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)

Digital Signature
Hash Function
Message Authentication Code

Integrity

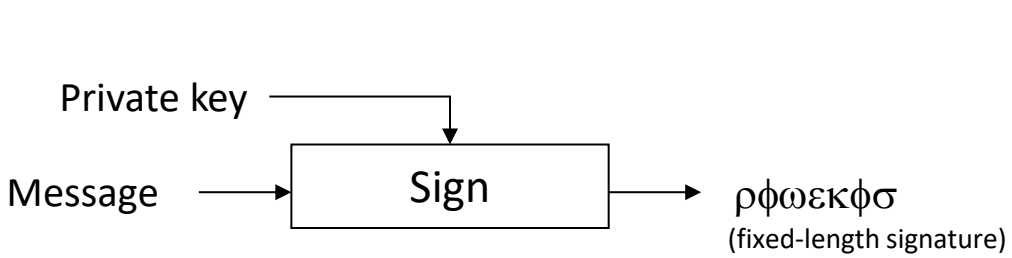
- Can encryption also provide integrity services? Does encrypting a message prevent:
  - Changing part of a message
  - Deletion of part of a message
  - Insertion of a false message
  - Falsifying the origin of a message
- Levels of integrity
  - Detect {accidental} modification
  - Data origin authentication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

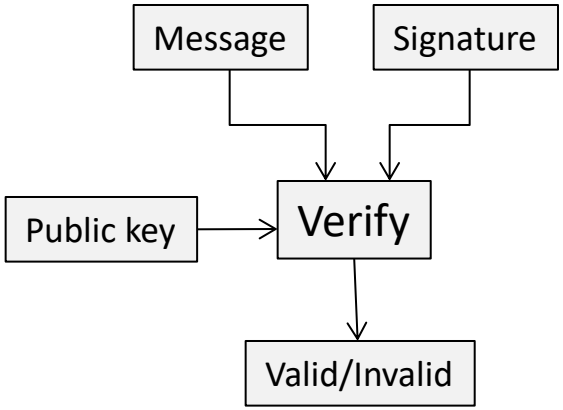
Electronic Signature

- 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)

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

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

RSA Signature Scheme

- **Setup:**
  - n = pq where p, q are large prime (say 512 bits long each)
  - ed = 1 mod (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?

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!

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

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?

