MONASH University
Information Technology

# FIT3031 INFORMATION & NETWORK SECURITY

MONASH University
Information Technology

FIT3031 INFORMATION & NETWORK SECURITY

# Lecture 3:

# Asymmetric Encryption Techniques

# Unit Structure: Lecture Topics

- ✓ OSI security architecture
  - **common security standards and protocols for network security applications**
  - **common information risks and requirements**
- ✓ operation of private key encryption techniques
- ✓ operation of public encryption techniques
- concepts and techniques for digital signatures, authentication and non-repudiation
- security threats of web servers, and their possible countermeasures
- Wireless Security Issues
- security threats of email systems and their possible countermeasures
- IP security
- intrusion detection techniques for security purpose
- risk of malicious software, virus and worm threats, and countermeasures
- firewall deployment and configuration to enhance protection of information assets
- network management protocol for security purpose

# Review of Last Lecture

**Key points from the last lecture:**

- **Cryptography is the major technique behind network security and a means to achieve: authentication, data integrity, confidentiality, digital signature and privacy**
- **There are two main types of cryptographic techniques:**
  - Symmetric encryption: uses a key
    - Block cipher
    - Stream cipher
  - Asymmetric encryption: uses a pair of keys
- **Feistel cipher architecture forms the basis of a number of symmetric encryption technique**
- **DES, one of the most widely used symmetric encryption, uses 16 rounds and operates on blocks of 64 bit. Currently 3DES is used**
- **Recently, the US National Institute of Standards and Technology (NIST) officially endorsed Advanced Encryption Standard (AES)**
- **Five common cipher block operation modes are: Electronic Codebook (ECB) mode, Cipher Block Chaining (CBC) mode, Cipher Feedback (CFB) mode, Output Feedback (OFB) mode and Counter (CTR) mode**
- **Stream cipher basic structure and RC4 algorithm uses variable key size and random permutation to scramble input message processed byte at a time.**
- **A trusted third party is needed for key distribution**

# Lecture 3: Objectives

- **Understand the principles of public key encryption**
- **Be familiar with public key encryption algorithms**
- **Understand how public key encryption can be used to exchange key for private key encryption**
- **Appreciate the use of hash function to achieve authentication of message**
- **Understand how public key encryption can be used to produce digital signature**

# Other names for these encryptions

| Symmetric Key Encryption | Asymmetric Key Encryption |
|---|---|
| Uses only **ONE key** | uses **TWO keys: a public and a private key** |
| Also known as **private key encryption** | Also called **public key encryption** |
| Also known as *shared key* or *shared secret* encryption or simply secret key encryption | |
| Also known as single key encryption | Also known as dual key encryption |
| Data encrypted and decrypted with the **same ONE** key | Data encrypted with **one key** can be decrypted only with the **other key** |
| A major challenge associated with symmetric key cryptosystems is the secure distribution of keys | A certificate through CA can also be used to uniquely identify the user and distribute the public keys. |
| Common symmetric key encryption algorithms include DES (the Data Encryption Standard) and AES (the Advanced Encryption Standard) | Common Asymmetric key encryption algorithms include RSA (the Data Encryption Standard) and DSA (the Advanced Encryption Standard) |
| symmetric encryption imposes a low computational burden, and tends to be much faster | Compared to symmetric encryption, asymmetric encryption imposes a high computational burden, and tends to be much slower |
| The security of the exchange relies on the security of the symmetric key | its major strength is its ability to establish a secure channel over a non secure medium |

# Lecture 3: Outline

- **Asymmetric encryption**
  - components
  - principle
- **Asymmetric Encryption algorithms**
  - RSA algorithm
  - Diffie-Hellman key exchange
- **Message Authentication Code**
- **Hash Function**
- **Digital signatures**
- **Key Management**

# Symmetric Cryptography

- **Traditional private/secret/single key cryptography uses only one key**
- **The key is shared by both sender and receiver**
- **Security is compromised if this key is disclosed, intentionally or unintentionally**
- **Concern:**
  - Key exchange
  - Number of keys required is directly proportional to the number of sender/receiver pairs
  - Does not protect sender from receiver forging a message & claiming it is sent by sender

# Asymmetric Cryptography

- Probably most significant advance in the 3000 year history of cryptography
- uses two keys – a public & a private key
- Asymmetric since parties are not equal
- Uses clever application of number theory concepts to function
- Complements rather than replaces private key crypto

