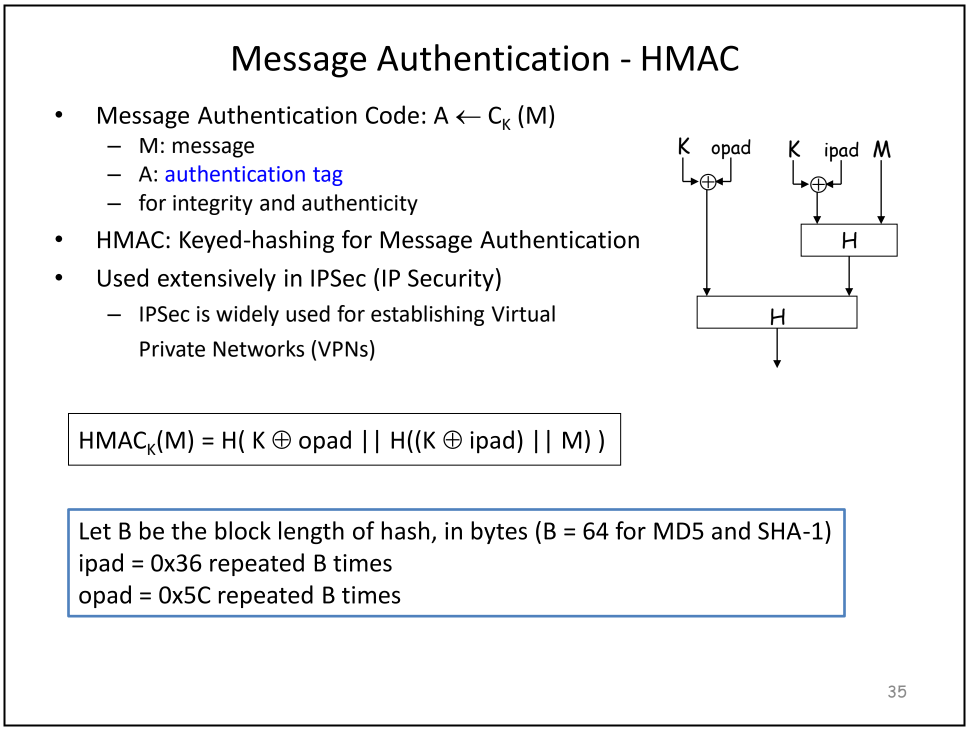
$C_2 = E(K, C_1 \oplus (P_3 \oplus T\`)) = E(K, T' \oplus (P_3 \oplus T\`)) = E(K, P_3)$ since $T' = C_1$

So $T''' = C_2 = T''$

New message is $P_1 P_2, P_3 \oplus T\`$ with valid MAC $T''' = T''$

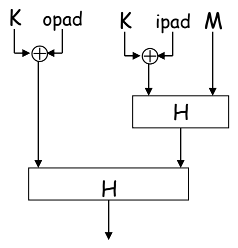
34

Previously people got confused with P1 P2 and P3 (from previous slide) so here we have colour differentiating messages



## Message Authentication - HMAC

- Message Authentication Code: $A \leftarrow C_K(M)$
  - M: message
  - A: authentication tag
  - for integrity and authenticity
- HMAC: Keyed-hashing for Message Authentication
- Used extensively in IPSec (IP Security)
  - IPSec is widely used for establishing Virtual Private Networks (VPNs)



$HMAC_K(M) = H( K \oplus opad \,||\, H((K \oplus ipad) \,||\, M) )$

Let B be the block length of hash, in bytes (B = 64 for MD5 and SHA-1)
ipad = 0x36 repeated B times
opad = 0x5C repeated B times