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
```

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?

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.

Birthday Problem

- How many people must be in a room before probability is ≥ 1/2 that two or more have same birthday?
  - $1 - 365/365 \cdot 364/365 \cdot \cdot \cdot (365-K+1)/365$
  - 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).

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.

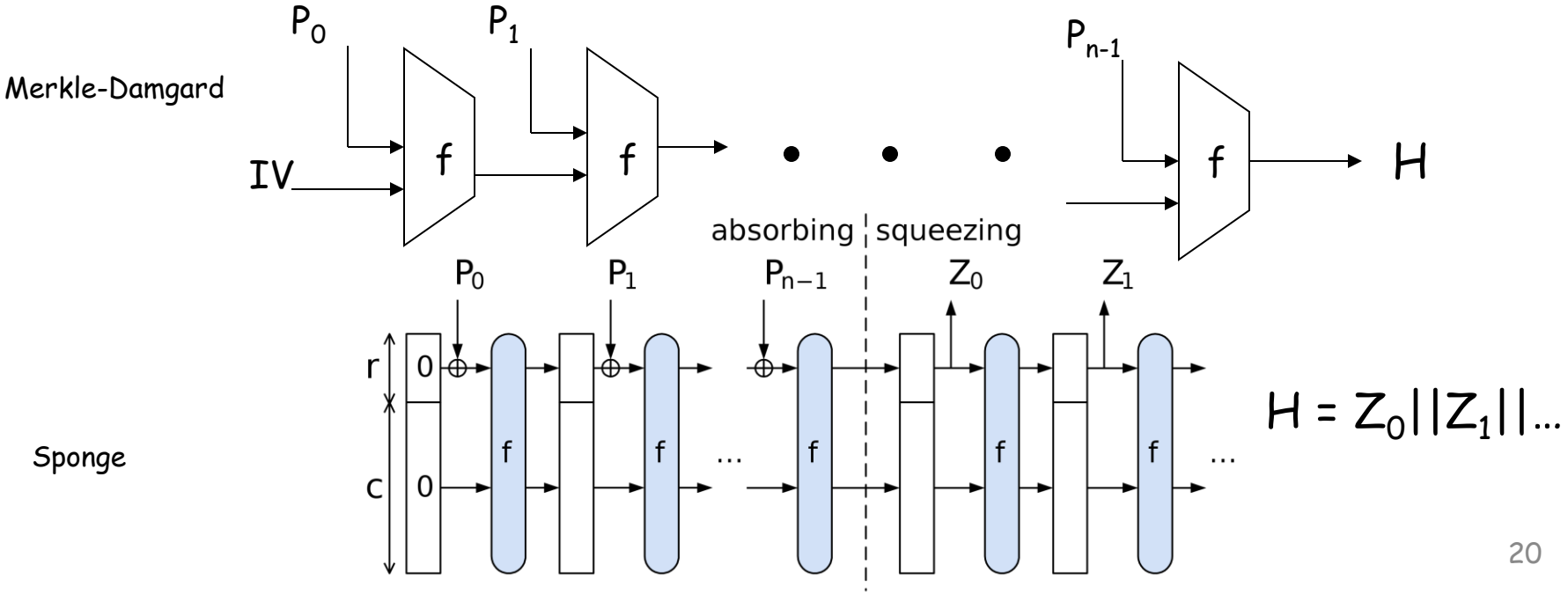
Block Ciphers as Hash Functions

- 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?

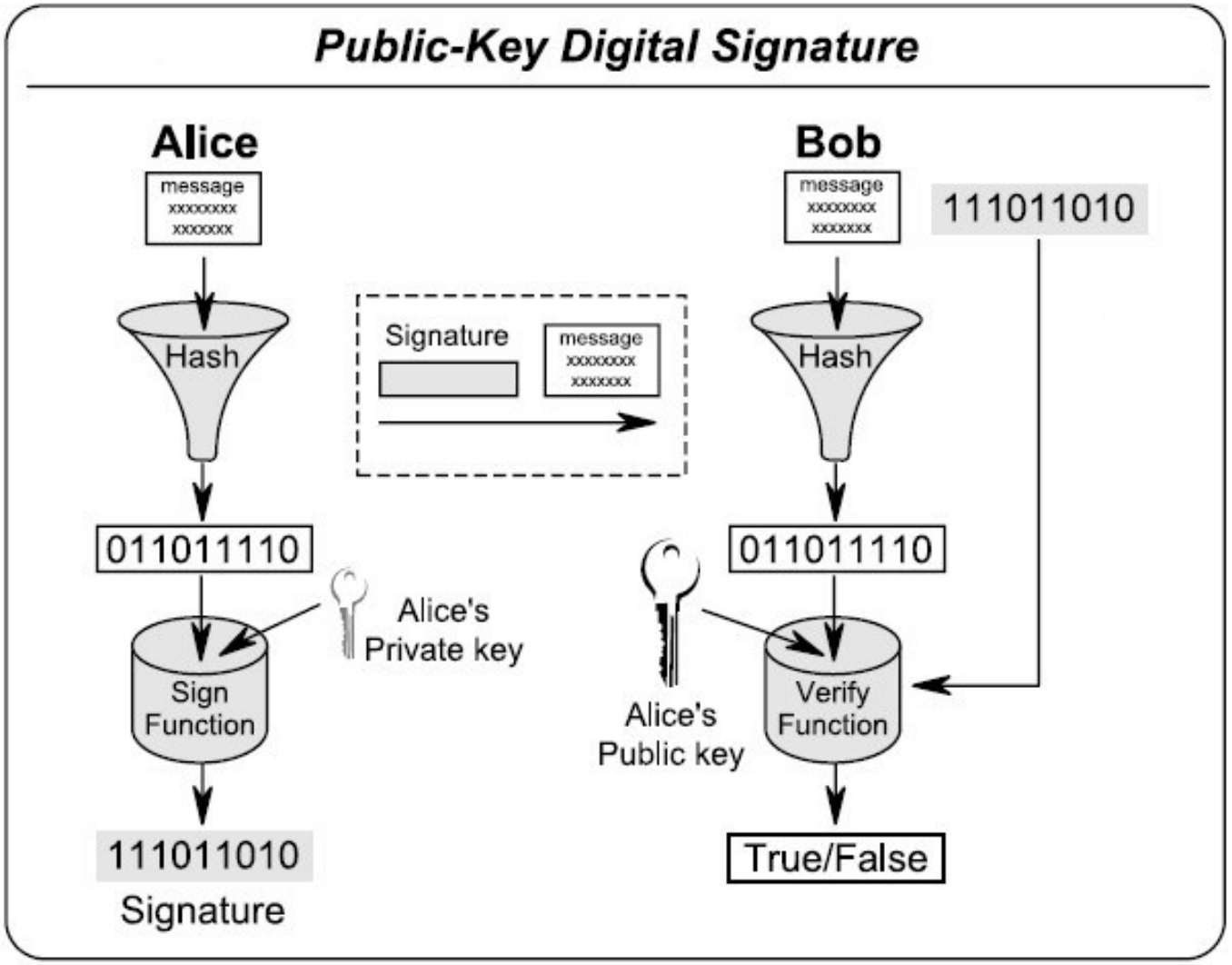
General Design of Hash Algorithms

- 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.

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