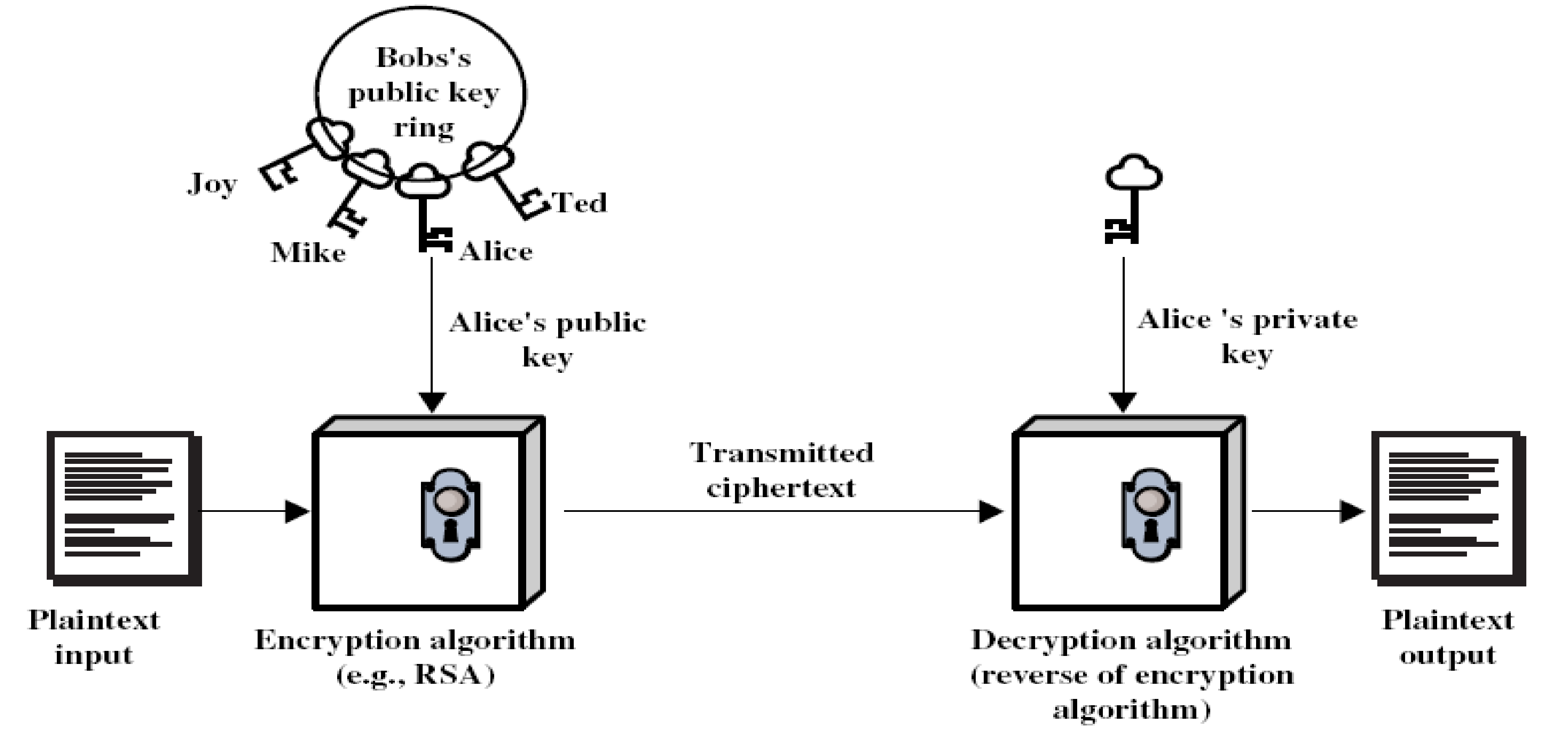
# Asymmetric Cryptography …

- **The key pair (k1 and k2) is related such that the data encrypted by k1 can only be decrypted by its corresponding pair k2**
- **Any of the keys can be used for encryption and the other for decryption**
- **Of the key pair**
  - **private key -** one of the keys is kept secret by the owner (hence called private key)
  - **public key -** the other is published (hence called public key)
- **Even if someone has one key, he/she can't compute the other**

# Asymmetric Encryption …

- **The encryption scheme has the following components:**
  - > plaintext
  - > encryption algorithm
  - > Private & public key **pair**
    - – public key-can be used by anyone
      - » encrypt messages and verify signatures
    - – private key- known to the owner only
      - » decrypt messages and sign (create) signatures
  - > ciphertext
  - > decryption algorithm
- **Asymmetric because**
  - those who encrypt the message cannot decrypt it
  - those who verify the signature cannot create it

# Asymmetric Encryption




**A simplified model of asymmetric encryption**

# Why asymmetric encryption ?

- **Developed to address two key issues:**
  - **key distribution** – how to have secure communications in general without having to trust a KDC with your key
  - **digital signatures** – how to verify a message comes intact from the claimed sender
- **public invention due to Whitfield Diffie & Martin Hellman at Stanford Uni in 1976**
  - known earlier in classified community

# Symmetric vs. Public-Key

| Conventional Encryption | Public-Key Encryption |
|---|---|
| *Needed to Work:*<br>1. The same algorithm with the same key is used for encryption and decryption.<br>2. The sender and receiver must share the algorithm and the key.<br>*Needed for Security:*<br>1. The key must be kept secret.<br>2. It must be impossible or at least impractical to decipher a message if no other information is available.<br>3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key. | *Needed to Work:*<br>1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption.<br>2. The sender and receiver must each have one of the matched pair of keys (not the same one).<br>*Needed for Security:*<br>1. One of the two keys must be kept secret.<br>2. It must be impossible or at least impractical to decipher a message if no other information is available.<br>3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key. |

# Public-key Characteristics

- **Public-Key algorithms rely on the following characteristics :**
  - computationally easy for a party B to generate a key pair (public key KUb, private key KRb)
  - easy for sender to generate ciphertext:
    - > $C=E_{KUb}(M)$
  - easy for the receiver to decrypt ciphertext using private key
    - – $M=D_{KRb}(C)=D_{KRb}[E_{KUb}(M)]$

# Public-key Characteristics…

- **Public-Key algorithms rely on the following characteristics :**
  - computationally infeasible to determine private key (KRb) knowing public key (KUb)
  - computationally infeasible to recover message M, knowing KUb and ciphertext C
  - either of the two keys can be used for encryption, with the other used for decryption:

    - $M = D_{KUb}[E_{KRb}(M)] = D_{KRb}[E_{KUb}(M)]$

# Public key application

- **Three categories of use:**
  - → encryption/decryption (provide secrecy)
  - → digital signatures (provide authentication)
  - → key exchange (of session keys)
- **Some algorithms are suitable for all uses, others are specific to one**

# Mathematical Background: Prime Numbers

- **Prime Numbers:**
  - prime numbers only have divisors of 1 and self
    - > they cannot be written as a product of other numbers
    - > note: 1 is prime, but is generally not of interest
  - Example:-
    - > 2,3,5,7 are prime,
    - > 4,6,8,9,10 are not prime
  - prime numbers are central to number theory
  - list of prime number less than 200 is:
    - > 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199

| | 2 | 3 | | 5 | | 7 | | | |
|---|---|---|---|---|---|---|---|---|---|
| 11 | | 13 | | | | 17 | | 19 | |
| | | 23 | | | | | | 29 | |
| 31 | | | | | | 37 | | | |
| 41 | | 43 | | | | 47 | | | |
| | | 53 | | | | | | 59 | |
| 61 | | | | | | 67 | | | |
| 71 | | 73 | | | | | | 79 | |
| | | 83 | | | | | | 89 | |
| | | | | | | 97 | | | |
| 101 | | 103 | | | | 107 | | 109 | |
| | | 113 | | | | | | | |
| | | | | | | 127 | | | |
| 131 | | | | | | 137 | | 139 | |
| | | | | | | | | 149 | |
| 151 | | | | | | 157 | | | |
| | | 163 | | | | 167 | | | |
| | | 173 | | | | | | 179 | |
| 181 | | | | | | | | | |
| 191 | | 193 | | | | 197 | | 199 | |

# Mathematical Background: Factorization

- **Prime Factorization:**
  - to factor a number n is to write it as a product of other numbers: $n = a \times b \times c$
    - > Example: $3600 = 2^4 \times 3^2 \times 5^2$
  - factoring a number is relatively hard compared to multiplying the factors together to generate the number
  - the prime factorisation of a number n is when it is written as a product of primes
    - > e.g. $91 = 7 \times 13$ ;

# Mathematical Background:GCD

- **Greatest Common Divisor (gcd):**
  - gcd of integers a & b is an integer c if
    - > c is divisor of both a & b; and
    - > any divisor of a & b is a divisor of c
  - gcd can be determined by comparing their prime factorizations and using least powers
    - > Example:

**Note: $5^0 = 1$**

» $300 = 2^2 \times 3^1 \times 5^2 \quad = 2 \times 2 \times 3 \times 5 \times 5 \times 5^0$

» $18 = 2^1 \times 3^2 \quad = 2 \times 3 \times 3 \times 5^0$

» $\gcd(18,300) = 2^1 \times 3^1 \times 5^0 = 6$

# Mathematical Background:Modular Arithmetic

- **Modular arithmetic:**
  - In modular arithmetic, numbers "wrap around" upon reaching a given fixed quantity, which is known as the modulus:
    - 12 mod 7=5; 15 mod 12 = 3
  - define **modulo operator** "`a mod n`" to be remainder when a is divided by n
  - The (mod n) operator maps all integers into the set of integers {0,1,…, n-1}
  - Two integers a & b are said to be congruent modulo n, if (a mod n) = (b mod n);
    - congruence is written as $a \equiv b \bmod n$
    - example: $73 \equiv 4 \bmod 23$
      - 73 mod 23 = 4 & 4 mod 23 = 4