35

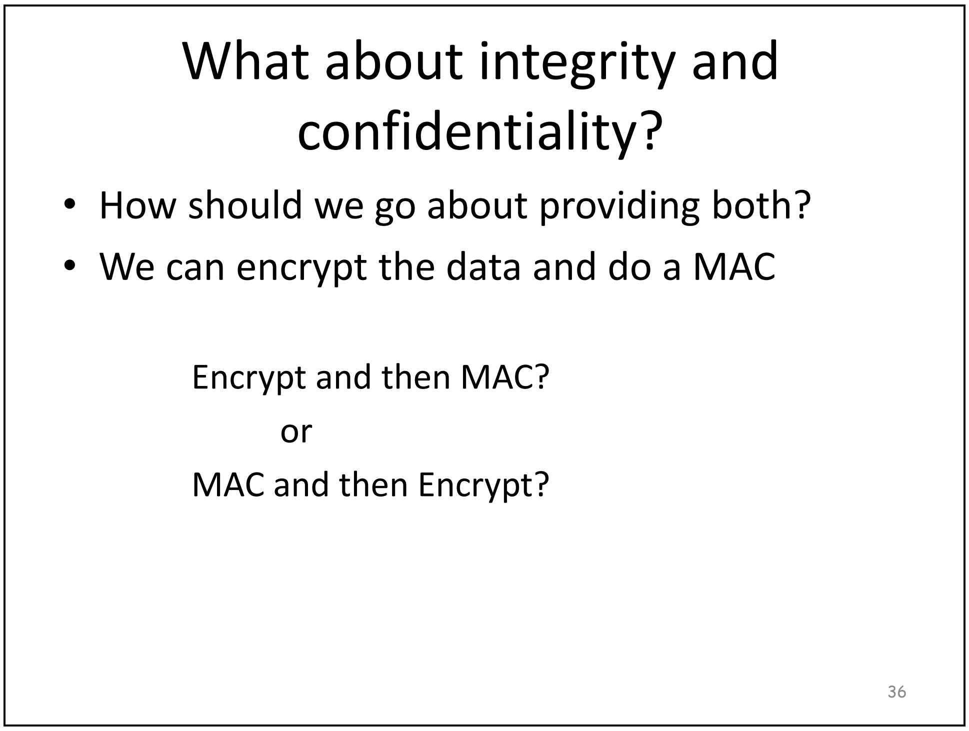
Study

Hash functions are generally seen as being quite efficient and fast - and they also give good output lengths. It has therefore been proposed that we build MAC out of hash rather than symmetric encryption.

All the needs to happen is that we need a way of integrating a secret key into the hash calculation.

$K \oplus opad$ and $K \oplus ipad$ is just a fancy way of saying you should use two different keys (although they could be derived from each other)

$K \oplus ipad$ is added in front of the message, i.e. it is the first message block, and the entire message is then hashed. The final hash output is then hashed together with key $K \oplus opad$.



## What about integrity and confidentiality?

- How should we go about providing both?
- We can encrypt the data and do a MAC

Encrypt and then MAC?
or
MAC and then Encrypt?

36

Interest only

You should think able to think about this as a practical issue.

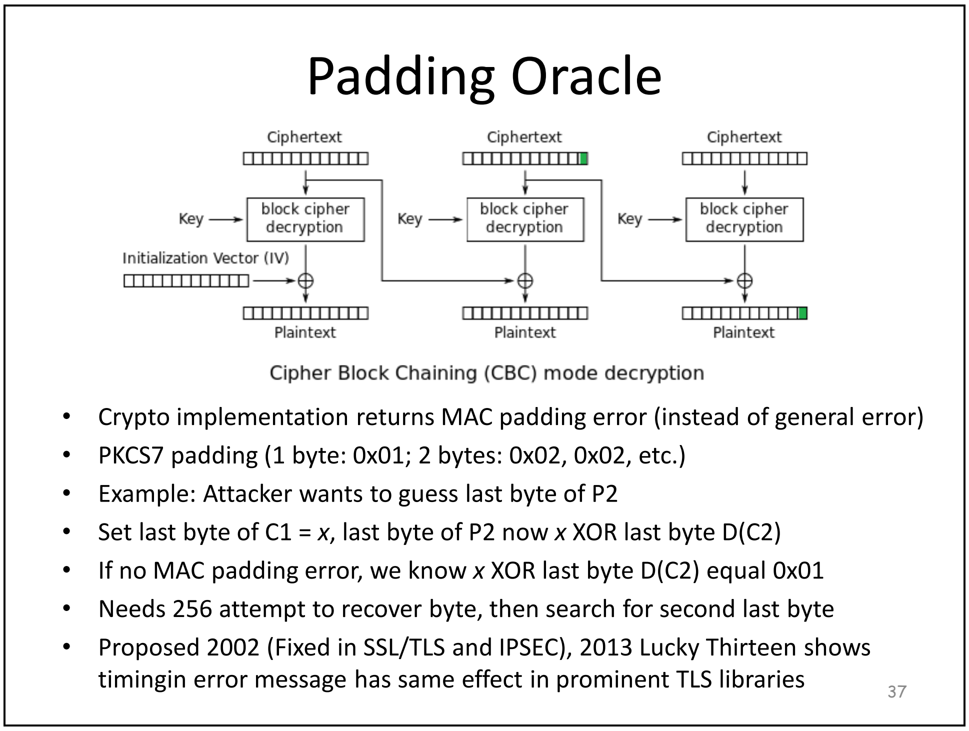
If we encrypt and then MAC then the verification is quite efficient – we check the MAC and if it fails we do not need to decrypt (we know the data is modified).

However, some people argue that the MAC is not truly representative of the plaintext (as it is calculated across the ciphertext message), just of the transmitted message. Once decrypted there is not remaining data origin authentication on the plaintext.