What is the main application of cryptographic hash functions?


Public-Key Digital Signature
Alice
message
xxxxxxx
xxxxxxx
Hash
011011110
Alice's
Private key
Sign
Function
111011010
Signature
Signature
message
xxxxxxx
xxxxxxx
Bob
message
xxxxxxx
xxxxxxx
111011010
Hash
011011110
Alice's
Public key
Verify
Function
True/False

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

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

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.

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

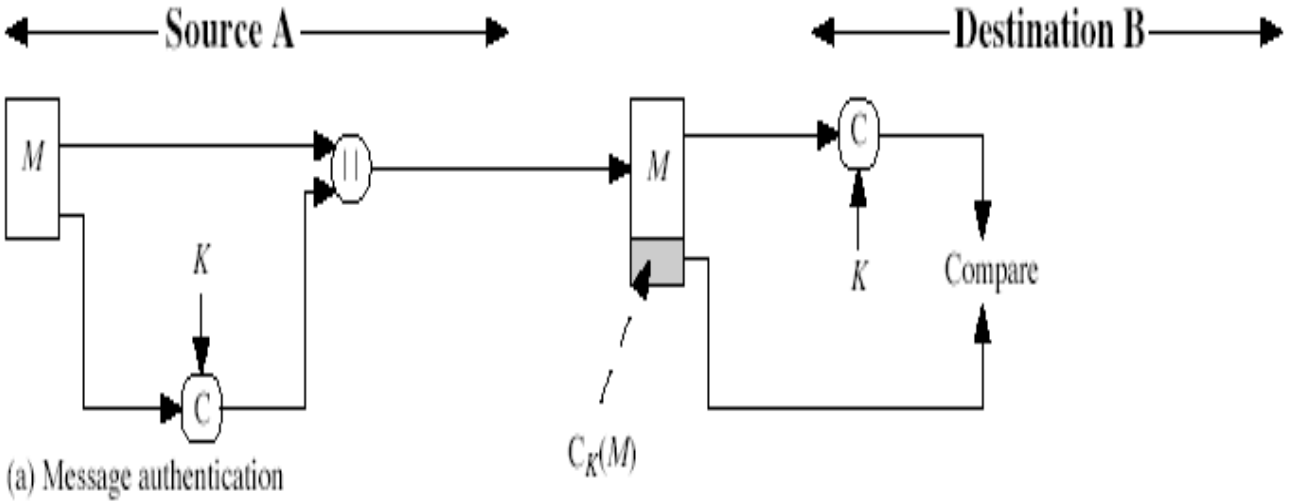
Message Authentication

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)

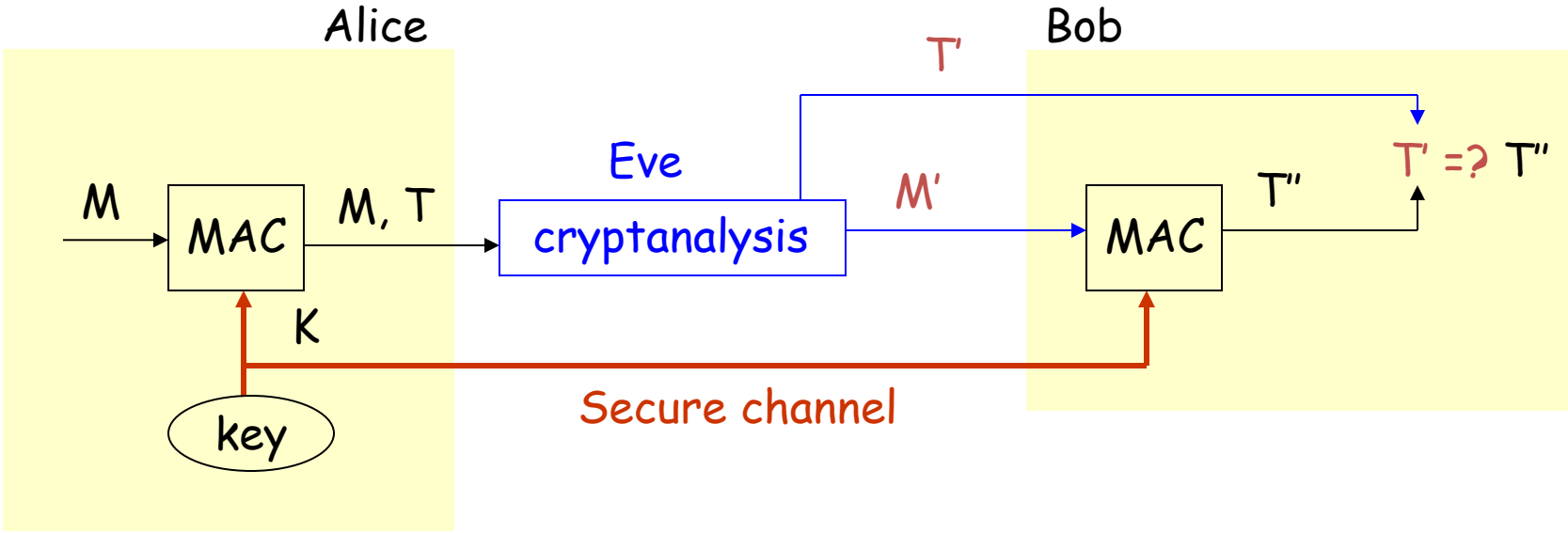
MAC

- 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