# Mathematical Background:Primitive Root

- ***a is a* Primitive Root of prime number *p***
  - If ***$a^n$ mod p = 1* to *p-1*** (distinct integers)
  - Where ***n*** is an integer ***1* through *p-1***
  - Then ***a*** is the primitive root of ***p***

# Asymmetric Encryption Algorithms

- **Like symmetric encryption brute force exhaustive search attack is always theoretically possible**
- **But keys used are too large ( e.g. 1024 bits)**
- **Security relies on a large enough difference in difficulty between easy (en/decrypt) and hard (cryptanalysis) problems**
- **More generally the hard problem is known, its just made too hard to do it in practice**
- **Requires the use of very large numbers**
- **Hence is *slow* compared to symmetric encryption**

# Asymmetric Encryption Algorithms

- **Two widely used algorithms:**
  - **RSA Algorithm**
    - > developed by Ron Rivest, Adi Shamir and Len Adleman at MIT proposed in 1978
    - > security relies on the cost of factoring large numbers
  - **Diffie-Hellman Algorithm**
    - > developed by Whitfield Diffie and Martin Hellman in 1976
    - > security relies on the difficulty of computing discrete logarithms

# RSA Algorithm - Key generation

- **Steps involve:**
  - select **two large primes number** p, q at random
  - calculate $N = p \times q$
  - calculate $ø(N) = (p-1)(q-1)$
  - Select an integer e such that
    - ❖ $1 < \mathbf{e} < ø(N)$, $\gcd(\mathbf{e}, ø(N))=1$
  - Select d such that
    - > $d \cdot e \equiv 1 \bmod ø(N)$ , i.e., **$d \cdot e \bmod ø(N)=1$**
  - public key: $KU=\{\mathbf{e}, N\}$
  - private decryption key: $KR=\{\mathbf{d}, N\}$

# RSA Algorithm - Key generation

- ***Example:***
  - select p=17, q=11 at random
  - calculate N=pxq $=17\times11=187$
  - calculate ø(N)=(p-1)x(q-1)=(17-1)(11-1)=160
  - Select an integer **e** such that
    - $1 < \mathbf{e} < 160$, gcd(**e**,160)=1; choose **e**=7
  - Select **d** such that **d x e** $\equiv 1 \bmod 160$,
    - > i.e., **d**$\times7\equiv 1 \bmod 160$; (**d**$\times$7 mod 160)=1; 161 mod 160=1
    - > The correct answer is : **d** = 23
  - public key: KU={**e**, N} == {**7**,187}
  - private decryption key: KR={**d**, N} == {**23**,187}

# RSA Algorithm …

- **RSA is a block cipher**
  - each block must have a binary value $M < N$
  - in practice block size is k bits, $2k < N < 2k+1$
- **Encryption mode:**
  - ciphertext: $C = M^e \bmod N$
  - message, $M= C^d \bmod N= (M^e)^d \bmod N= M^{ed} \bmod N$
- **Signature mode:**
  - signature: $S = M^{d} \bmod N$
  - message, $M= S^{e} \bmod N= (M^d)^e \bmod N= (M^{ed}) \bmod N$
- **Note that the message M must be smaller than the modulus N**

# Requirement of RSA

- **For RSA algorithm to work satisfactorily, the following requirements must be met:**
  - It is possible to find values of e, d, N such that **$M^{ed} = M \bmod N$ for all $M < N$**
  - It is relatively easy to calculate $M^e$ and $C^d$ for all values of $M < N$
  - It is **infeasible** to determine **d** given **e** and **N**. This requirement is met when **e** and **N** are very large values

# RSA Example - Encryption/Decryption

- **sample RSA encryption/decryption is:**
- **given message M = 88 (note: M < N; i.e. 88 < 187)**
- **encryption: $C = M^e \bmod N$**
  - $C = 88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88 \bmod 187)] \bmod 187 = (132 \times 77 \times 88) \bmod 187 = 11$
  - $C = 11$
- **decryption: $M = C^d \bmod N$**
  - $M = 11^{23} \bmod 187$
  - $= [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$
  - $=(11 \times 121 \times 55 \times 33 \times 33) \bmod 187 = 88$

# Diffie-Hellman Key Exchange

- **First public-key algorithm proposed**
- **By Diffie & Hellman in 1976 along with the exposition of public key concepts**
  - note: now known that James Ellis (UK CESG) secretly proposed the concept in 1970
- **A practical method for secure exchange of a secret key**
- **Used in a number of commercial products**

# Diffie-Hellman Key Exchange

- **a public-key distribution scheme**
  - **cannot** be used to exchange an arbitrary message
  - rather it can **establish a common key**
  - known only to the **two** participants
- **value of key depends on the participants (and their private and public key information)**
- **based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) - easy**
- **security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard**

# Diffie-Hellman Setup

- **all users agree on global parameters:**
  - **large prime integer** or polynomial **$q$**
  - **$a$** being a **primitive root** of **$q$**
- **each user (e.g. A) generates their key**
  - chooses a secret/private key (number): $x_A < q$
  - compute their public key: $y_A = a^{x_A} \bmod q$
- **each user publish their public key $y_A$**

# Diffie-Hellman Key Exchange

- shared session key for users A & B is $K_{AB}$ :-

  $K_{AB} = a^{x_A . x_B} \bmod q$

  $= y_A^{\ x_B} \bmod q$ (which B can compute)

  $= y_B^{\ x_A} \bmod q$ (which A can compute)

- $K_{AB}$ is used as session key in private-key encryption scheme between Alice and Bob
- if Alice and Bob subsequently communicate, they will have the **same key** as before, unless they choose new public-keys
- attacker needs an $x_A$ or $x_B$, must solve discrete log