For MAC and encrypt the argument is that the MAC is calculated on the plaintext and then encrypted along with the plaintext. This is argued to be more inline with the purpose of a MAC…

However, if you were interested in modification during transmission then you would have to completely decrypt the data, AND then verify the MAC before knowing if something was modified.

Depending on implementation this could also lead to padding oracle attacks. As such, encrypt and MAC is generally considered more secure in general.



## Padding Oracle

Ciphertext
Ciphertext
Ciphertext
Key
block cipher decryption
Key
block cipher decryption
Key
block cipher decryption
Initialization Vector (IV)
Plaintext
Plaintext
Plaintext

Cipher Block Chaining (CBC) mode decryption

- Crypto implementation returns MAC padding error (instead of general error)
- PKCS7 padding (1 byte: 0x01; 2 bytes: 0x02, 0x02, etc.)
- Example: Attacker wants to guess last byte of P2
- Set last byte of C1 = *x*, last byte of P2 now *x* XOR last byte D(C2)
- If no MAC padding error, we know *x* XOR last byte D(C2) equal 0x01
- Needs 256 attempt to recover byte, then search for second last byte
- Proposed 2002 (Fixed in SSL/TLS and IPSEC), 2013 Lucky Thirteen shows timingin error message has same effect in prominent TLS libraries

37

Interest only

Proposed in 2002.

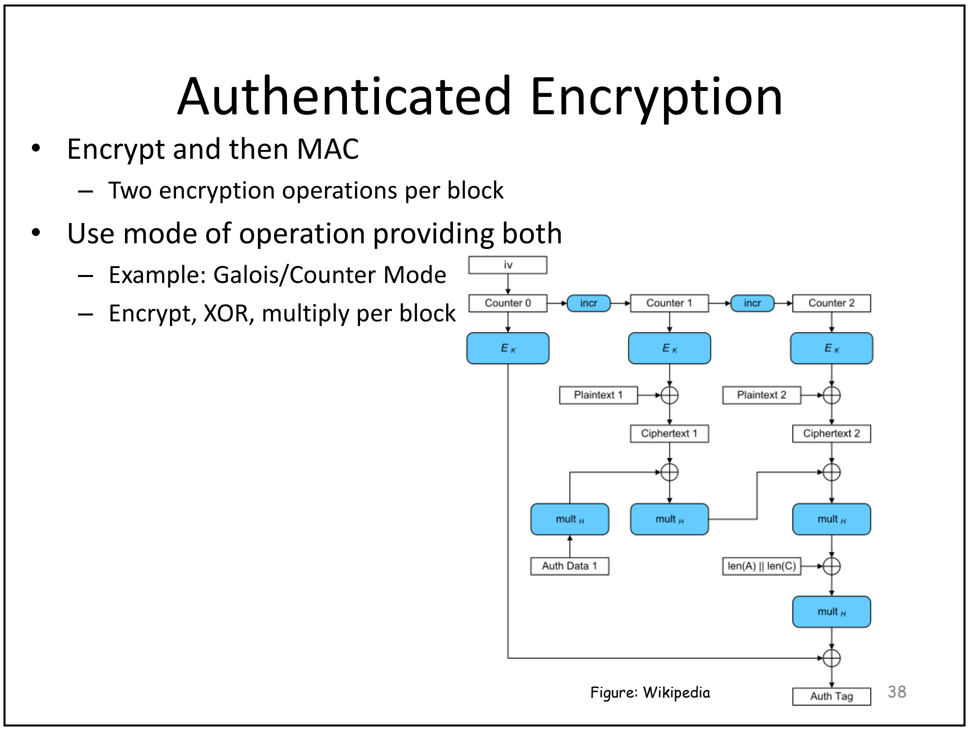
Crypto implementation returned a padding error when the plaintext padding was wrong. Therefore  a legitimate crypto participant could be used as an oracle (someone giving information that allows the attacker the learn something).

In our example:

If the last byte of P2 is 0x01 then the implementation would consider it a valid padding. But last byte of P2 is last byte of C1 XOR to last byte of D(C2). So attacker can set last byte of C1 to any value and send new message C0, C1, C2 to the recipient. If no error comes back attacker knows that last byte of P2 XOR to his changed byte in C1 is equal to 0x01 and he knows true value of P2 last byte. Then attacker can choose last byte of C1 and start guessing second last byte of C1 so that last two bytes of C1 XOR D(C2) is 0x02 0x02. This allows attacker to guess one byte in 256 tries, and the entire plaintext block in 16x256 tries.

More info
https://en.wikipedia.org/wiki/Padding_oracle_attack
https://en.wikipedia.org/wiki/Lucky_Thirteen_attack

PKCS7 padding
1 block 0x01
2 blocks 0x02 0x02
3 blocks 0x03 0x03 0x03
Etc.