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



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.

$C_0 = E(K, P_0)$,

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

$C_2 = E(K, C_1 \oplus P_2)$,…

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

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.**

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 $T''' = T''$.

How?

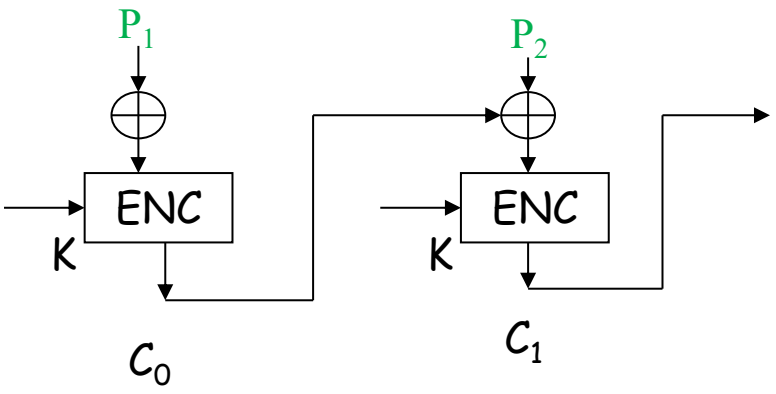
P`

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

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

P``

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



P```

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

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

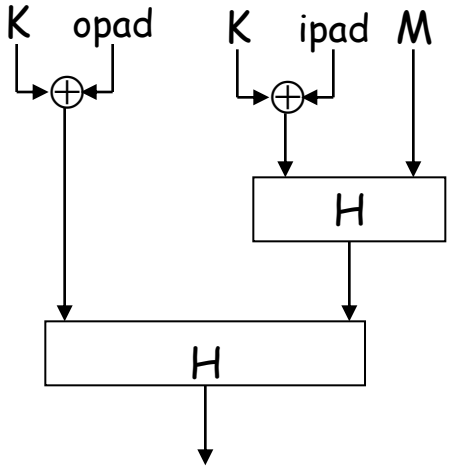