# Diffie-Hellman Example

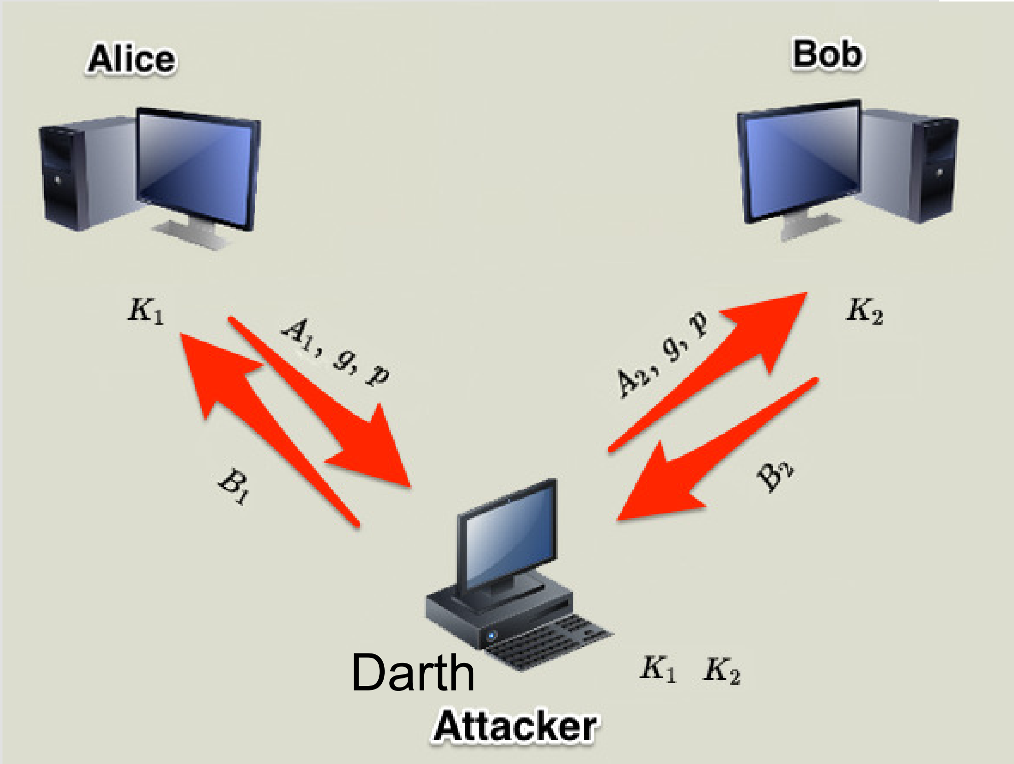
- **users Alice & Bob who wish to exchange keys:**
- **both agree on prime number $q=353$ and primitive root $a=3$**
- **select random secret private keys:**
  - A chooses $x_A=97$, B chooses $x_B=233$
- **compute respective public keys:**
  - $y_A=3^{97} \bmod 353 = 40$ (Alice)
  - $y_B=3^{233} \bmod 353 = 248$ (Bob)
- **compute shared session key as:**
  - $K_{AB}= y_B^{\ x_A} \bmod 353 = 248^{97}= 160$ (Alice)
  - $K_{AB}= y_A^{\ x_B} \bmod 353 = 40^{233}= 160$ (Bob)

# Key Exchange Protocols

- users could create random private/public D-H keys each time they communicate
- users could create a known private/public D-H key and publish in a directory, then consulted and used to securely communicate with them
- both of these are vulnerable to a man-in-the-Middle Attack
- Hence; authentication of the keys is needed

# Man-in-the-Middle Attack

- Darth prepares by creating two set of private / public key pairs
- Alice transmits her public key to Bob
- Darth intercepts this and transmits his first public key to Bob. Darth also calculates a shared key with Alice
- Bob receives the public key and calculates the shared key (with Darth instead of Alice)
- Bob transmits his public key to Alice
- Darth intercepts this and transmits his second public key to Alice. Darth calculates a shared key with Bob
- Alice receives the key and calculates the shared key (with Darth instead of Bob)
- Darth can then intercept, decrypt, re-encrypt, forward all messages between Alice & Bob



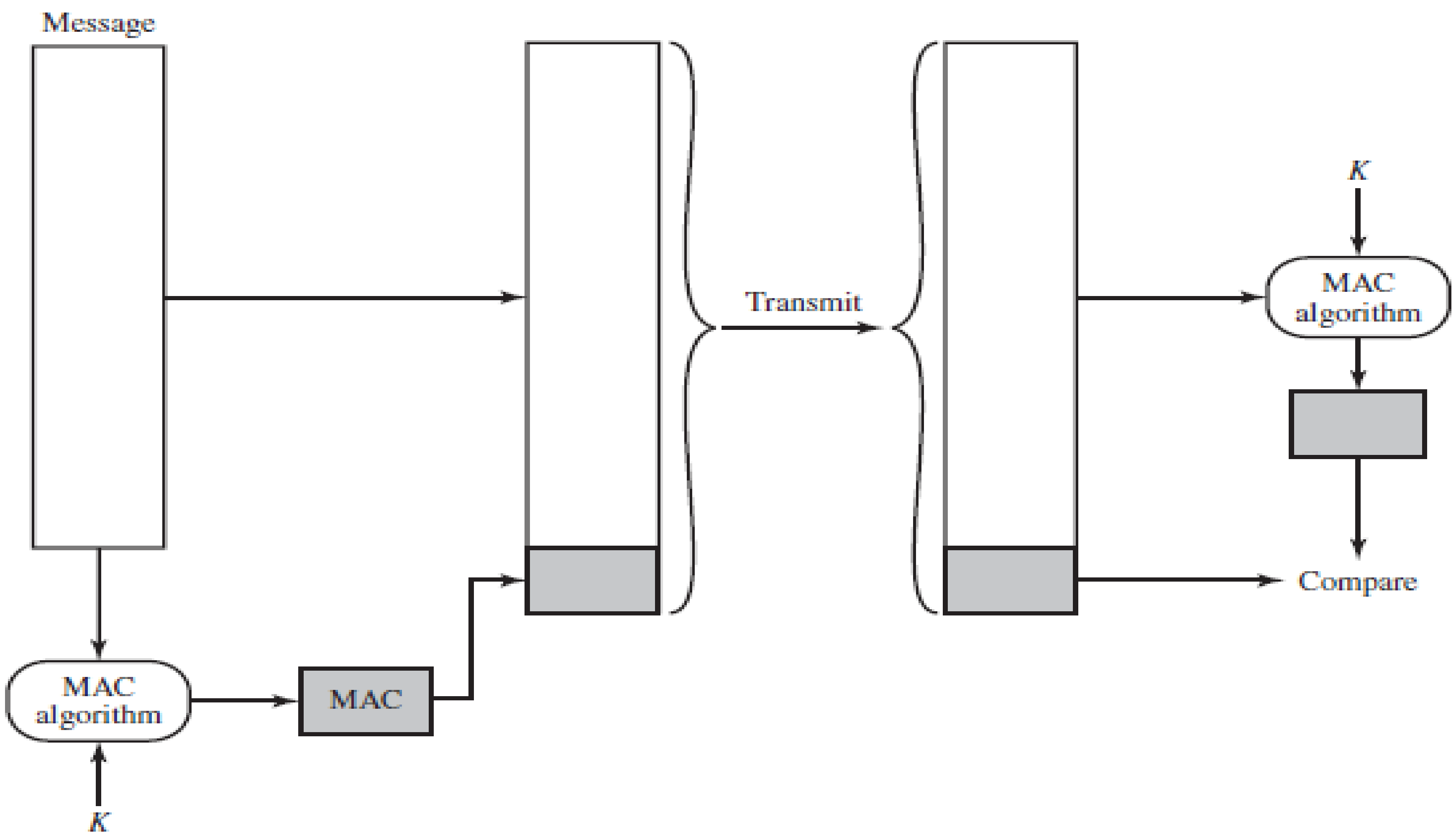
# Message Authentication

- **message authentication is concerned with:**
  - protecting the integrity of a message
  - validating identity of originator
  - non-repudiation of origin (dispute resolution)
- **The alternative functions used:**
  - → message authentication code (MAC)
  - → hash function
  - → message encryption
  - → Digital Signature

