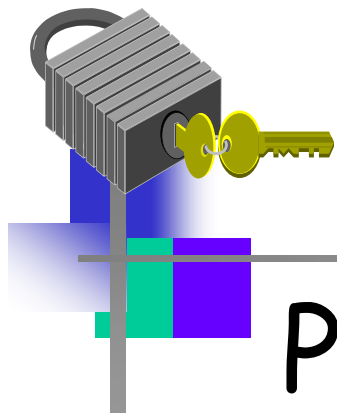


Computer and Information Security

Chapter 4 Public Key Cryptography

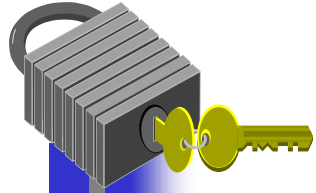


Chapter 4:

Public Key Cryptography

You should not live one way in private, another in public.
— Publilius Syrus

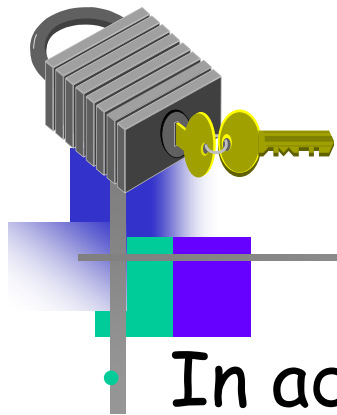
Three may keep a secret, if two of them are dead.
— Ben Franklin



Overview

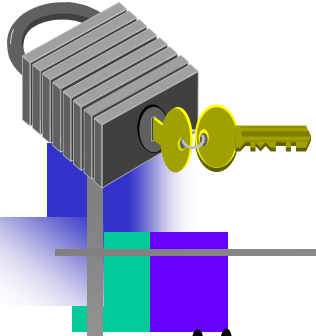
Message Authentication

- Secure Hash Functions and HMAC
- Public-Key Cryptography Principles
 - Encryption
 - Decryption
- Public-Key Cryptography Algorithms
 - Knapsack, RSA, ECC
- Digital Signatures
- Key Management



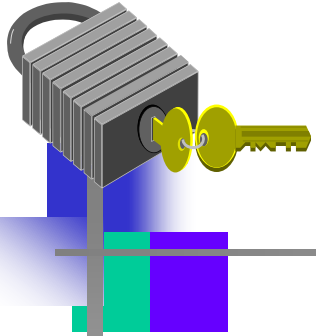
Authentication

- In addition to confidentiality, message authentication is an important security function
- "A message, file, document or data is said to be authentic when it is genuine and came from its alleged source."
- Encryption prevents against passive attacks (eavesdropping)
- Message Authentication prevents against active attacks or falsification.



Message Authentication

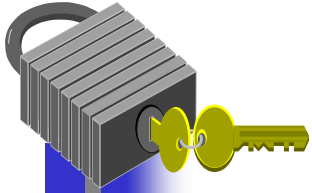
- Message authentication is concerned with:
 - protecting the integrity of a message
 - validating identity of originator
 - non-repudiation of origin (dispute resolution)
- The three alternative functions used:
 - hash function
 - message encryption
 - message authentication code (MAC)



Message Authentication

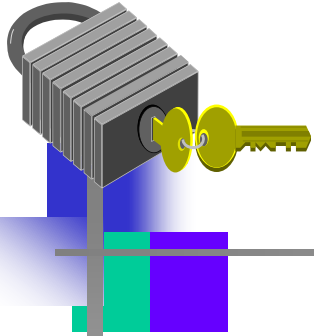
- Requirements - must be able to verify that:
 1. Message came from apparent source or author
 2. Contents have not been altered
 3. Timeliness - that it was sent at a certain time or sequence.
- Protection against active attack (falsification of data and transactions)

Approaches to Message Authentication



- Authentication Using Conventional Encryption
 - Only the sender and receiver should share a key
- Message Authentication without Message Encryption
 - An authentication tag is generated and appended to each message
- Message Authentication Code
 - Calculate the MAC as a function of the message and the key.

$$MAC = F(K, M)$$



Message Authentication

- **Using Encryption**
 - Assume only sender and receiver share a key
 - Then a correctly encrypted message should be from the sender
- Usually also contains error-detection code, sequence number and time stamp
- Encryption alone is not suitable for authentication. Blocks could have been reordered, changing meaning

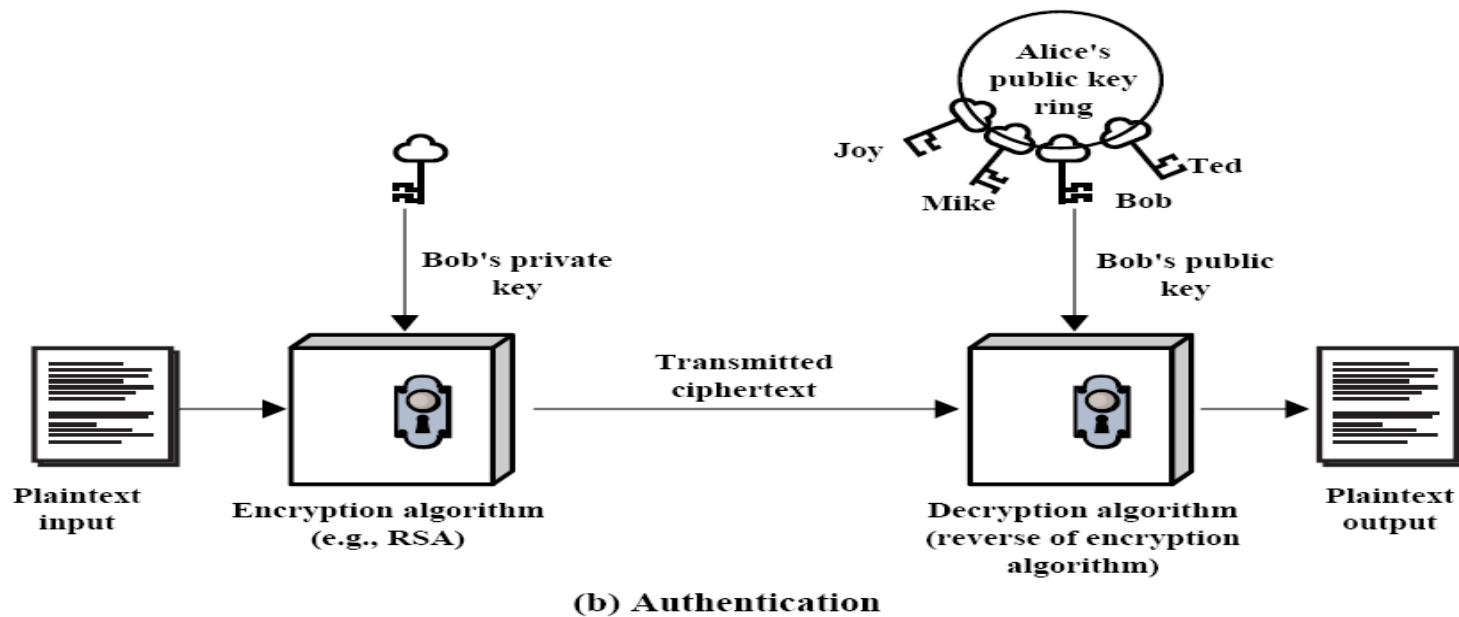
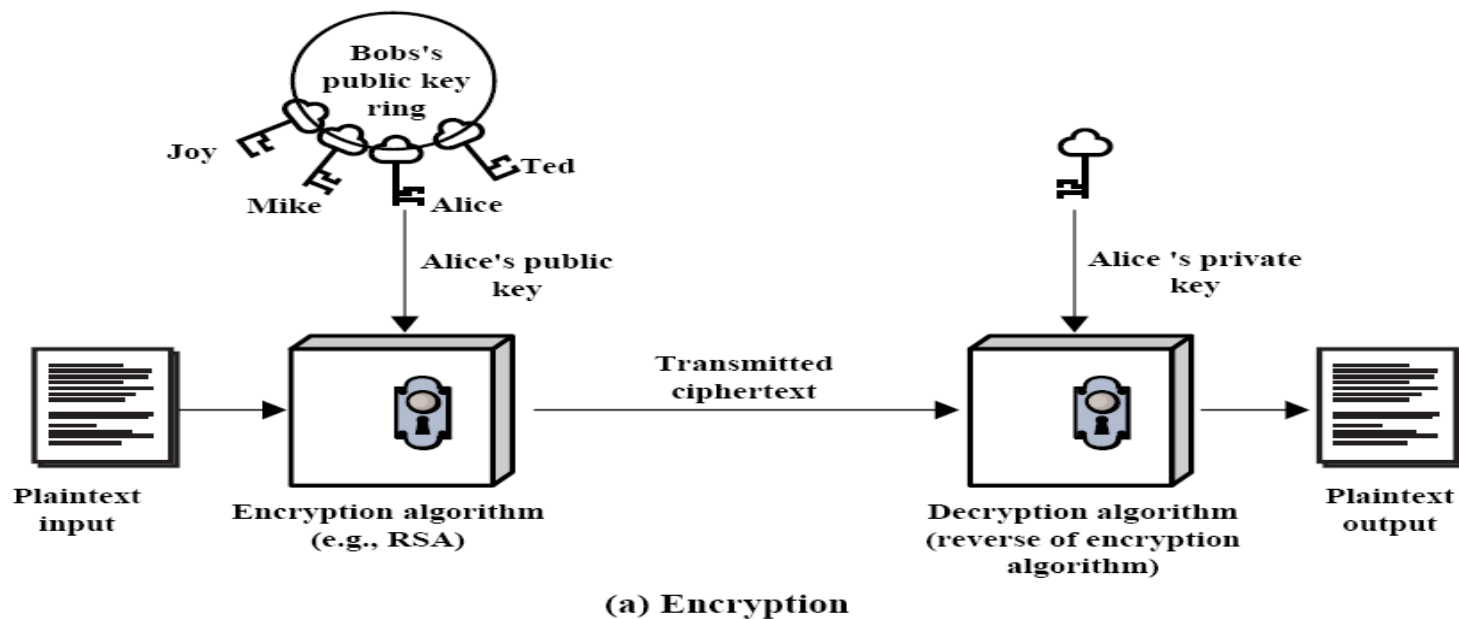
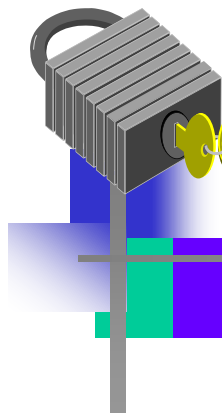
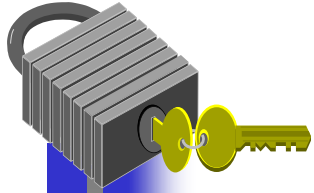


Figure 3.7 Public-Key Cryptography



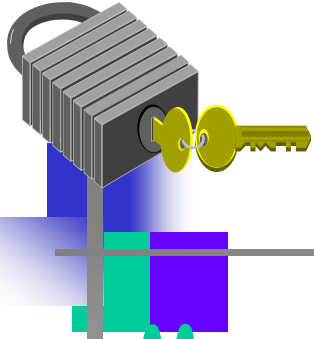
Message Authentication

Without Encryption

No confidentiality is preferred when:

1. Same message is broadcast to many destinations
2. Heavy load and cannot decrypt all messages
- some chosen at random
3. No danger in sending plaintext

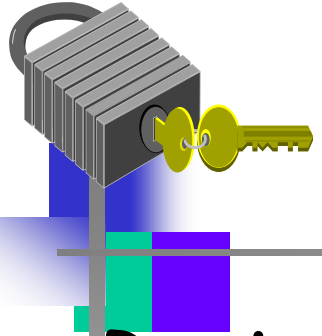
Append authentication tag to each message



Message Authentication

• Message Authentication Code (MAC)

- Small block of data that is appended to the message
- MAC is generated by using a secret key
- Assumes both parties A,B share common secret key K_{AB}
- Code is function of message and key
 $MAC_M = F(K_{AB}, M)$
- Message plus code are transmitted



Message Authentication Code

- Recipient uses key to compute new code
- If received code matches calculated code then
 - Receiver is sure message has not been altered
 - Message is from sender, since only sender shares the key
 - If the message includes correct sequence number, that number could not have been altered by hacker