$C_2 = E(K, C_1 \oplus (P_3 \oplus T`)) = E(K, T' \oplus (P_3 \oplus T`)) = E(K, P_3)$ since $T' = C_1$

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

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

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

What about integrity and confidentiality?

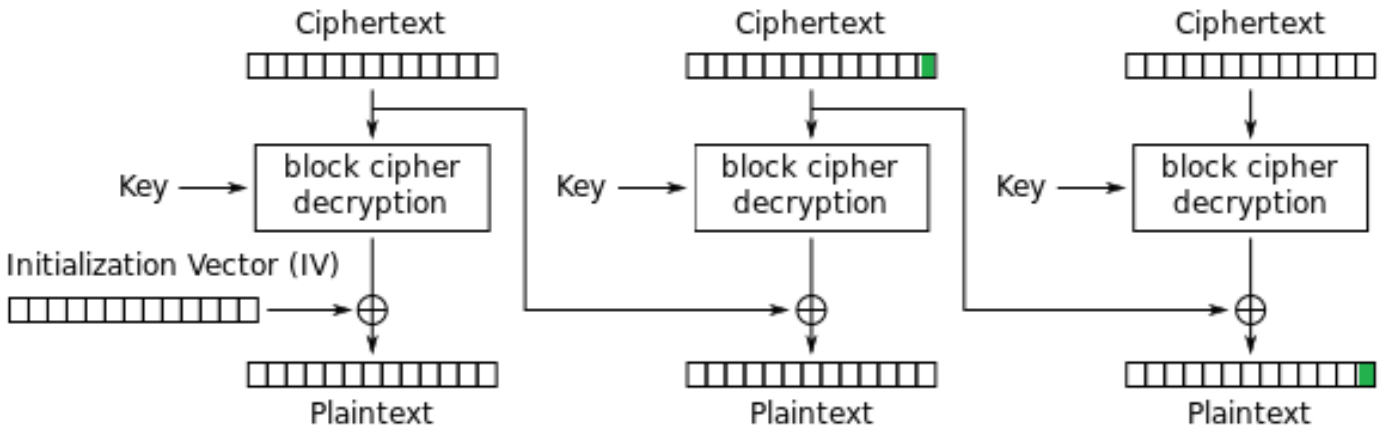
- How should we go about providing both?
- We can encrypt the data and do a MAC

Encrypt and then MAC?

or

MAC and then Encrypt?

Padding Oracle



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

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

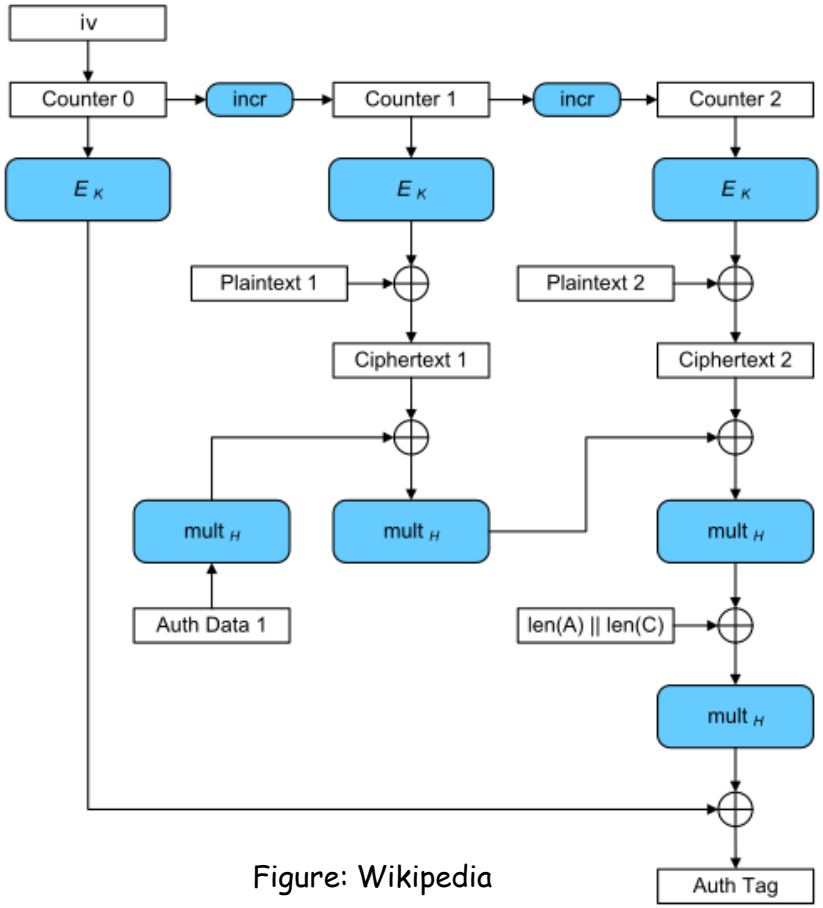


Figure: Wikipedia

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

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…