# Message Authentication Code

- **Message Authentication Code (MAC) is a small block of data generated, appended and transmitted with the message**
- **MAC is a function of the message, M as well as the secret key, K**
  – MAC = $C_K(M)$
- **MAC is a one way function**
  – From MAC, message M cannot be derived
- **The receiver performs same computation on message and checks it matches the received MAC**
- **If a single bit of the message is changed the calculated MAC won't match the transmitted MAC**
- **A MAC is not a digital signature**

# Message Authentication Code …

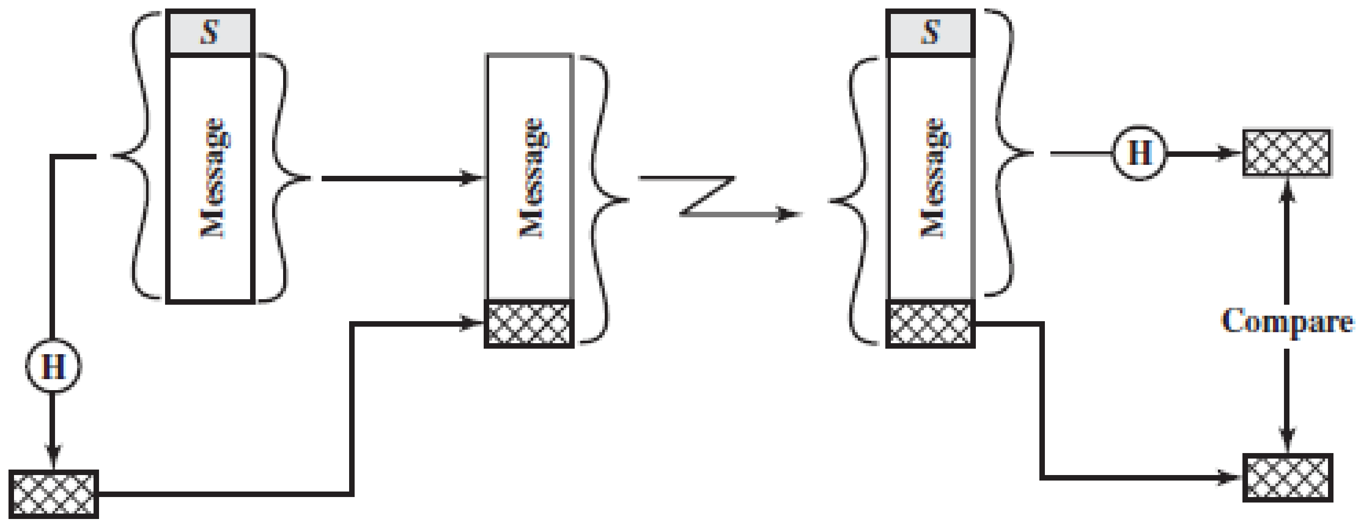


# Hash Function

- **Like MAC, hash function is a one way function that accepts a message M and produces a fixed-size message digest H(M)**
- **Unlike MAC, hash function takes only the message (not the key) as input to generate the message digest**
- **Hash is used to detect changes to message**
- **Can be used with both symmetric and asymmetric encryption**
  - used in various ways with message
  - Most often to create a digital signature

# Hash Function

- Secret value S is known to A & B (optional)
- A calculates hash function of secret value + Message, i.e., MD=H(S ‖ M)
- M ‖ MD is sent
- B recalculates H(S ‖ M) and verify




(c) Using secret value

# Requirement of Hash Function

- **Can be applied to any sized message M**
- **Produces fixed-length output h**
- **Is easy to compute h=H(M) for any message M**
- **Given h is infeasible to find x such that H(x)=h**
  - one-way property
- **Given x it is computationally infeasible to find y (≠x)**
  - such that H(y)=H(x)
    - > collision resistance
- **Is computationally infeasible to find any pair (x,y) such that H(y)=H(x)**
  - collision resistance

**https://www.tools4noobs.com/online_tools/hash/**

MONASH University
Information Technology

# Hash Function Requirements

| Requirement | Description |
|---|---|
| Variable input size | H can be applied to a block of data of any size. |
| Fixed output size | H produces a fixed-length output. |
| Efficiency | $H(x)$ is relatively easy to compute for any given $x$, making both hardware and software implementations practical. |
| Preimage resistant (one-way property) | For any given hash value $h$, it is computationally infeasible to find $y$ such that $H(y) = h$. |
| Second preimage resistant (weak collision resistant) | For any given block $x$, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$. |
| Collision resistant (strong collision resistant) | It is computationally infeasible to find any pair $(x, y)$ such that $H(x) = H(y)$. |
| Pseudorandomness | Output of H meets standard tests for pseudorandomness |