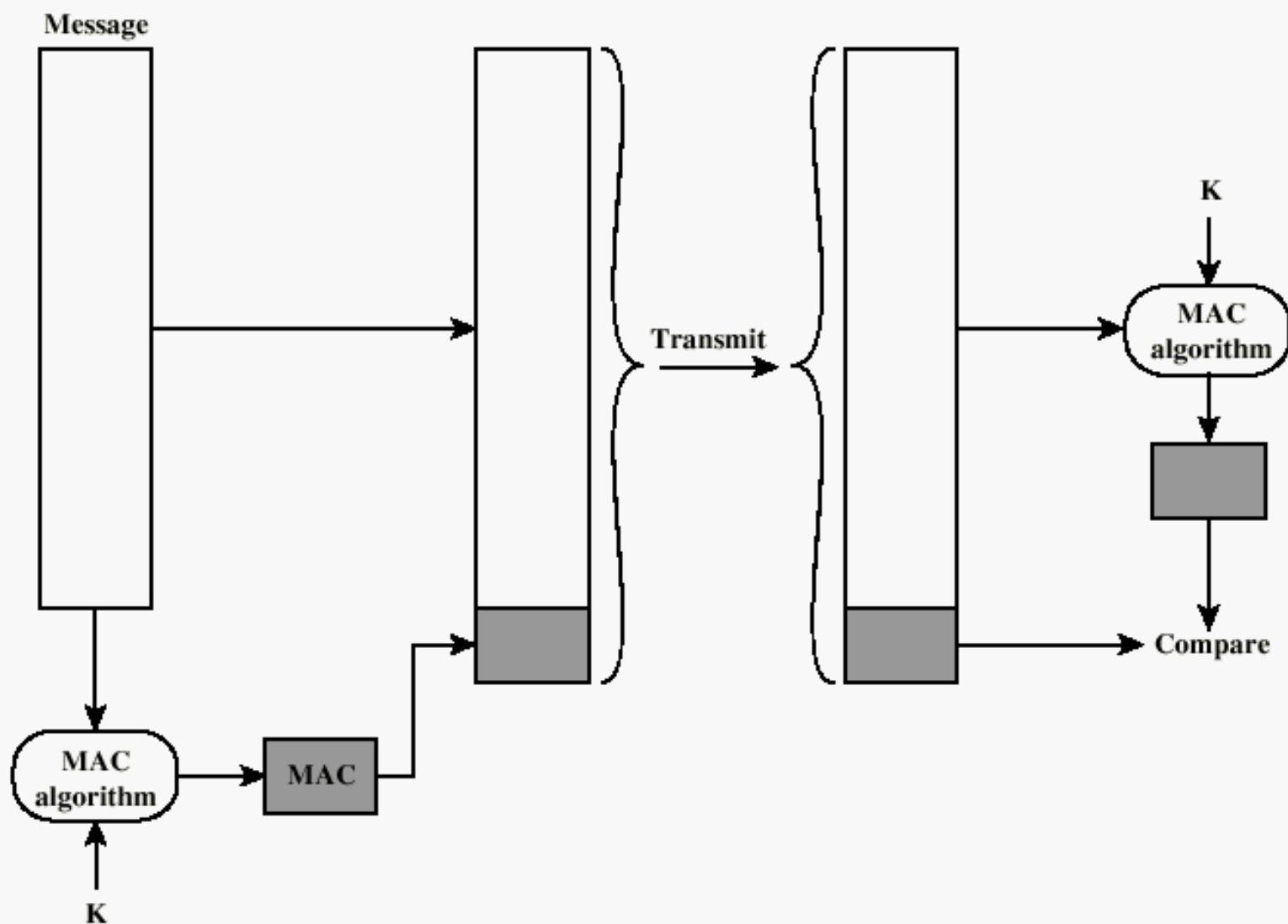
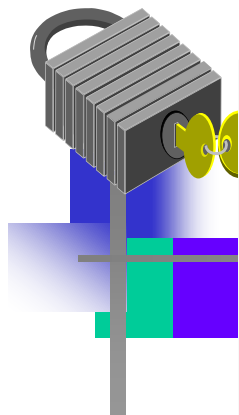
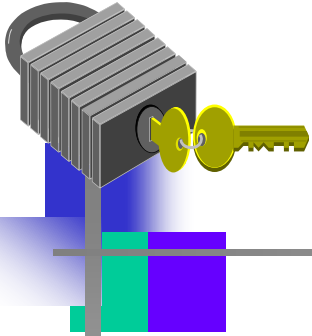
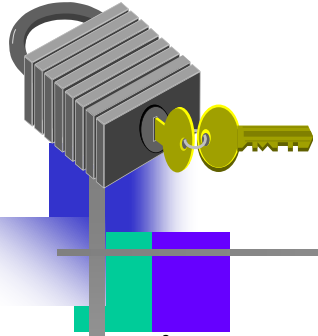


Figure 3.1 Message Authentication Using a Message Authentication Code (MAC)

Message Authentication Code

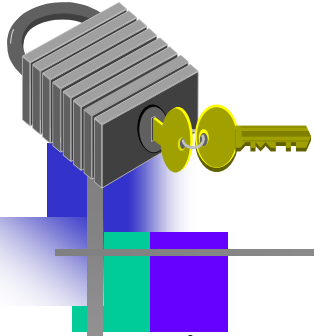


- Different from encryption
 - MAC does not have to be reversible as the cipher text does in encryption
 - Because of mathematical properties, it is less vulnerable to being broken than encryption
- 16 to 32 bit code is typical



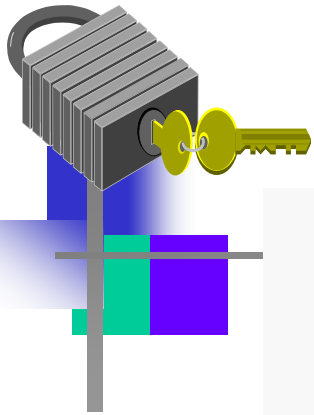
One-way HASH function

- Alternative to Message Authentication Code
- Accepts a variable size message M as input and produces a fixed-size message digest $H(M)$ as output
- Unlike the MAC, a hash function does not take a secret key as input
- Message digest also provides data integrity, since if bits are accidentally altered in transit, the message digest will also be in error.

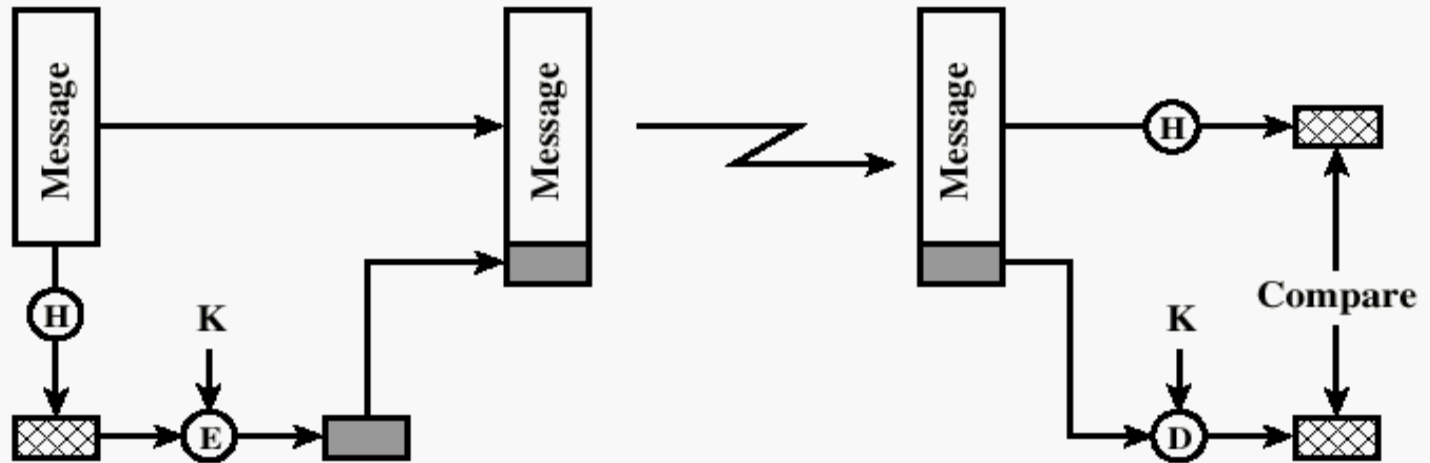


One-Way Hash Function

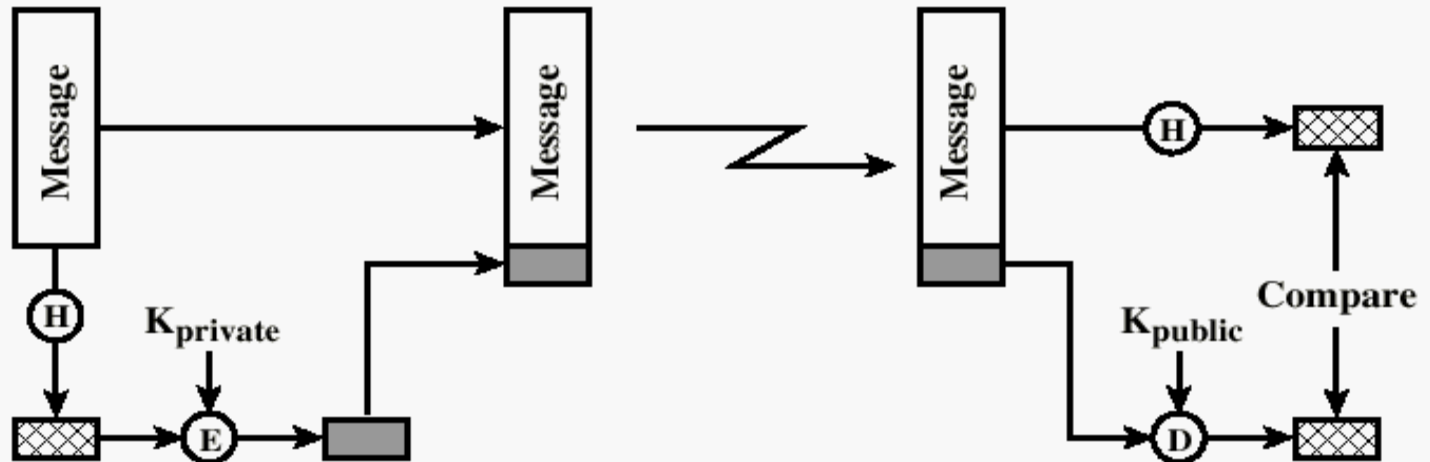
- The message can be authenticated:
- Using encryption using a shared secret key
- Using public-key encryption
 - Also provides a digital signature
 - Does not require key distribution
- Using a secret value



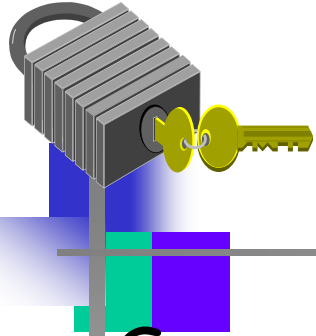
One-way HASH function



(a) Using conventional encryption

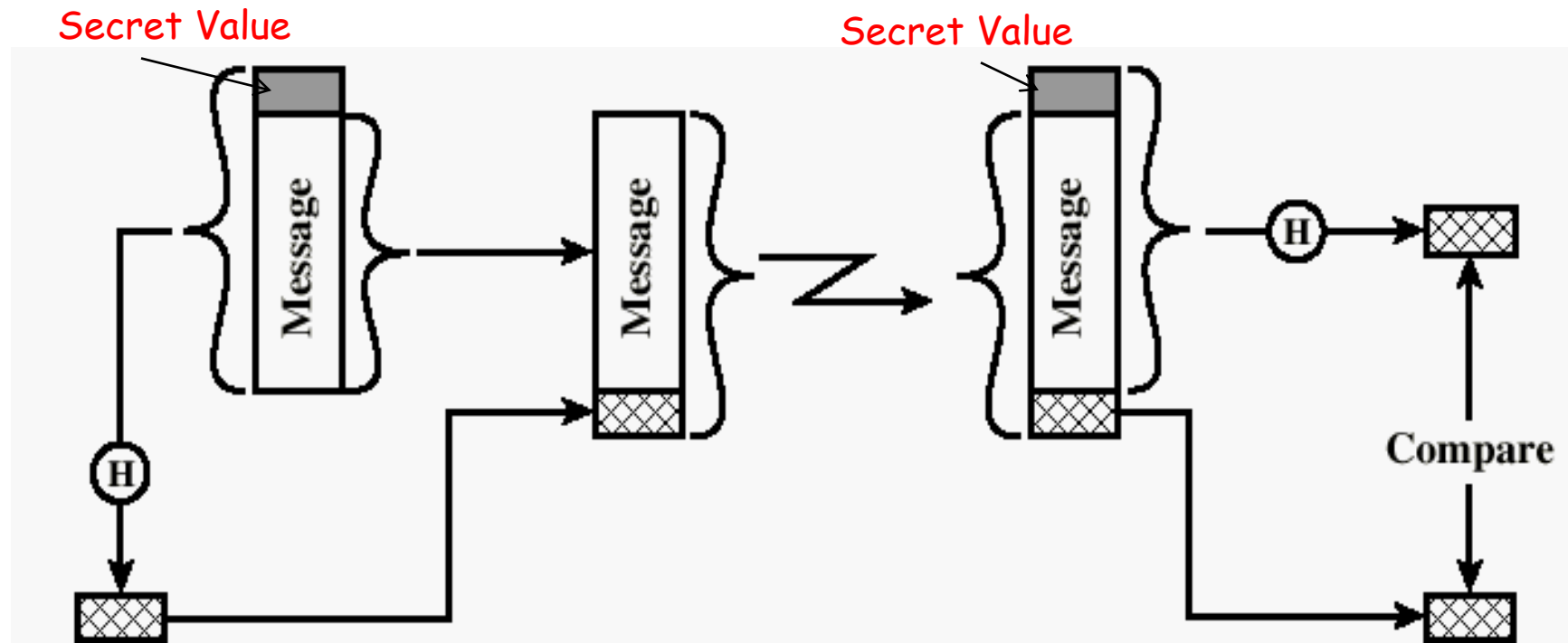


(b) Using public-key encryption

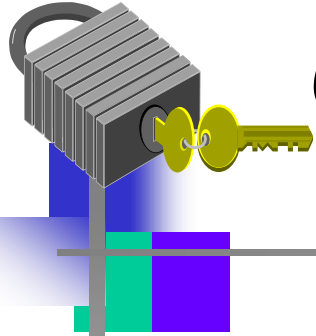


One-Way HASH Function

- Secret value is added before the hash and removed before transmission.