## Authenticated Encryption

- Encrypt and then MAC
  - Two encryption operations per block
- Use mode of operation providing both
  - Example: Galois/Counter Mode
  - Encrypt, XOR, multiply per block

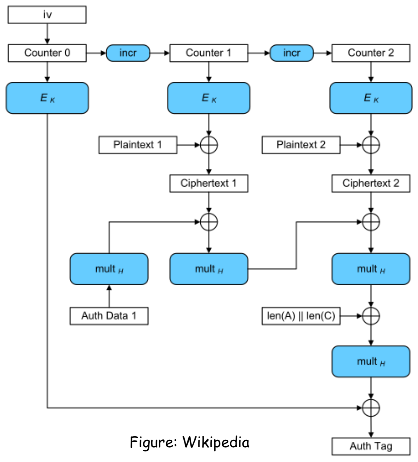
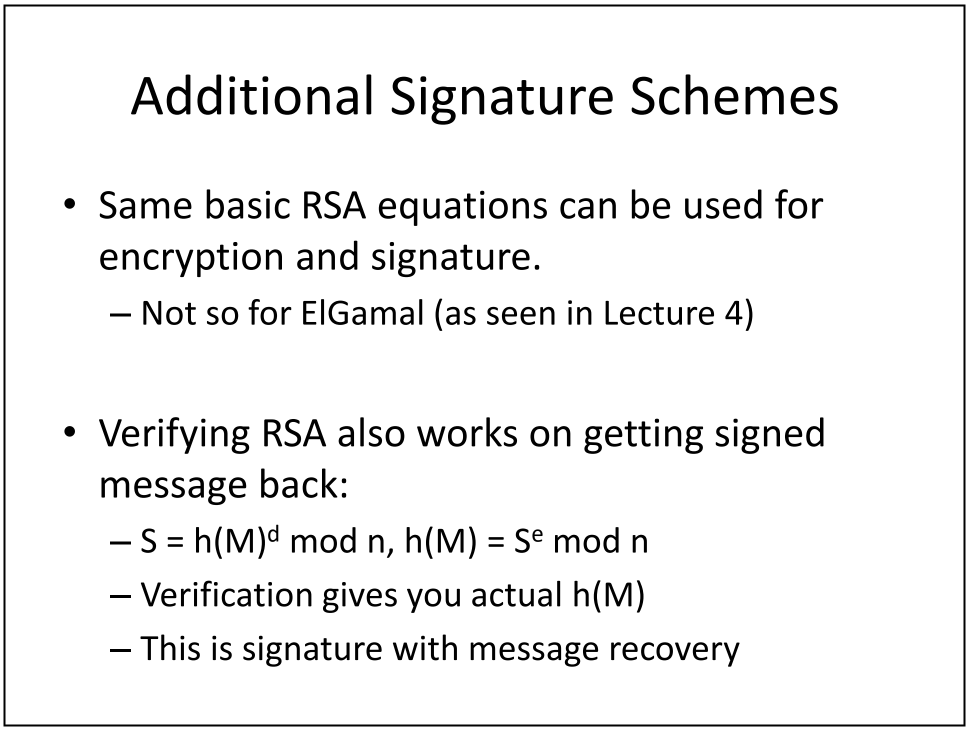


Figure: Wikipedia

38

For interest only

Some modes can provide both confidentiality and authentication from processing message once (encrypt and MAC combined into one mode).



## Additional Signature Schemes

- Same basic RSA equations can be used for encryption and signature.
  - Not so for ElGamal (as seen in Lecture 4)
- Verifying RSA also works on getting signed message back:
  - $S = h(M)^d \bmod n$, $h(M) = S^e \bmod n$
  - Verification gives you actual h(M)
  - This is signature with message recovery

Slides on Elgamal and DSA signature for interest only.

With message recovery as the thing you signed is recovered when you verify.

## ElGamal Signature

ElGamal
Private key x
Base g
Prime p
Random k
Message H(m)

Public key $y = g^x \ mod \ p$

$r = g^k \ mod \ p$

$s = (H(m) - xr).k^{-1} \ mod \ (p-1)$

Signature (r,s)

Verify

$g^{H(m)} \ mod \ p = y^r.r^s \ mod \ p$

## Digital Signature Algorithm (DSA)

DSA
Private key x
Base g
Primes p and q
Random k
Message H(m)

Public key $y = g^x \ mod \ p$

$r = (g^k \ mod \ p) \ mod \ q$

$s = (k^{-1}.(H(m) + xr)) \ mod \ q$

Signature (r,s)

Verify

$w = s^{-1} \ mod \ q$

$u_1 = H(m).w \ mod \ q$

$u_2 = r.w \ mod \ q$

Check $r = v = ((g^{u_1}.y^{u_2}) \ mod \ p) \ mod \ q$

– This is signature with appendix (So is ElGamal)

## The end!

?

Any questions…