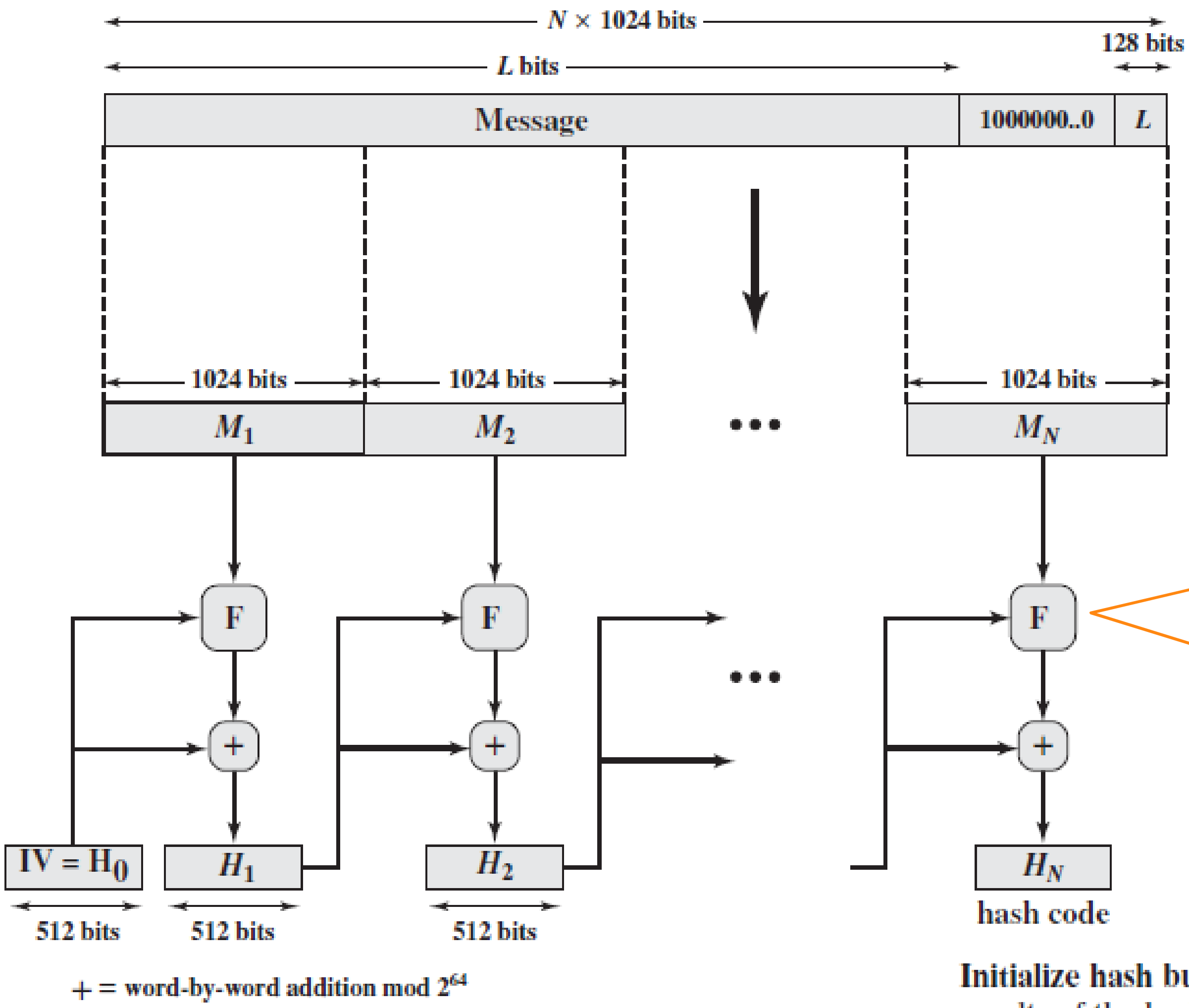
# Hash functions

- **MD2, MD4, MD5 - 128 bit message digest**
  - developed by Ronald Rivest
  - There are many successful attacks to MD2, MD4, MD5
- **SHA, SHA-1 -160 bit message digest**
  - On February 23, 2017 CWI Amsterdam and Google announced they had performed a collision attack against SHA-1, publishing two dissimilar PDF files which produce the same SHA-1 hash [1]
- **SHA-256 - 256 bit message digest**
- **SHA-384 - 384 bit message digest**
- **SHA-512 - 512 bit message digest**
- **SHA (Secure Hash Algorithm) – 256, 384, 512-bit have been proposed to use with AES-128, 192 and 512 bit encryption by the National Institute of Standards & Technology, USA**

**[1] https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html**

# SHA Versions

| | SHA-1 | SHA-224 | SHA-256 | SHA-384 | SHA-512 |
|---|---|---|---|---|---|
| **Message Digest Size** | 160 | 224 | 256 | 384 | 512 |
| **Message Size** | $< 2^{64}$ | $< 2^{64}$ | $< 2^{64}$ | $< 2^{128}$ | $< 2^{128}$ |
| **Block Size** | 512 | 512 | 512 | 1024 | 1024 |
| **Word Size** | 32 | 32 | 32 | 64 | 64 |
| **Number of Steps** | 80 | 64 | 64 | 80 | 80 |




# SHA-512 Overview

- Compression Function (F)
- Process the message in 1024-bit (128-word) blocks.
- Consists of 80 rounds within…
- which forms the heart of the algorithm.
- Each round takes as input the 512-bit buffer value $H_i$, and updates the contents of that buffer.

**Initialize hash buffer:** A 512-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as eight 64-bit registers ($a, b, c, d, e, f, g, h$). These registers are initialized to the following 64-bit integers (hexadecimal values):

| | |
|---|---|
| $a$ = 6A09E667F3BCC908 | $e$ = 510E527FADE682D1 |
| $b$ = BB67AE8584CAA73B | $f$ = 9B05688C2B3E6C1F |
| $c$ = 3C6EF372FE94F82B | $g$ = 1F83D9ABFB41BD6B |
| $d$ = A54FF53A5F1D36F1 | $h$ = 5BE0CD19137E2179 |

These values are stored in big-endian format, which is the most significant byte of a word in the low-address (leftmost) byte position. These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.

# SHA-512 Compression Function (F)

- **heart of the algorithm**
- **processing message in 1024-bit blocks**
- **consists of 80 rounds**
  – updating a 512-bit buffer
  – using a 64-bit value Wt derived from the current message block
  – and a round constant based on cube root of first 80 prime numbers

# Keyed Hash Functions as MACs

- **want a MAC based on a hash function**
  - because hash functions are generally faster
  - crypto hash function code is widely available
- **hash includes a key along with message**
- **original proposal:**
  - KeyedHash = **Hash(Key||Message)**
  - some weaknesses were found with this
- **eventually led to development of HMAC**