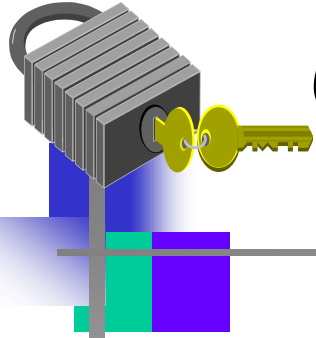


(c) Using secret value



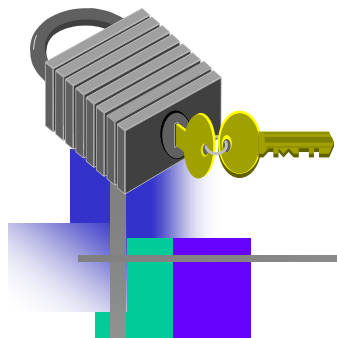
One-way HASH Function Advantages

- Using a hash function instead of encryption has advantages:
- Encryption is slow
- Encryption hardware can be expensive
- Encryption hardware is optimized for large data sets
- An encryption algorithm may be protected by a patent



One-way HASH Function

- The one-way hash function is a secure hash function
- It is important for authentication and is also used in digital signatures
- The most important hash function is SHA (Secure Hash Algorithm).

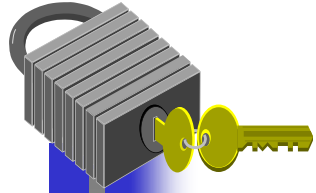


Hash Function

- Condenses arbitrary message to fixed size

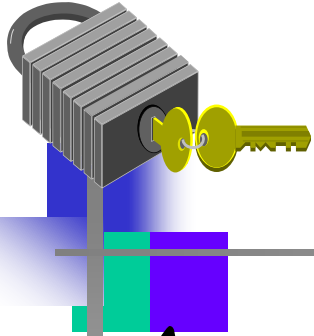
$$h = H(M)$$

- Usually assume hash function is public
- Hash used to detect changes to message
- Want a cryptographic hash function
 - computationally infeasible to find data mapping to specific hash (**one-way property**)
 - computationally infeasible to find two data to same hash (**collision-free property**)



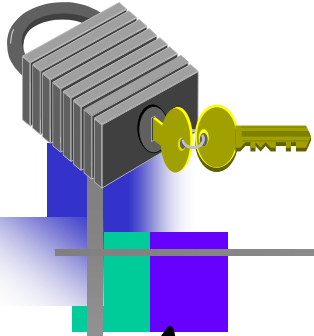
Secure HASH Functions

- Purpose of the HASH function is to produce a "fingerprint"
- Properties of a HASH function H :
 1. H can be applied to a block of data at any size
 2. H produces a fixed length output
 3. $H(x)$ is easy to compute for any given x .
 4. For any given block x , it is computationally infeasible to find x such that $H(x) = h$ (**one-way property**)
 5. For any given block x , it is computationally infeasible to find y with $H(y) = H(x)$. (**weak collision resistance**)
 6. It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$ (**strong collision resistance**)



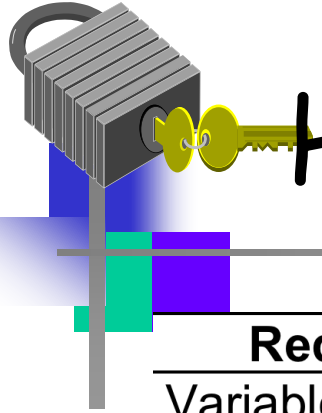
Simple Hash Function

- A weak hash function satisfies the first 5 properties.
- A strong hash function also satisfies the 6th property (**strong collision resistance**)
 - Effective against the birthday attack
- Message Digest provides both authentication and integrity



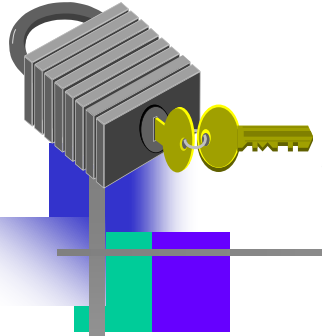
Simple Hash Function

- A weak hash function satisfies the first 5 properties.
- A strong hash function also satisfies the 6th property (**strong collision resistance**)
 - Effective against the birthday attack
- Message Digest provides both authentication and integrity



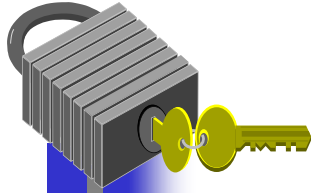
Hash Function Requirements

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



Security of Hash Functions

- Attacking a secure hash function can be done by using cryptanalysis or brute force.
- Strength of function depends on the length of the hash code produced by the algorithm.
- For example:
 - A search machine can find a collision for 128 bit code length in 24 days - considered inadequate
 - With 160 bits, finding a collision might take 4000 years (or less with today's speeds)



Simple Hash Function

- General principle

- Input is a sequence of n-bit blocks
- Input is processed one block at a time to produce an n-bit hash function
- A simple example is the XOR of each block

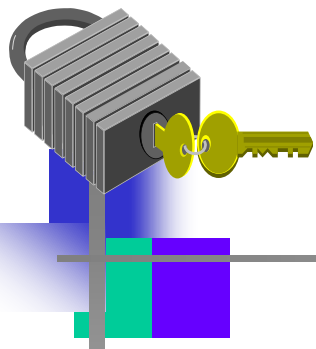
$$C_i = b_{i1} \bowtie b_{i2} \bowtie \dots \bowtie b_{im}$$

C_i is ith bit of hash code $1 \leq i \leq n$

m is number of n-bit block in input

b_{ij} is ith bit in jth block

\bowtie Is the XOR operation

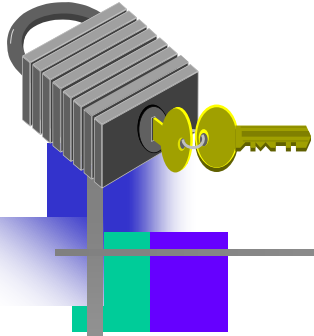


Simple Hash Function

	bit 1	bit 2	• • •	bit n
block 1	b_{11}	b_{21}		b_{n1}
block 2	b_{12}	b_{22}		b_{n2}
	•	•	•	•
	•	•	•	•
	•	•	•	•
block m	b_{1m}	b_{2m}		b_{nm}
hash code	C_1	C_2		C_n

Figure 3.3 Simple Hash Function Using Bitwise XOR

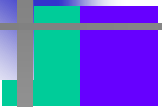
Simple Hash Function Improved

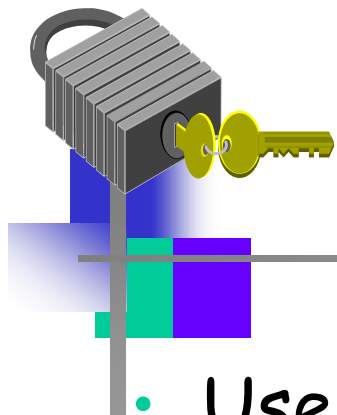


- To improve- perform a one-bit circular shift on the hash value after each block is processed
 - Initially set the n -bit hash value to zero
 - Process each successive n -bit block of data by:
 - Rotating current hash value to the left by 1 bit
 - XOR the block into the hash value
 - This has the effect of "randomizing" the input

Other Secure HASH functions

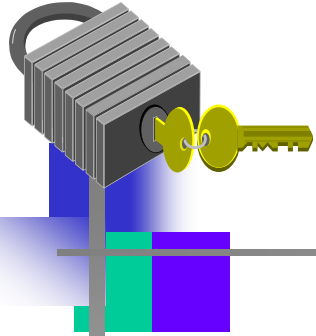


	SHA-1	MD5	RIPEMD-160
Digest length	160 bits	128 bits	160 bits
Basic unit of processing	512 bits	512 bits	512 bits
Number of steps	80 (4 rounds of 20)	64 (4 rounds of 16)	160 (5 paired rounds of 16)
Maximum message size	$2^{64}-1$ bits	∞	∞



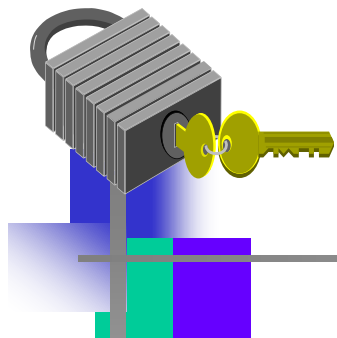
HMAC

- Use a MAC derived from a cryptographic hash code, such as SHA-1.
- **Motivations:**
 - Cryptographic hash functions execute faster in software than encryption algorithms such as DES
 - Library code for cryptographic hash functions is widely available
 - No export restrictions from the US



HMAC Design Objectives

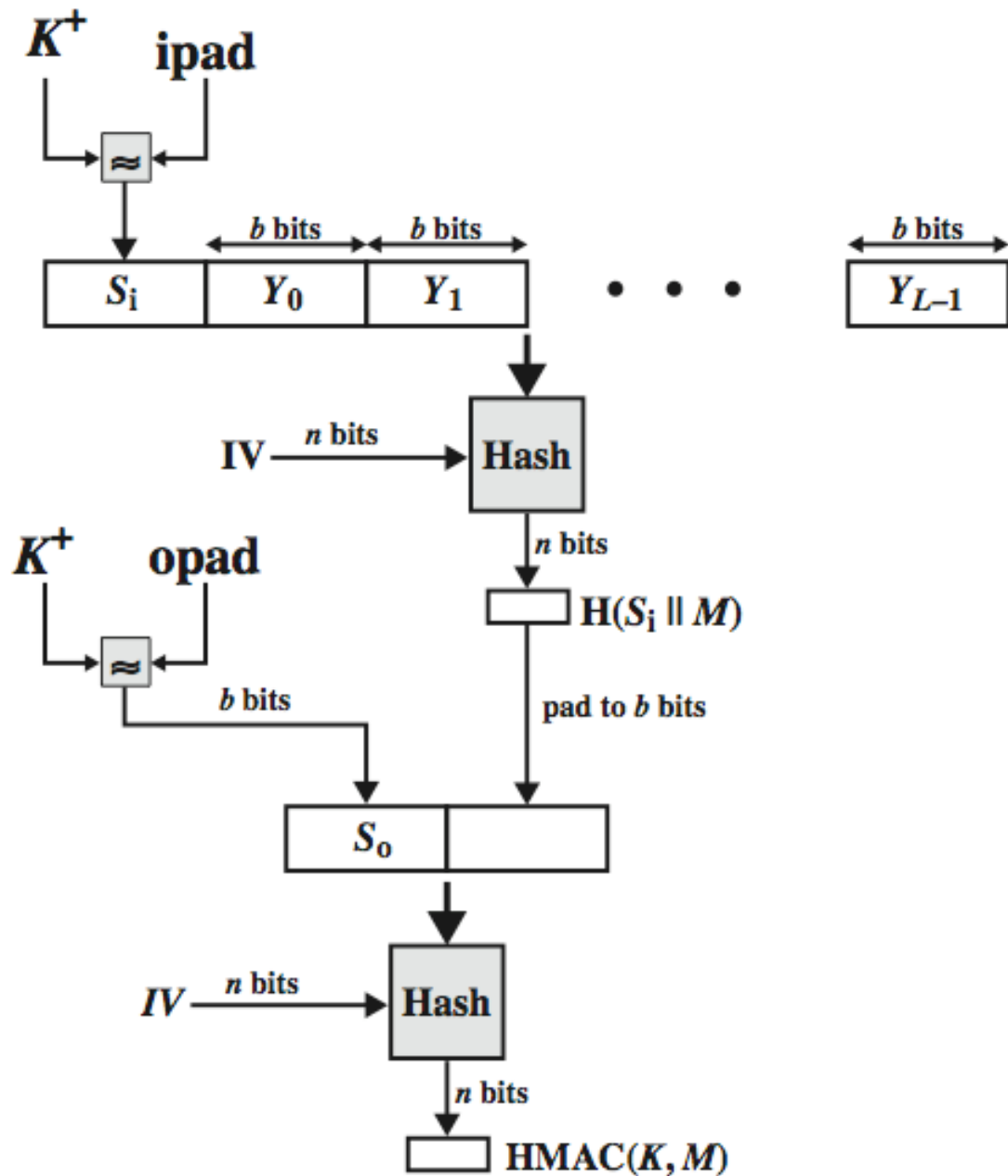
- Use, without modifications, hash functions
- Allow for easy replaceability of embedded hash function
- Preserve original performance of hash function without significant degradation
- Use and handle keys in a simple way.
- Have well understood cryptographic analysis of authentication mechanism strength