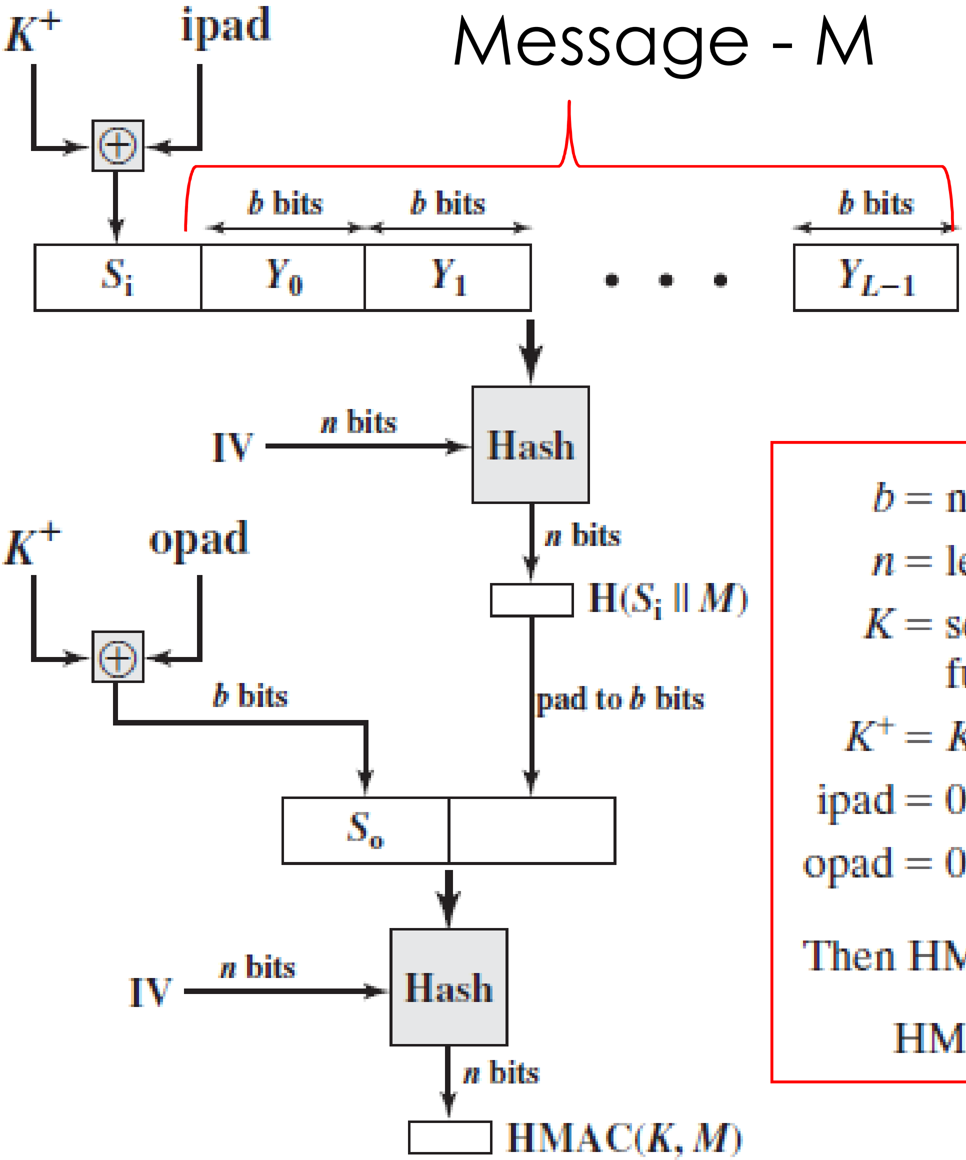
# HMAC Design Objectives

- use, without modifications of hash functions
- allow for easy replaceability of embedded hash function
- preserve original performance of hash function without significant degradation
- use and handle keys in a simple way.
- have well understood cryptographic analysis of authentication mechanism strength

# HMAC

- **specified as Internet standard RFC 2104**
- **uses hash function on the message:**
  - $HMAC_K(M) = Hash[(K^+ \text{ XOR } opad) \| Hash[(K^+ \text{ XOR } ipad) \| M)]]$
  - where $K^+$ is the key padded out to size
  - opad, ipad are specified padding constants
- **overhead is just 2 more hash calculations than the message needs alone**
- **any hash function can be used**
  - e.g.. MD5, SHA-1, RIPEMD-160, Whirlpool

# HMAC Overview




$b$ = number of bits in a block

$n$ = length of hash code produced by embedded hash function

$K$ = secret key; if key length is greater than $b$, the key is input to the hash function to produce an $n$-bit key; recommended length is $> n$

$K^+$ = $K$ padded with zeros on the left so that the result is $b$ bits in length

ipad = 00110110 (36 in hexadecimal) repeated $b/8$ times

opad = 01011100 (5C in hexadecimal) repeated $b/8$ times

Then HMAC can be expressed as

$$\text{HMAC}(K, M) = \text{H}[(K^+ \oplus \text{opad}) \| \text{H}[(K^+ \oplus \text{ipad}) \| M]]$$

MONASH University
Information Technology

# HMAC Security

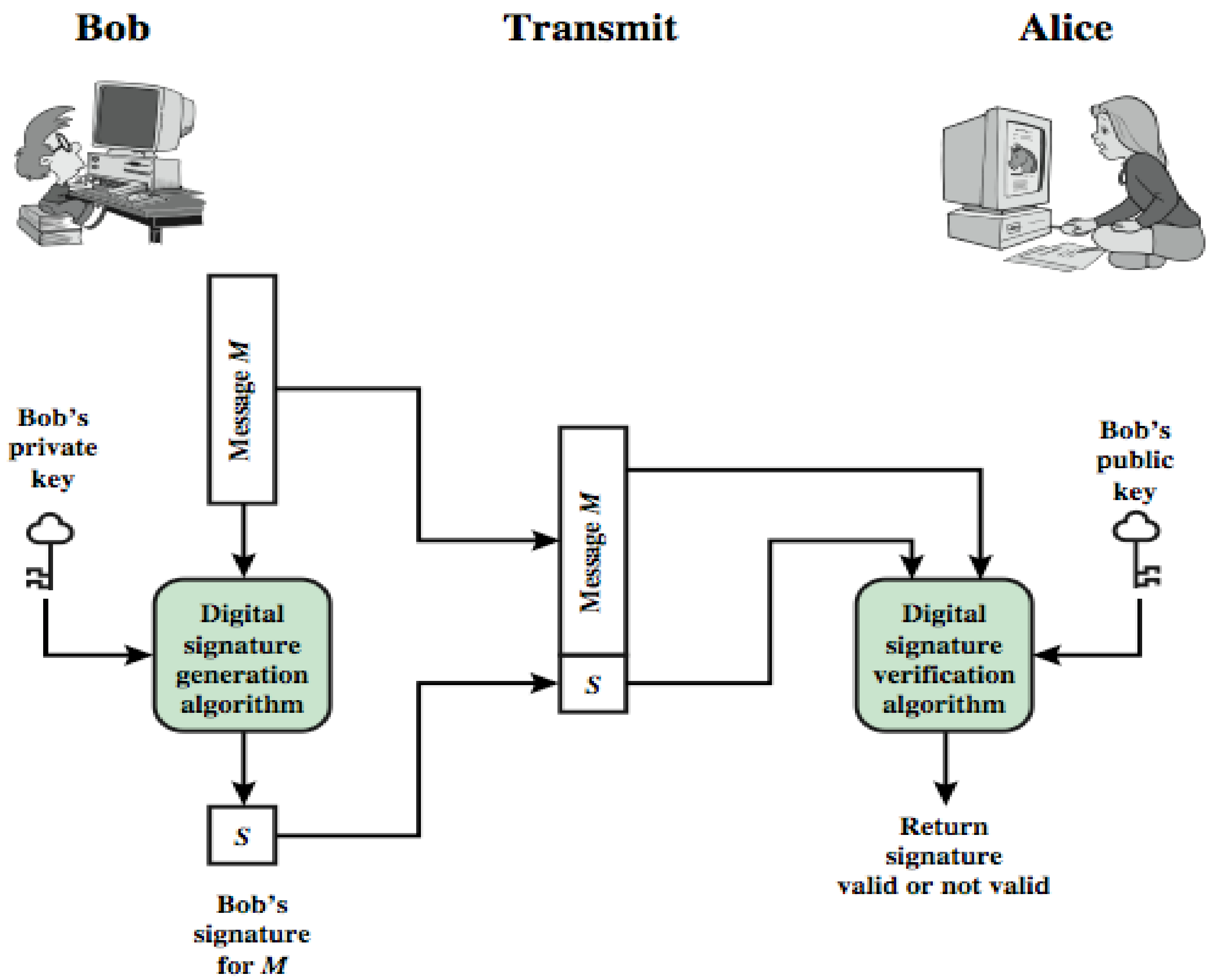
- **proved security of HMAC relates to that of the underlying hash algorithm**
- **attacking HMAC requires either:**
  - brute force attack on key used
  - birthday attack (but since keyed would need to observe a very large number of messages)
- **choose hash function used based on speed verses security constraints**

# Digital Signatures

- **shall look at message authentication**
  – but does not address issues of lack of trust
- **Digital signatures may provide the ability to:**
  – verify author, date & time of signature
  – authenticate message contents
  – be verified by third parties to resolve disputes
- **hence include authentication function with additional capabilities**