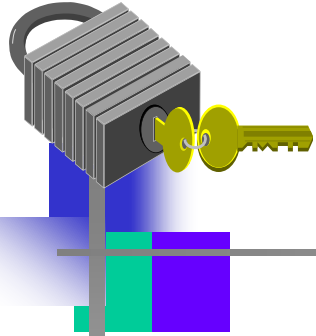


HMAC

- specified as Internet standard RFC2104
- uses hash function on the message:
$$\text{HMAC}_K(M) = \text{Hash}[(K^+ \text{ XOR opad}) || \text{Hash}[(K^+ \text{ XOR ipad}) || M]]$$
 - where K^+ is the key padded out to size
 - opad, ipad are specified padding constants
- overhead is just 3 more hash calculations than the message needs alone
- any hash function can be used
 - eg. MD5, SHA-1, RIPEMD-160, Whirlpool

HMAC Structure

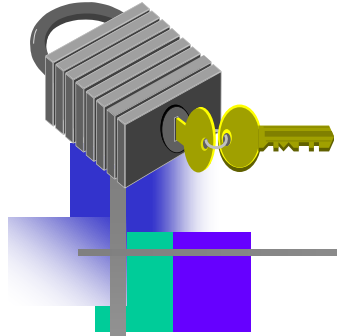




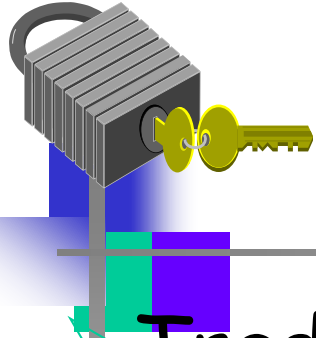
HMAC Security

- 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

CipherBased MAC (CMAC)

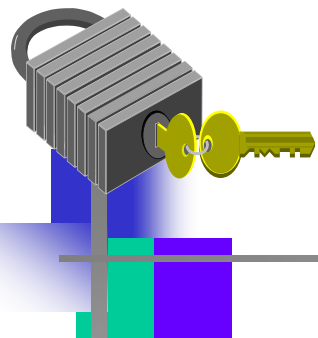


- Based on use of block cipher
- Widely used in government and industry
- Has message size limitation (nb, where $b=128$ for AES, $b=64$ for 3DES)
- Can overcome using 2 keys & padding
- Thus forming the Cipher-based Message Authentication Code (CMAC)
- Adopted by NIST SP800-38B



Private Key Cryptography

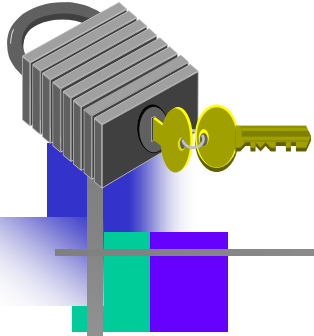
- Traditional **private/secret/single key** cryptography uses **one key**
- Shared by both sender and receiver
- If this key is disclosed communications are compromised
- Also is **symmetric**, parties are equal
- Does not protect sender from receiver forging a message and claiming it is sent by sender



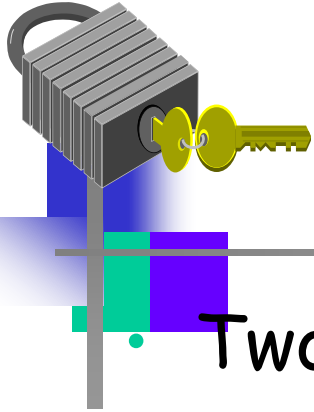
Public-Key Cryptography Principles

- Probably most significant advance in the 3000 year history of cryptography
- Uses **two** keys - a public and a private key
- **Asymmetric** since parties are **not** equal
- Uses clever application of number theoretic concepts to function
- Complements **rather than** replaces private key cryptography (slower)

Public-Key Cryptography Features

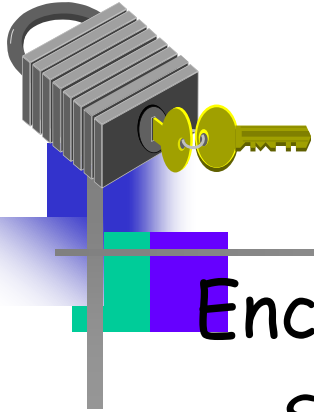


- Knowing the public key, anyone can encrypt messages or verify signatures, but cannot decrypt messages or create signatures
- Use of two keys has consequences in the areas of confidentiality, key distribution, and authentication.
- Based on mathematical functions rather than on operations on bit patterns.



Public Key Cryptography

- Two keys
 - Sender uses recipient's **public key** to encrypt
 - Recipient uses **private key** to decrypt
- Based on "trap door one way function"
 - "One way" means easy to compute in one direction, but hard to compute in other direction
 - Example: Given p and q , product $N = pq$ easy to compute, but given N , it's hard to find p and q
 - "Trap door" used to create key pairs



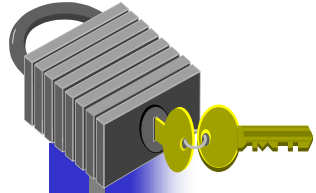
Public Key Cryptography

Encryption

- Suppose we **encrypt** M with Bob's public key
- Bob's private key can **decrypt** to recover M

- Digital Signature

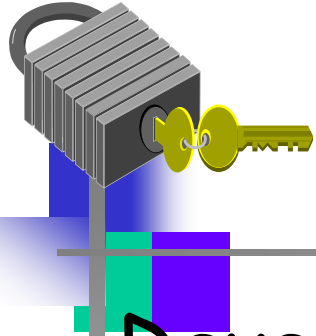
- **Sign** by "encrypting" with your private key
- Anyone can **verify** signature by "decrypting" with public key
- But only you could have signed
- Like a handwritten signature, but way better...



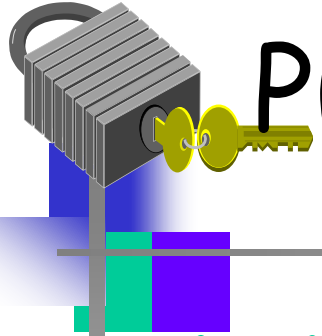
Misconceptions

- Public-key encryption is more secure
 - Security of any scheme depends on:
 - Length of the key
 - Computational work involved in breaking the cipher
- Public-key makes conventional encryption obsolete:
 - Computational overhead of public-key encryption is high
 - Both require similar key distribution protocols
- Both encryption schemes can offer security

Why Public-Key Cryptography?

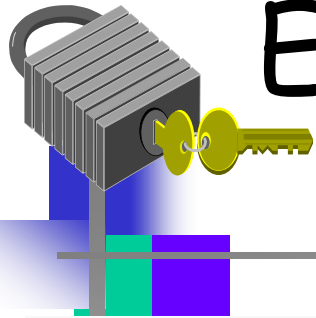


- Developed by Whitfield **Diffie** & Martin **Hellman** at Stanford University in 1976
 - known earlier in classified community
- Developed to address two key issues:
 - **key distribution** - how to have secure communications in general without having to trust a Key Distribution Center with your key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

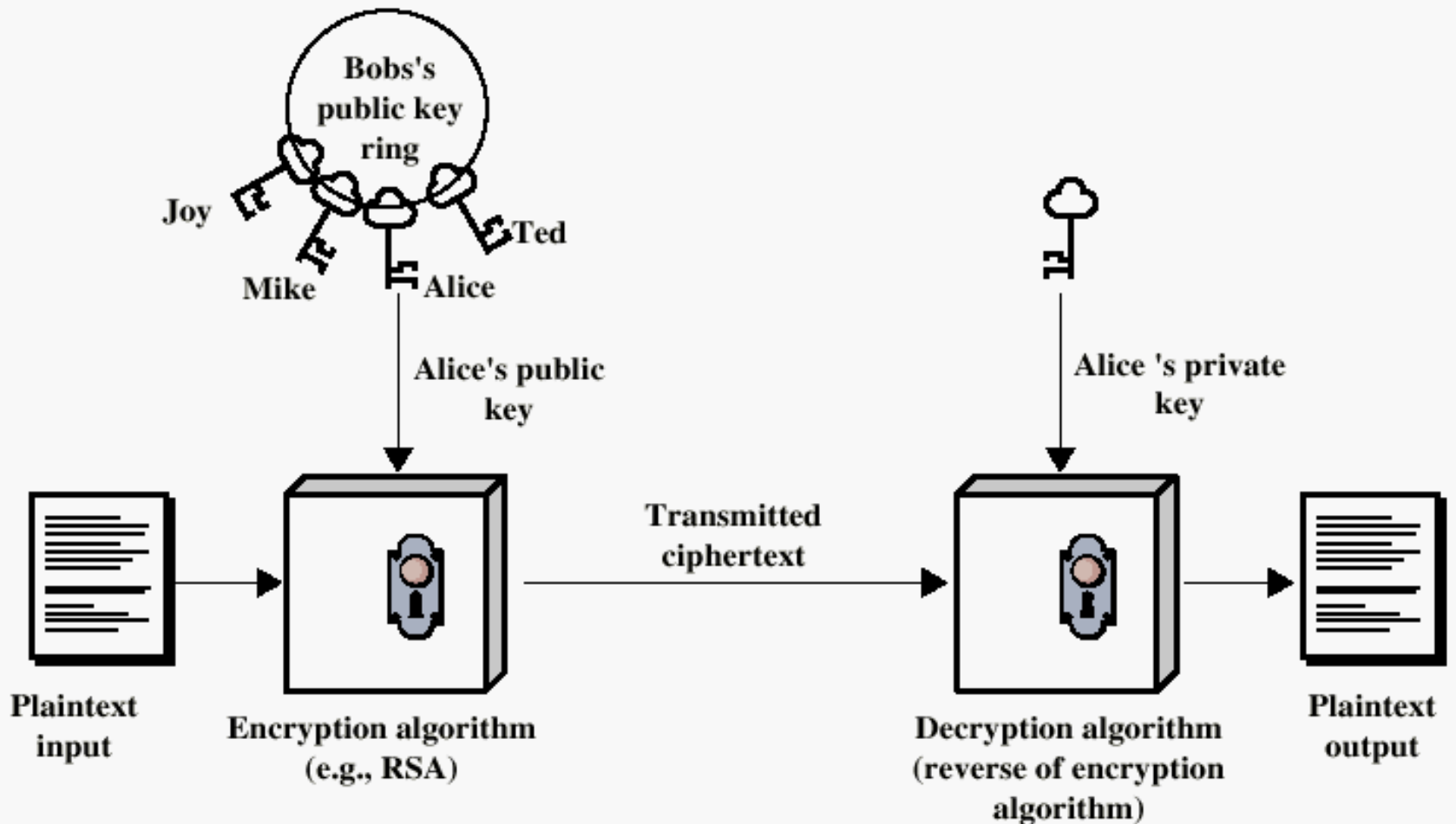


Public-Key Cryptography Principles

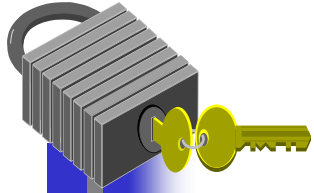
- **Public key** is made public for others to use, to encrypt messages and verify signatures
- **Private key** is known only to owner and is used to decrypt messages and create signatures
- The scheme has six ingredients:
 - **Plaintext** - readable message
 - **Encryption algorithm** - transforms plaintext into ciphertext
 - **Public and private key**
 - **Ciphertext** - scrambled message - output
 - **Decryption algorithm** - reverse of encryption



Encryption using Public-Key system

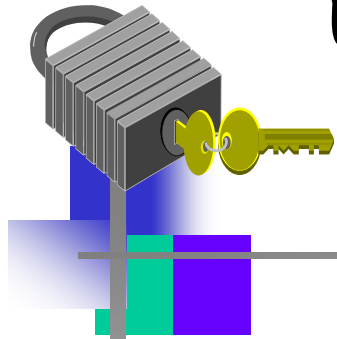


Essential Steps in Encryption



1. Each user generates a pair of keys to be used for encryption/decryption
2. Each user places public key in a public register or file
3. To send a private message to A, B encrypts the message using A's public key
4. When A receives the message, A uses her private key to decrypt it. (No one else can decrypt it, without that private key.)

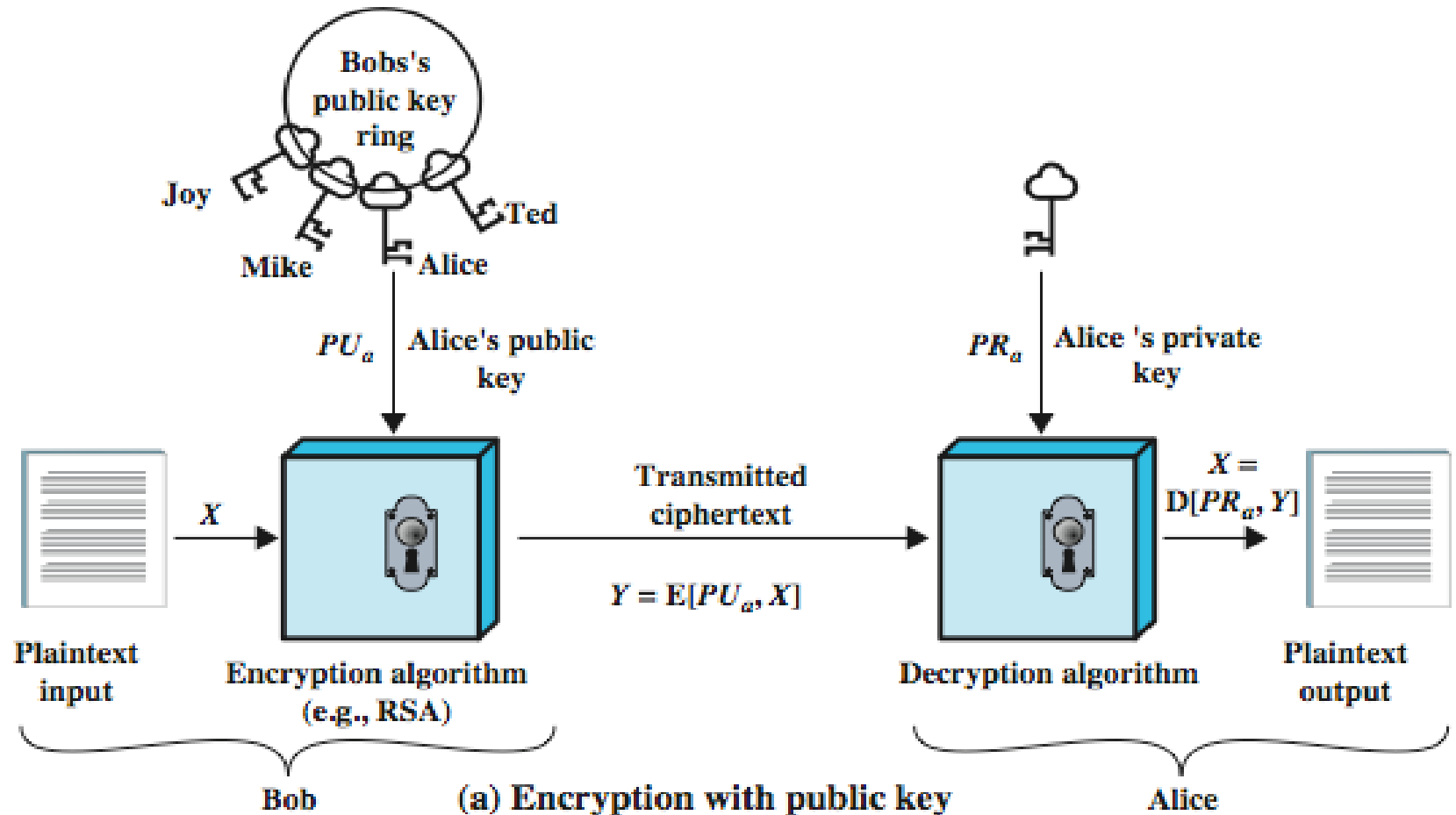
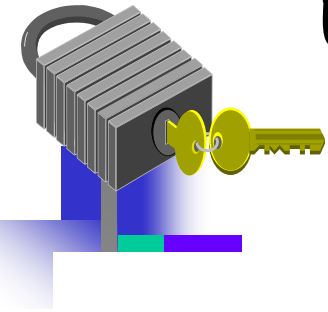
Encryption Using Public-Key System

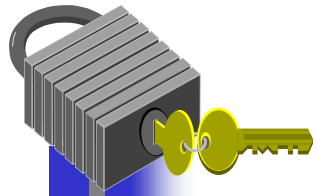


Features:

- All participants have access to public keys
- Private keys are generated locally and do not need to be distributed
- As long as a user protects the private key, incoming communication is secure
- A user can change keys at any time and re-publish the public key.

Encryption Using Public-Key System

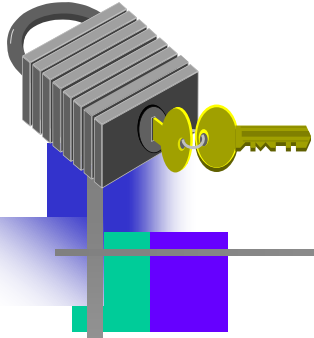




Symmetric vs Public-Key

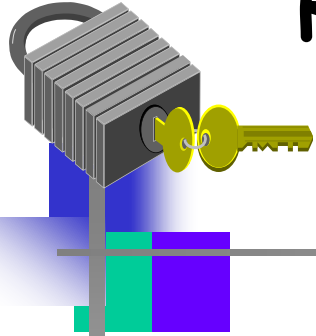
Conventional Encryption	Public-Key Encryption
<i>Needed to Work:</i> <ol style="list-style-type: none">1. The same algorithm with the same key is used for encryption and decryption.2. The sender and receiver must share the algorithm and the key. <i>Needed for Security:</i> <ol style="list-style-type: none">1. The key must be kept secret.2. It must be impossible or at least impractical to decipher a message if no other information is available.3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	<i>Needed to Work:</i> <ol style="list-style-type: none">1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption.2. The sender and receiver must each have one of the matched pair of keys (not the same one). <i>Needed for Security:</i> <ol style="list-style-type: none">1. One of the two keys must be kept secret.2. It must be impossible or at least impractical to decipher a message if no other information is available.3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

Applications for Public-Key Cryptosystems



- Three categories:
 - **Encryption/decryption:** The sender encrypts a message with the recipient's public key.
 - **Digital signature:** The sender "signs" a message with its private key.
 - **Key exchange:** Two sides cooperate to exchange a session key.

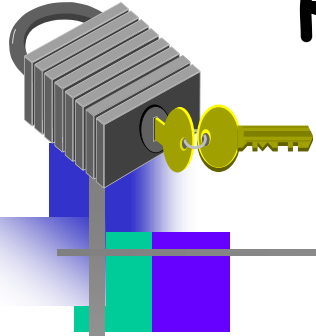
Requirements for Public Key Cryptography



1. Computationally easy for a party B to generate a pair (public key PU_b , private key PR_b)
2. Easy for sender to generate ciphertext
$$C = E(PU_b, M)$$
3. Easy for the receiver to decrypt ciphertext using private key:

$$M = D(PR_b, C) = D[PR_b, E(PU_b, M)]$$

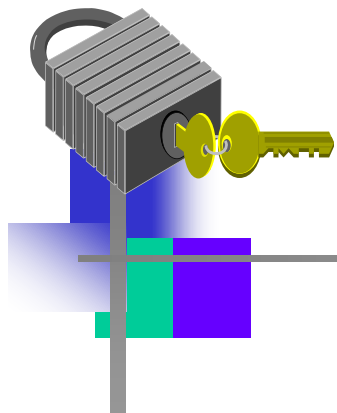
Requirements for Public Key Cryptography



4. Computationally infeasible to determine private key (PR_b) knowing public key (PU_b)
5. Computationally infeasible to recover message M , knowing PU_b and ciphertext C
6. *Either of the two keys can be used for encryption, with the other used for decryption:

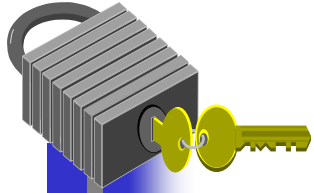
$$M = D[PU_b, E(PR_b, M)] = D[PR_b, E(PU_b, M)]$$

- * 6th requirement is useful but not necessary



Knapsack





Knapsack Problem

Given a set of n weights W_0, W_1, \dots, W_{n-1} and a sum S , is it possible to find $a_i \in \{0, 1\}$ so that

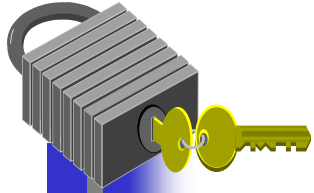
$$S = a_0W_0 + a_1W_1 + \dots + a_{n-1}W_{n-1}$$

(technically, this is “subset sum” problem)

- **Example**

- Weights (62, 93, 26, 52, 166, 48, 91, 141)
- Problem: Find subset that sums to $S=302$
- Answer: $62+26+166+48=302$

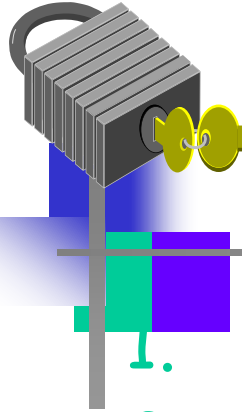
- The (general) knapsack is NP-complete



Knapsack Problem

General knapsack (GK) is hard to solve

- But **superincreasing knapsack** (SIK) is easy
- SIK: each weight greater than the *sum of all previous weights*
- **Example**
 - Weights (2,3,7,14,30,57,120,251)
 - Problem: Find subset that sums to $S=186$
 - Work from largest to smallest weight
 - Answer: $120+57+7+2=186$



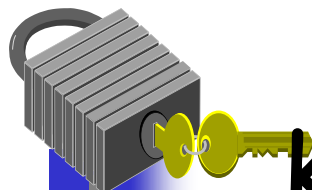
Knapsack Cryptosystem

1. Generate superincreasing knapsack (SIK)
 2. Convert SIK into "general" knapsack (GK)
 3. **Public Key:** GK
 4. **Private Key:** SIK plus conversion factor
- Ideally...
- Easy to encrypt with GK
 - With private key, easy to decrypt (convert ciphertext to SIK problem)
 - Without private key, must solve GK



Knapsack Keys

- Start with (2,3,7,14,30,57,120,251) as the SIK
 - Choose $m = 41$ and $n = 491$ (m, n relatively prime, n exceeds sum of elements in SIK)
- Compute "general" knapsack
 - $2 \cdot 41 \bmod 491 = 82$
 - $3 \cdot 41 \bmod 491 = 123$
 - $7 \cdot 41 \bmod 491 = 287$
 - $14 \cdot 41 \bmod 491 = 83$
 - $30 \cdot 41 \bmod 491 = 248$
 - $57 \cdot 41 \bmod 491 = 373$
 - $120 \cdot 41 \bmod 491 = 10$
 - $251 \cdot 41 \bmod 491 = 471$
- "General" knapsack: (82,123,287,83,248,373,10,471)

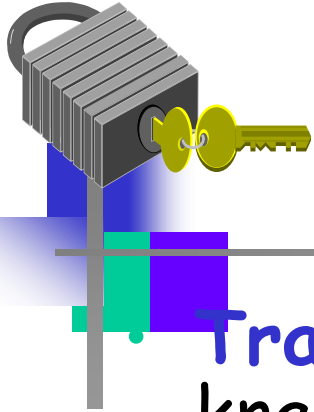


Knapsack Cryptosystem

- **Private key:** (2,3,7,14,30,57,120,251)

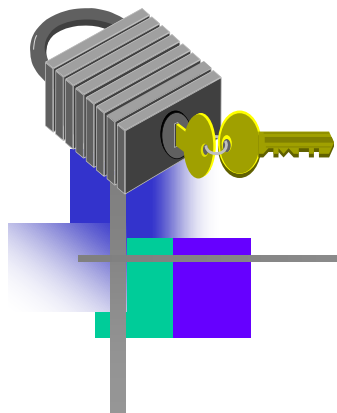
$$m^{-1} \bmod n = 41^{-1} \bmod 491 = 12$$

- **Public key:** (82,123,287,83,248,373,10,471),
n=491
- Example: Encrypt 10010110
 $82 + 83 + 373 + 10 = 548$
- To decrypt,
 - $548 \cdot 12 = 193 \bmod 491$
 - Solve (easy) SIK with $S = 193$
 - Obtain plaintext 10010110

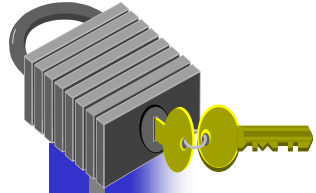


Knapsack Weakness

- **Trapdoor:** Convert SIK into "general" knapsack using modular arithmetic
- **One-way:** General knapsack easy to encrypt, hard to solve; SIK easy to solve
- This knapsack cryptosystem is **insecure**
 - Broken in 1983 with Apple II computer
 - The attack uses **lattice reduction**
- "General knapsack" is not general enough!
- This special knapsack is easy to solve!



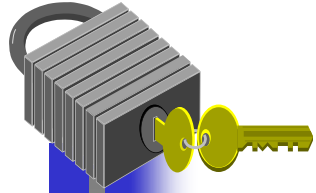
RSA



RSA

By Clifford Cocks (GCHQ), independently, Rivest, Shamir, and Adleman (MIT)

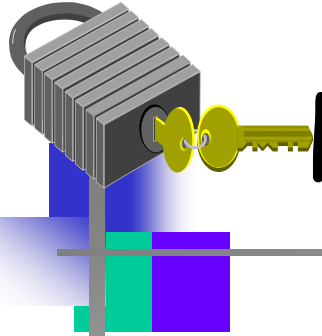
- RSA is the gold standard in public key crypto
- Let p and q be two large prime numbers
- Let $N = pq$ be the modulus
- Choose e relatively prime to $(p-1)(q-1)$
- Find d such that $ed = 1 \bmod (p-1)(q-1)$
- **Public key** is (N, e)
- **Private key** is d



RSA

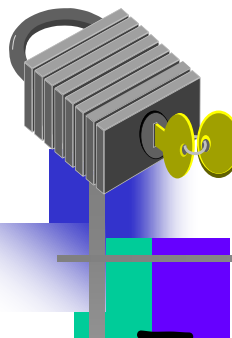
Message M is treated as a number

- To encrypt M we compute
$$C = M^e \bmod N$$
- To decrypt ciphertext C compute
$$M = C^d \bmod N$$
- Recall that e and N are public
- If Trudy can factor $N=pq$, she can use e to easily find d since $ed = 1 \bmod (p-1)(q-1)$
- **Factoring the modulus breaks RSA**
 - Is factoring the only way to break RSA?