# Digital Signature Model:

Simplified Depiction of Essential Elements of Digital Signature Process

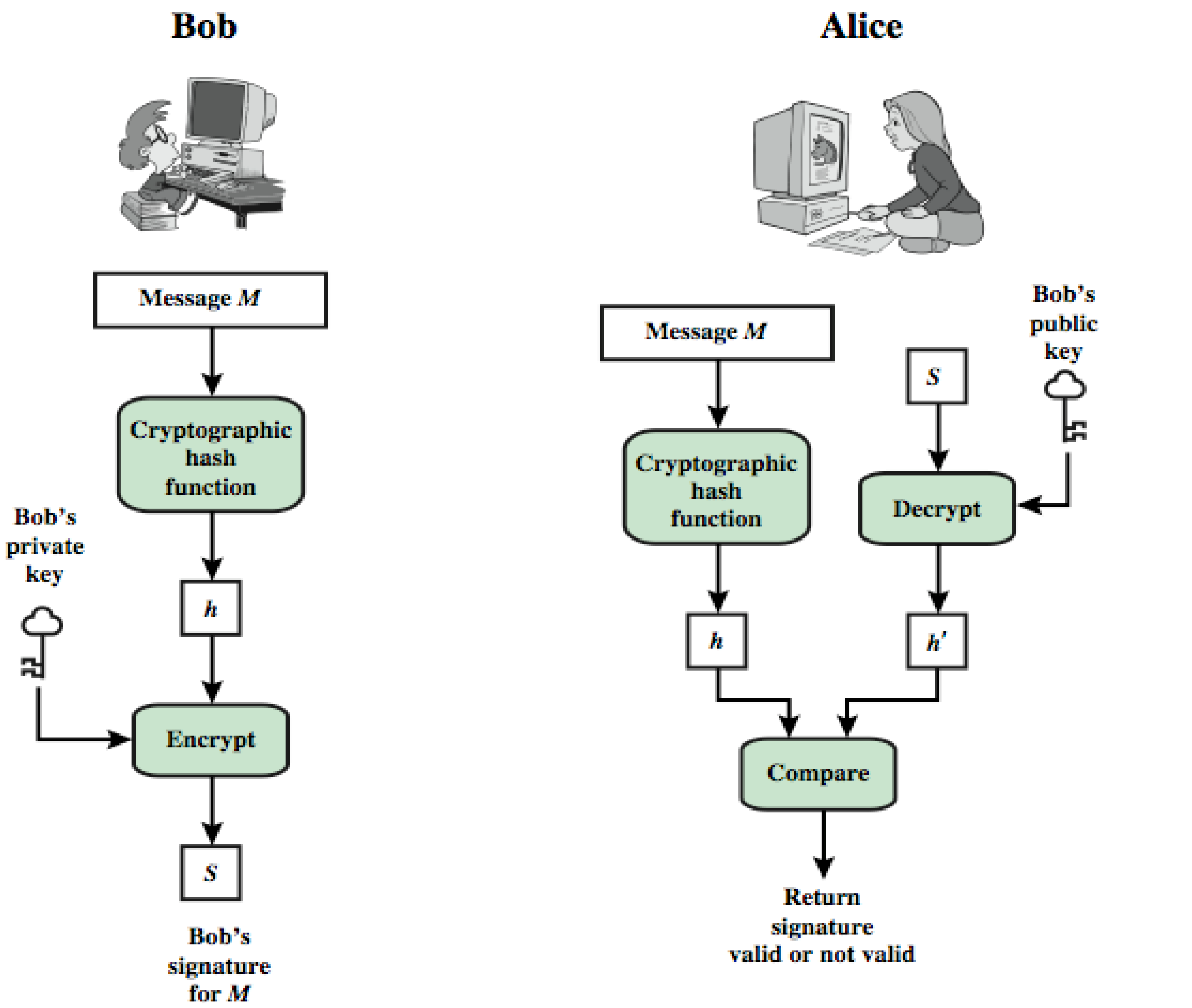


# Digital Signature Model

Digital Signature Model:

Alternate explanation:

Digital Signature Process



# Signcryption

- **How to protect data confidentiality and message integrity at the same time (both in asymmetric setting)?**
- **Sign-then-encrypt**
- **Encrypt-then-sign**
- **How to do it efficient?**
  – The computational costs and communication overheads of a signcryption scheme should be smaller than those of the best known signature-then-encryption schemes with the same provided functionalities

# Signcryption

- Invented by Yuliang Zheng in 1997
- Combine two algorithms (Signature + Public Key Encryption) together
- Input (Signcrypt): sender's secret key + receiver's public key + message
- Output (Signcrypt): Signcrypted message
- Input (De-Signcrypt): sender's public key + receiver's secret key + signcrypted message
- Output (De-Signcrypt): message or REJECT

# Digital Certificate

- **Digital signatures reproduce the electronic version of the normal signatures**
  - provide proof of identity
  - proof of message integrity
  - Supports non repudiation
- **Similar to MAC but not the same**
  - recipient must not be able to generate digital signature
  - in MAC recipient knows the key
- **A signed authenticator is appended to the message**
  - a data item logically associated with the message and originator
- **Digital certificate binds an identity to a pair of keys that can be used to encrypt & sign digital information.**
  - It makes it possible for anyone to verify claims from individuals that they have the right to use a given key.

# Key Management

- **Asymmetric encryption addresses the problem of key distribution:**
  - > distribute the public keys
  - > exchange the secret keys (symmetric keys)
- **Symmetric encryption is faster, problem is key sharing**
  - > Key can be shared by using asymmetric encryption
- **How to distribute public key?**
  - > sender can deny his public key
  - > hacker can create a false key and impersonate someone
  - > ***Solution: a trusted body is required to certify public key***
  - > ***Example Verisign, Thawte, AffirmTrust, certSIGN etc……..***

# Certification Authority (CA)

- **Certification Authority (CA) certifies the public key of an user**
  - user A generates his/her public key and submits to CA for certification
  - CA determines identity and background of A
  - CA appends time stamp to public key, generates hash code and encrypts with CA's private key
    - > this constitutes the signature of CA
    - > hash code ensure that public key is unaltered
  - Signed public key of A is now available for presentation
  - X.509 certification standard
  - CA example: VeriSign, Thawte,
- **Any one equipped with CA's public key can authenticate A's authenticity**
- **Most CA's public keys are stored in browser**

# Exchange of Session Key using Asymmetric algorithm

- **A session key is generated and shared for each communication session between A and B**
- **A generates a random session key**
  - destroyed when the session ends
- **Encrypt it with the public key of B**
  - certified public key is known
- **Send the encrypted session key to B**
- **B decrypt the session key using his/her private key**
- **Once both parties have the session key, rest of the communication is made using symmetric encryption, e.g, DES, AES**
  - much faster and easier

# Further Reading

- **Study Guide 3**
- **Chapter 3 of the textbook: Network Security Essentials-Application & Standards" by William Stallings 5th Edition, Prentice Hall, 2013**
- **Additional resources for this week**


- **Acknowledgement: part of the materials presented in the slides was developed with the help of Instructor's Manual and other resources made available by the author of the textbook.**