Public Key Algorithms- RSA

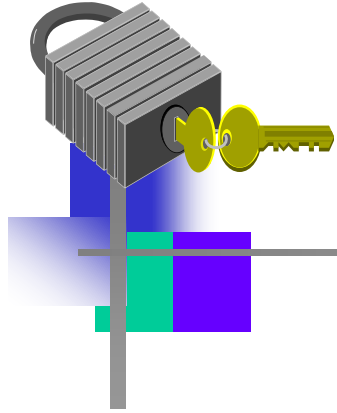
- Most popular and widely implemented
- Block cipher and each block has binary value $< n$
- Plain text and cipher text are integers between 0 and $(n-1)$ for some n
- Both sender and receiver know n
- $C = M^e \bmod n$
- $M = C^d \bmod n = (M^e \bmod n)^d = (M^{ed} \bmod n)$



RSA Encryption/Decryption

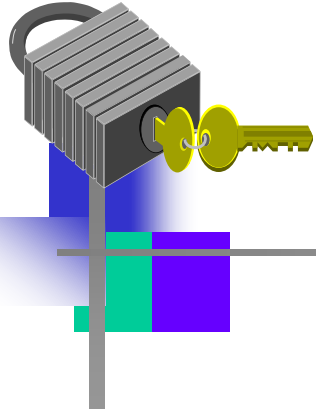
- To encrypt a message M the sender:
 - obtains **public key** of recipient $PU=\{e, n\}$
 - computes: $C = M^e \bmod n$, where $0 \leq M < n$
- To decrypt the ciphertext C the owner:
 - uses their **private key** $PR=\{d, n\}$
 - computes: $M = C^d \bmod n$
- Note that the message M must be smaller than the modulus n (block if needed)

The RSA Algorithm - Encryption

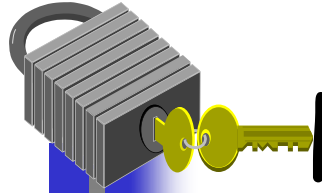


- Plaintext: $M < n$
- Ciphertext: $C = M^e \pmod n$

The RSA Algorithm - Decryption

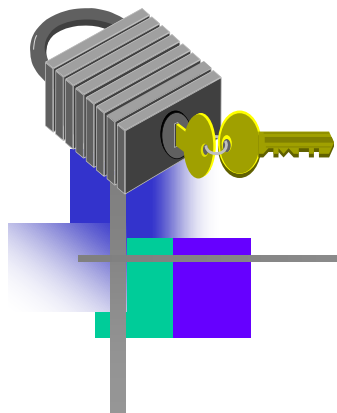


- Ciphertext: C
- Plaintext: $M = C^d \pmod{n}$



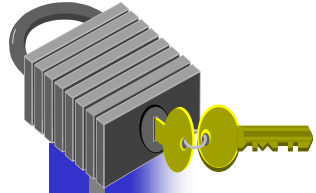
Public Key Algorithms- RSA

- Both sender and receiver know n and e
- Only receiver knows d
- Public key = $\{e, n\}$
- Private key = $\{d, n\}$
- Requirements
 - Should be possible to find e, d, n such that $M^{ed} = M \bmod n$ for all $M < n$
 - Relatively easy to calculate M^e, C^d for all $M < n$
 - Should be infeasible to determine d , given e and n



Simple Examples

RSA



Simple RSA Example

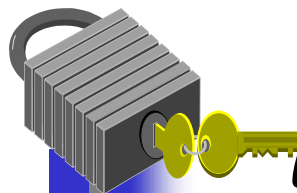
Example of RSA

- Select "large" primes $p = 11$, $q = 3$
- Then $N = pq = 33$ and $(p - 1)(q - 1) = 20$
- Choose $e = 3$ (relatively prime to 20)
- Find d such that $ed = 1 \pmod{20}$
 - We find that $d = 7$ works
- **Public key:** $(N, e) = (33, 3)$
- **Private key:** $d = 7$



Simple RSA Example

- **Public key:** $(N, e) = (33, 3)$
- **Private key:** $d = 7$
- Suppose message $M = 8$
- Ciphertext C is computed as
$$C = M^e \bmod N = 8^3 = 512 = 17 \bmod 33$$
- Decrypt C to recover the message M by
$$\begin{aligned} M &= C^d \bmod N = 17^7 = 410,338,673 \\ &= 12,434,505 * 33 + 8 = 8 \bmod 33 \end{aligned}$$



More Efficient RSA (1)

Modular exponentiation example

- $5^{20} = 95367431640625 = 25 \text{ mod } 35$

- A better way: **repeated squaring**

- $20 = 10100 \text{ base } 2$

- $(1, 10, 101, 1010, 10100) = (1, 2, 5, 10, 20)$

- Note that $2 = 1 \cdot 2$, $5 = 2 \cdot 2 + 1$, $10 = 2 \cdot 5$, $20 = 2 \cdot 10$

- $5^1 = 5 \text{ mod } 35$

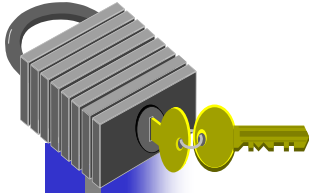
- $5^2 = (5^1)^2 = 5^2 = 25 \text{ mod } 35$

- $5^5 = (5^2)^2 \cdot 5^1 = 25^2 \cdot 5 = 3125 = 10 \text{ mod } 35$

- $5^{10} = (5^5)^2 = 10^2 = 100 = 30 \text{ mod } 35$

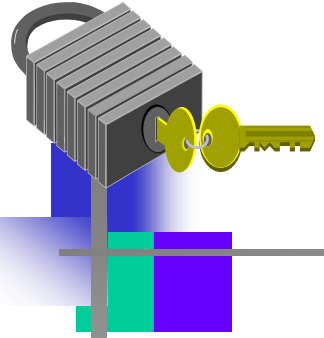
- $5^{20} = (5^{10})^2 = 30^2 = 900 = 25 \text{ mod } 35$

- No huge numbers and it's efficient!



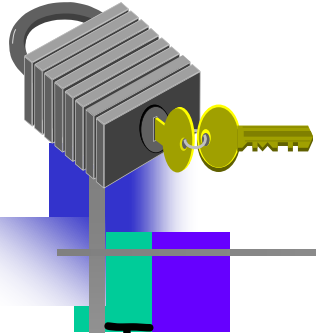
More Efficient RSA (2)

- Use $e = 3$ for all users (but not same N or d)
 - + Public key operations only require 2 multiplies
 - Private key operations remain expensive
 - If $M < N^{1/3}$ then $C = M^e = M^3$ and **cube root attack**
 - For any M , if C_1, C_2, C_3 sent to 3 users, cube root attack works (uses Chinese Remainder Theorem)
- Can prevent cube root attack by padding message with random bits
- Note: $e = 2^{16} + 1$ also used
("better" than $e = 3$)



Requirements continued...

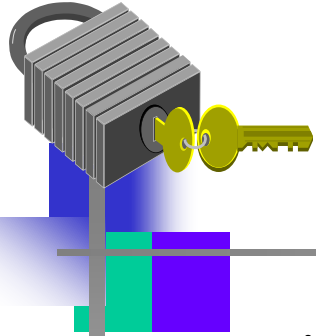
- Easy to calculate M^e and C^d for all $M < n$
- Infeasible to determine d , given e and n
- First two requirements are easy.
- Third one is also possible if e, n are large



Defeating RSA

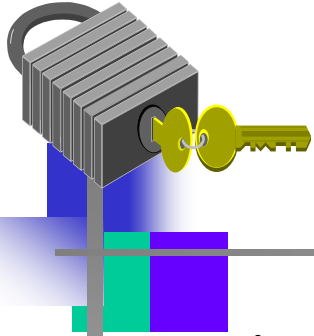
- Two approaches:
 1. Brute force - try all possible private keys
 - Larger number of bits in e and d , the more secure the algorithm
 - But the larger the size of the key the slower the algorithm will run
 2. Cryptanalysis- factoring n into two primes (hard)
 - Challenge -1994- 1600 computers -solved in 8 months with $n = 129$ digits (428 bits) - need larger key size

The RSA Algorithm - Key Generation

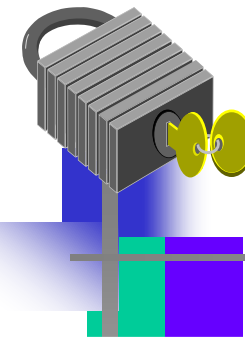


1. Select p, q p and q both prime
2. Calculate $n = p \times q$
3. Calculate $\phi(n) = (p-1)(q-1)$
4. Select integer e e relatively prime to $\phi(n)$
5. Calculate d such that $de \bmod \phi(n) = 1$
6. Public Key $KU = \{e, n\}$
7. Private key $KR = \{d, n\}$

The RSA Algorithm - Key Generation Example



1. Select $p=17, q=11$ p and q both prime
2. Calculate $n = p \times q = 17 \times 11 = 187$
3. Calculate $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$
4. Select integer e relatively prime to $\phi(n)$ and $e < \phi(n)$: let $e = 7$
5. Calculate d : $de \bmod \phi(n) = 1$ or $de \bmod 160 = 1$ and < 160 because $23 \times 7 = 161 = (1 \times 160) + 1$
6. Publish Public Key $PU = \{e, n\} = \{7, 187\}$
7. Keep secret Private Key $PR = \{d, n\} = \{23, 187\}$



Example of RSA Algorithm

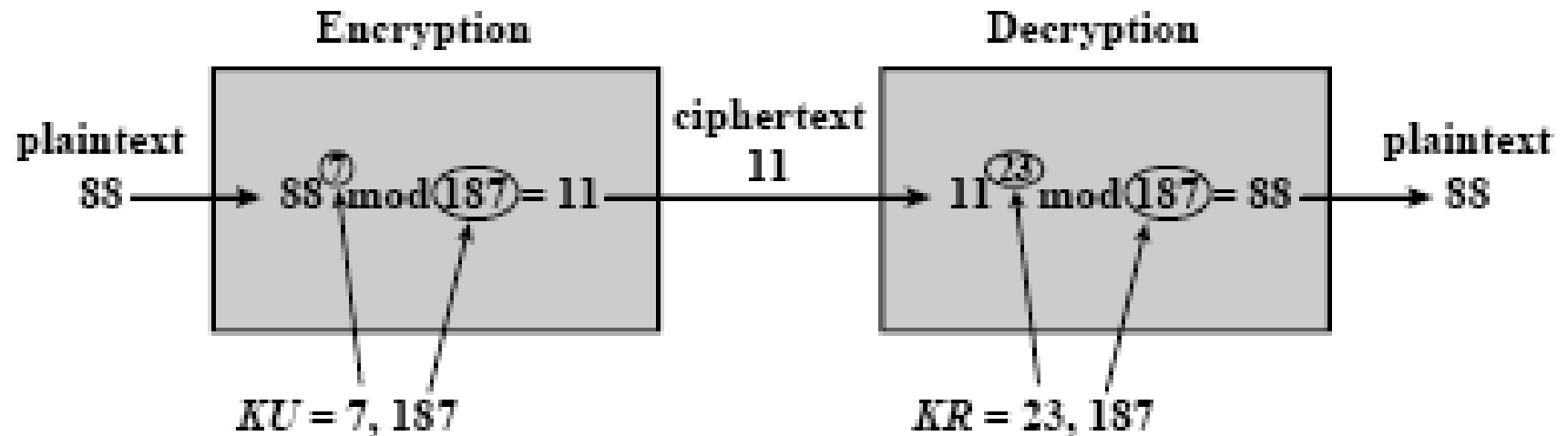
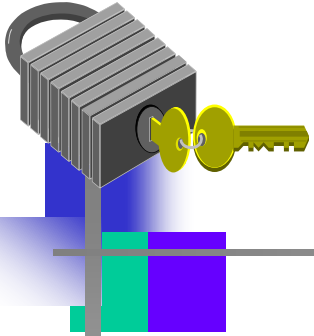


Figure 3.9 Example of RSA Algorithm

RSA Algorithm

Encryption/Decryption



➤ Sample RSA encryption/decryption is:

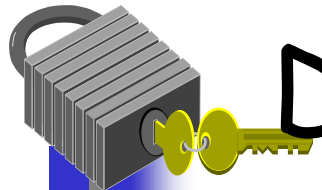
➤ Given message $M = 88$ (note $88 < 187$)

➤ **Encryption:**

$$C = 88^7 \bmod 187 = 11$$

➤ **Decryption:**

$$M = 11^{23} \bmod 187 = 88$$



Does RSA Really Work?

Given $C = M^e \bmod N$ we must show

$$M = C^d \bmod N = M^{ed} \bmod N$$

- We'll use **Euler's Theorem**:

If x is relatively prime to n then $x^{\phi(n)} = 1 \bmod n$

- Facts:

1) $ed = 1 \bmod (p - 1)(q - 1)$

2) By definition of "mod", $ed = k(p - 1)(q - 1) + 1$

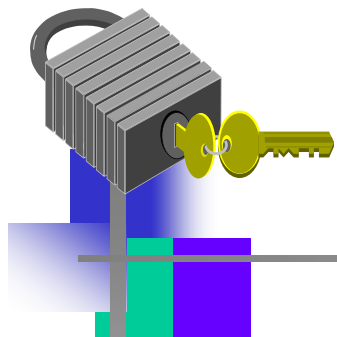
3) $\phi(N) = (p - 1)(q - 1)$

- Then $ed - 1 = k(p - 1)(q - 1) = k\phi(N)$

- Finally, $M^{ed} = M^{(ed - 1) + 1} = M \cdot M^{ed - 1} = M \cdot M^{k\phi(N)}$

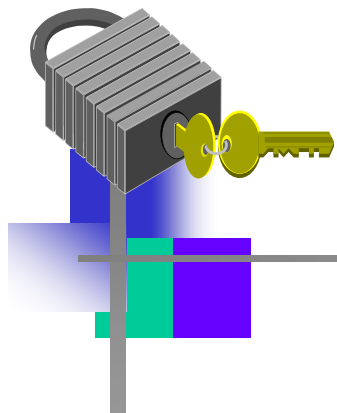
$$= M \cdot (M^{\phi(N)})^k \bmod N = M \cdot 1^k \bmod N$$

- $= \mathbf{M \bmod N}$



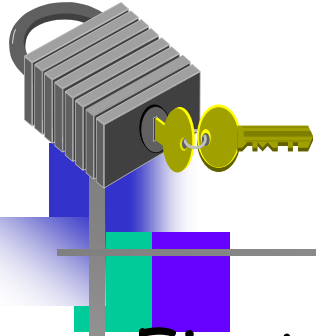
RSA

- Based on exponentiation in a finite (Galois) field over integers modulo a prime
 - exponentiation takes $O((\log n)^3)$ operations (easy)
- Uses large integers (eg. 1024 bits)
- Security due to cost of factoring large numbers
 - factorization takes $O(e^{\log n \log \log n})$ operations (hard)
- Still considered strong enough for most applications



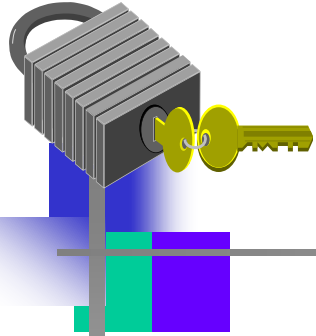
Diffie-Hellman

Public Key Algorithm Diffie-Hellman

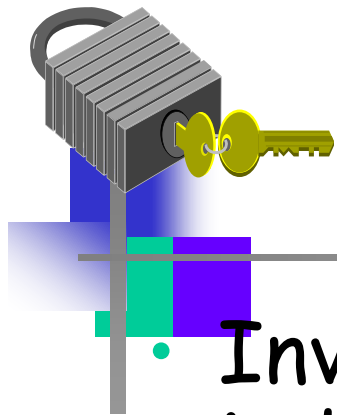


- First introduced by Diffie-Hellman in 1976
- Mathematical functions rather than simple operations on bit patterns
- Allows two separate keys
 - Exchange keys securely
 - Compute discrete logarithms
- Some misconceptions, corrected
 - NOT more secure than symmetric key
 - Does NOT Makes symmetric key obsolete
 - Central agent is needed for both

Diffie-Hellman Key Exchange

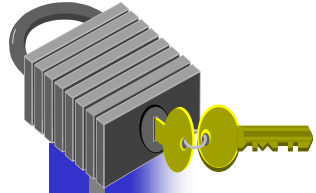


- The purpose of the algorithm is to enable two users to **securely exchange a key** that can then be used for subsequent encryption of messages.
- The algorithm itself is limited to the exchange of secret values.
- A number of commercial products employ this key exchange technique.



Diffie-Hellman

- Invented by Williamson (GCHQ) and, independently, by D and H (Stanford)
- A “key exchange” algorithm
 - Used to establish a shared symmetric key
- **Not** for encrypting or signing
- Based on **discrete log** problem:
 - **Given:** g , p , and $g^k \bmod p$
 - **Find:** exponent k



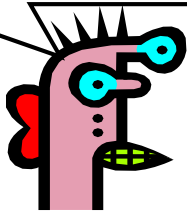
Diffie-Hellman

- Let p be prime, let g be a **generator**
 - For any $x \in \{1, 2, \dots, p-1\}$ there is n s.t. $x = g^n \bmod p$
- Alice selects her private value a
- Bob selects his private value b
- Alice sends $g^a \bmod p$ to Bob
- Bob sends $g^b \bmod p$ to Alice
- Both compute shared secret, $g^{ab} \bmod p$
- Shared secret can be used as symmetric key



Diffie-Hellman basics

Pick secret, random
X

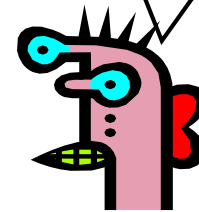


Alice

$g^x \bmod p$

$g^y \bmod p$

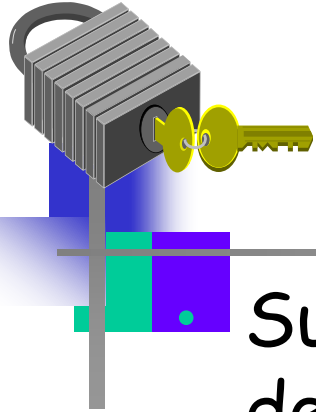
Pick secret, random
Y



Bob

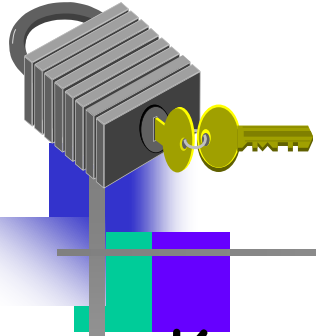
Compute $k = (g^y)^x = g^{xy} \bmod p$

Compute $k = (g^x)^y = g^{xy} \bmod p$



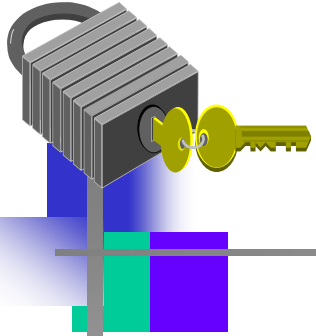
Diffie-Hellman

- Suppose Bob and Alice use Diffie-Hellman to determine symmetric key $K = g^{ab} \bmod p$
- Trudy can see $g^a \bmod p$ and $g^b \bmod p$
 - But... $g^a g^b \bmod p = g^{a+b} \bmod p \neq g^{ab} \bmod p$
- If Trudy can find a or b , she gets key K
- If Trudy can solve **discrete log** problem, she can find a or b



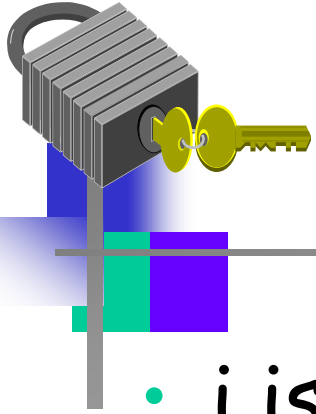
Diffie-Hellman

- Key exchange algorithm using public and private values
- Based on the discrete logarithm problem
- To understand the discrete logarithm problem
 - Define the primitive root of p to be one whose powers generate all the integers from 1 to $(p-1)$ from some prime number p



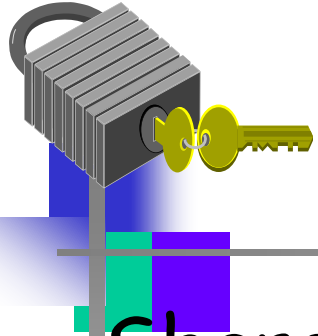
Diffie Hellman details

- If a is a primitive root of prime p ,
- $a \bmod p, a^2 \bmod p, \dots, a^{p-1} \bmod p$ are distinct and contain 1 through $(p-1)$ in some order
- For b less than p and a , find unique exponent i such that
- $b = a^i \bmod p$ where $0 \leq i \leq (p-1)$



DH details continued...

- i is the discrete logarithm
- Denoted $\text{ind}_{a,p}(b)$
- It is hard to calculate it given $a^i \bmod p$



DH Key Exchange Details

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

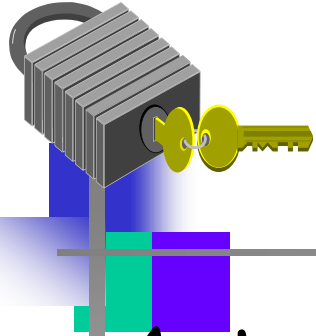
$$K_{AB} = a^{x_A \cdot x_B} \bmod q$$

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

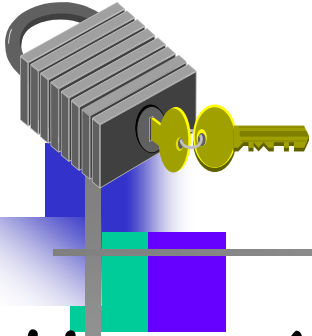
$$= y_B^{x_A} \bmod q \quad (\text{which A can compute})$$

- 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 , must solve discrete log (hard)

Diffie-Hellman Key Exchange



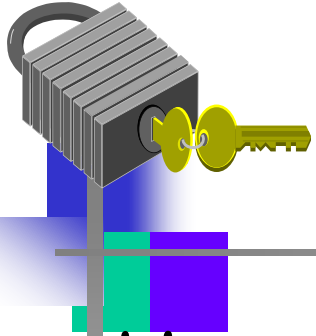
- Actual key exchange for either party consists of raising the others "public key" to power of their private key.
- Resulting number (or as much of as is necessary) is used as the key for a block cipher or other private key scheme.
- Alice and Bob can keep and use same key again.



Diffie Hellman Example

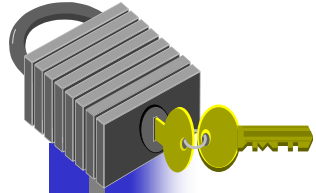
Users Alice & Bob who wish to swap keys:

- Agree on prime $q=353$ and $a=3$
- Select random secret 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**
- Authentication of the keys is needed



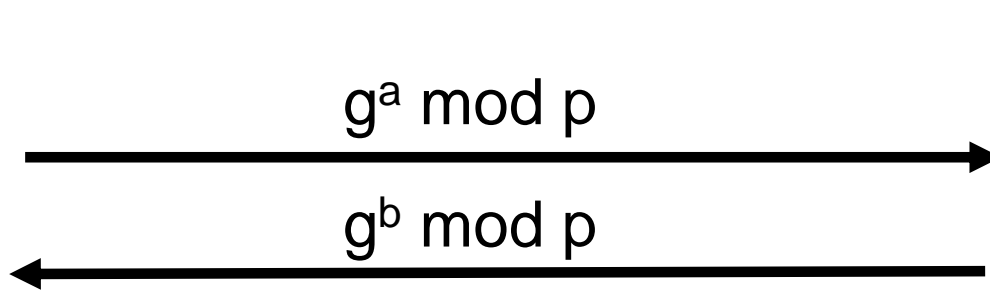
Diffie-Hellman

Public: g and p

- **Private:** Alice's exponent a , Bob's exponent b

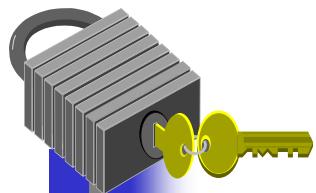


Alice, a



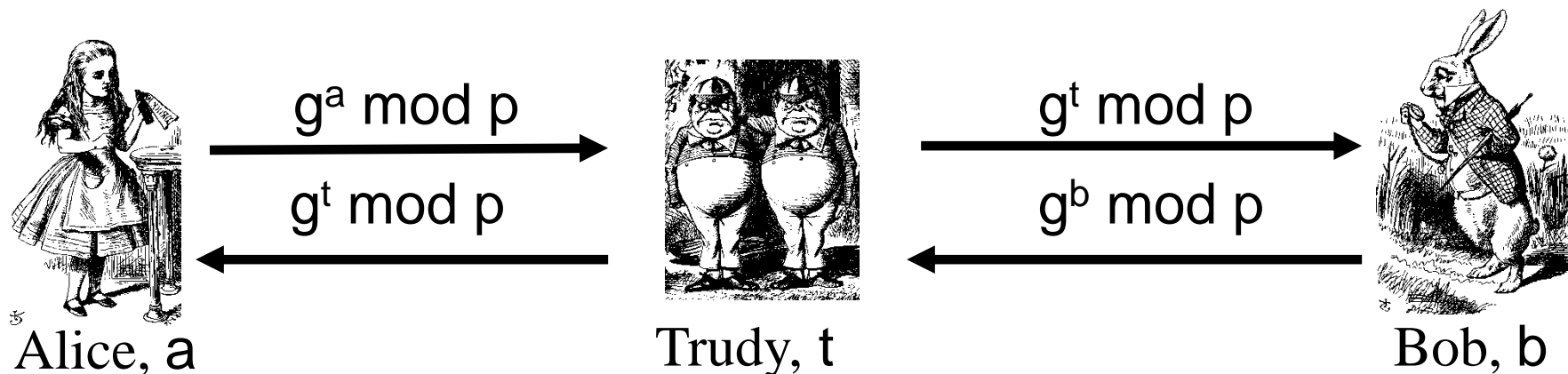
Bob, b

- Alice computes $(g^b)^a = g^{ba} = g^{ab} \bmod p$
- Bob computes $(g^a)^b = g^{ab} \bmod p$
- Use $K = g^{ab} \bmod p$ as symmetric key

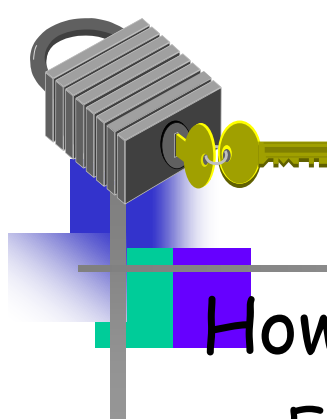


Diffie-Hellman

- Subject to man-in-the-middle (MiM) attack



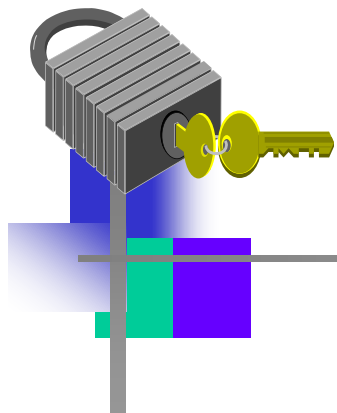
- ❑ Trudy shares secret $g^{at} \bmod p$ with Alice
- ❑ Trudy shares secret $g^{bt} \bmod p$ with Bob
- ❑ Alice and Bob don't know Trudy exists!



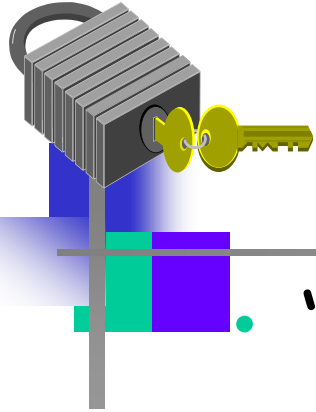
Diffie-Hellman

How to prevent MiM attack?

- Encrypt DH exchange with symmetric key
 - Encrypt DH exchange with public key
 - Sign DH values with private key
 - Other?
- At this point, DH may look pointless...
 - ...but it's not (more on this later)
 - In any case, you **MUST** be aware of MiM attack on Diffie-Hellman

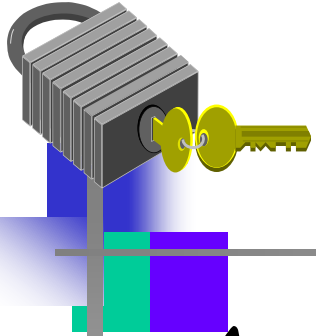


Elliptic Curve Cryptography



Elliptic Curve Crypto (ECC)

- “Elliptic curve” is **not** a cryptosystem
- Elliptic curves are a different way to do the math in public key system
- Elliptic curve versions DH, RSA, etc.
- Elliptic curves may be more efficient
 - Fewer bits needed for same security
 - But the operations are more complex



What is an Elliptic Curve?

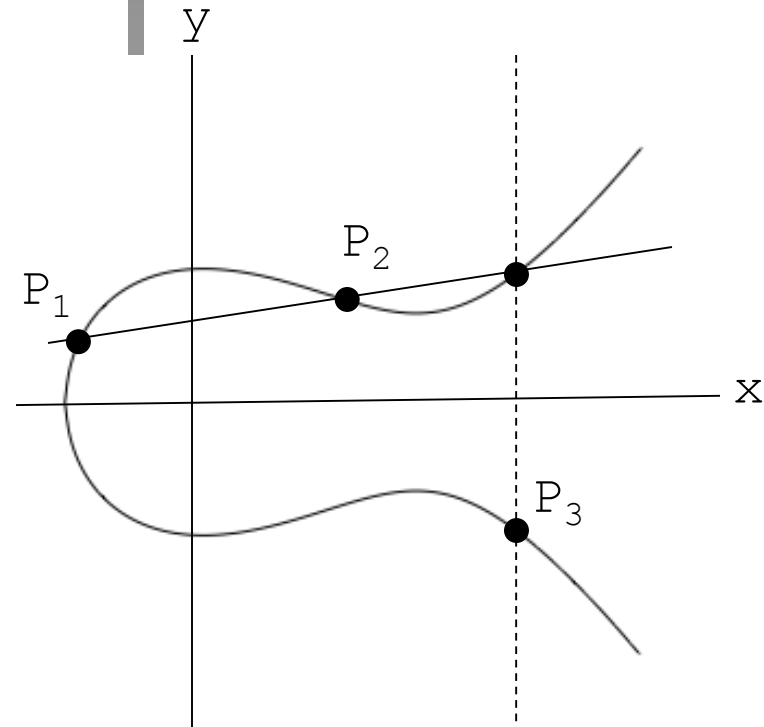
- An elliptic curve E is the graph of an equation of the form

$$y^2 = x^3 + ax + b$$

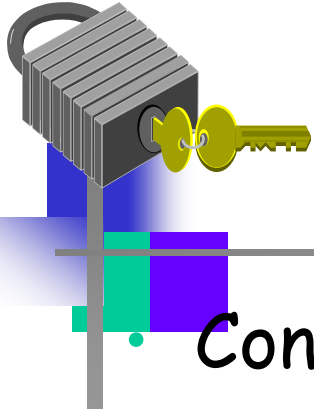
- Also includes a “point at infinity”
- What do elliptic curves look like?
- See the next slide!



Elliptic Curve Picture



- Consider elliptic curve
$$E: y^2 = x^3 - x + 1$$
- If P_1 and P_2 are on E , we can define
$$P_3 = P_1 + P_2$$
as shown in picture
- Addition is all we need



Points on Elliptic Curve

- Consider $y^2 = x^3 + 2x + 3 \pmod{5}$

$$x = 0 \Rightarrow y^2 = 3 \Rightarrow \text{no solution} \pmod{5}$$

$$x = 1 \Rightarrow y^2 = 6 = 1 \Rightarrow y = 1, 4 \pmod{5}$$

$$x = 2 \Rightarrow y^2 = 15 = 0 \Rightarrow y = 0 \pmod{5}$$

$$x = 3 \Rightarrow y^2 = 36 = 1 \Rightarrow y = 1, 4 \pmod{5}$$

$$x = 4 \Rightarrow y^2 = 75 = 0 \Rightarrow y = 0 \pmod{5}$$

- Then points on the elliptic curve are

$(1, 1)$ $(1, 4)$ $(2, 0)$ $(3, 1)$ $(3, 4)$ $(4, 0)$
and the point at infinity: ∞



Elliptic Curve Math

• Addition on: $y^2 = x^3 + ax + b \pmod{p}$

$$P_1 = (x_1, y_1), P_2 = (x_2, y_2)$$

$$P_1 + P_2 = P_3 = (x_3, y_3) \text{ where}$$

$$x_3 = m^2 - x_1 - x_2 \pmod{p}$$

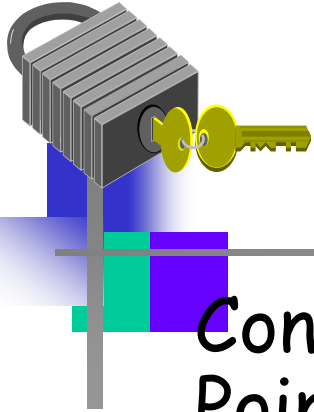
$$y_3 = m(x_1 - x_3) - y_1 \pmod{p}$$

And $m = (y_2 - y_1) * (x_2 - x_1)^{-1} \pmod{p}$, if $P_1 \neq P_2$

$$m = (3x_1^2 + a) * (2y_1)^{-1} \pmod{p}, \text{ if } P_1 = P_2$$

Special cases: If m is infinite, $P_3 = \infty$, and

$$\infty + P = P \text{ for all } P$$



Elliptic Curve Addition

Consider $y^2 = x^3 + 2x + 3 \pmod{5}$.

Points on the curve are $(1, 1)$ $(1, 4)$
 $(2, 0)$ $(3, 1)$ $(3, 4)$ $(4, 0)$ and ∞

- What is $(1, 4) + (3, 1) = P_3 = (x_3, y_3)$?

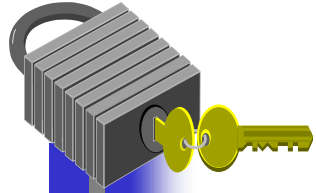
$$m = (1-4) * (3-1)^{-1} = -3 * 2^{-1}$$

$$= 2(3) = 6 = 1 \pmod{5}$$

$$x_3 = 1 - 1 - 3 = 2 \pmod{5}$$

$$y_3 = 1(1-2) - 4 = 0 \pmod{5}$$

- On this curve, $(1, 4) + (3, 1) = (2, 0)$

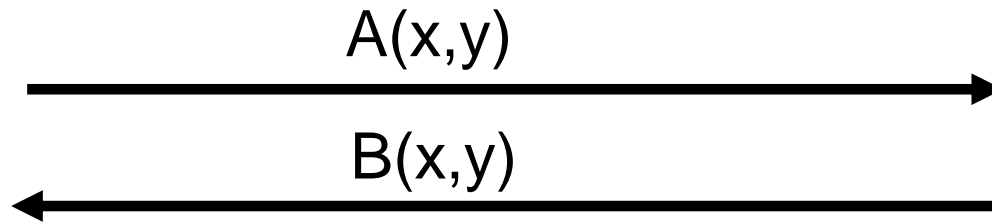


ECC Diffie-Hellman

- **Public:** Elliptic curve and point (x,y) on curve
- **Private:** Alice's A and Bob's B



Alice, A



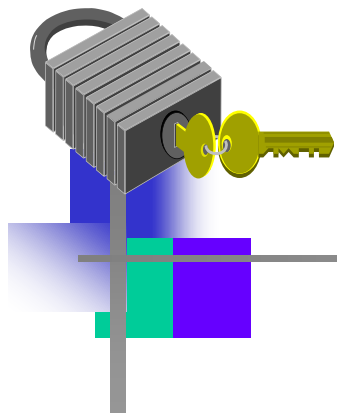
Bob, B

- Alice computes $A(B(x,y))$
- Bob computes $B(A(x,y))$
- These are the same since $AB = BA$

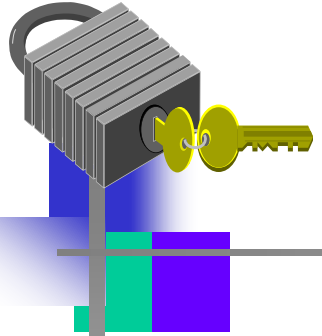


ECC Diffie-Hellman

- **Public:** Curve $y^2 = x^3 + 7x + b \pmod{37}$
and point $(2, 5) \Rightarrow b = 3$
- **Alice's private:** $A = 4$
- **Bob's private:** $B = 7$
- Alice sends Bob: $4(2, 5) = (7, 32)$
- Bob sends Alice: $7(2, 5) = (18, 35)$
- Alice computes: $4(18, 35) = (22, 1)$
- Bob computes: $7(7, 32) = (22, 1)$



Uses for Public Key Crypto



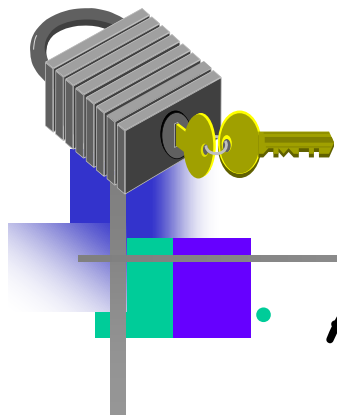
Uses for Public Key Crypto

- Confidentiality
 - Transmitting data over insecure channel
 - Secure storage on insecure media
- Authentication (later)
- Digital signature provides integrity and **non-repudiation**
 - No non-repudiation with symmetric keys



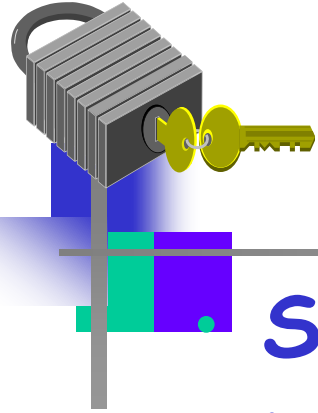
Non-non-repudiation

- Alice orders 100 shares of stock from Bob
- Alice computes **MAC** using symmetric key
- Stock drops, Alice claims she did **not** order
- Can Bob prove that Alice placed the order?
- **No!** Since Bob also knows the symmetric key, he could have forged message
- **Problem:** Bob knows Alice placed the order, but he can't prove it



Non-repudiation

- Alice orders 100 shares of stock from Bob
- Alice **signs** order with her private key
- Stock drops, Alice claims she did not order
- Can Bob prove that Alice placed the order?
- **Yes!** Only someone with Alice's private key could have signed the order
- This assumes Alice's private key is not stolen (revocation problem)

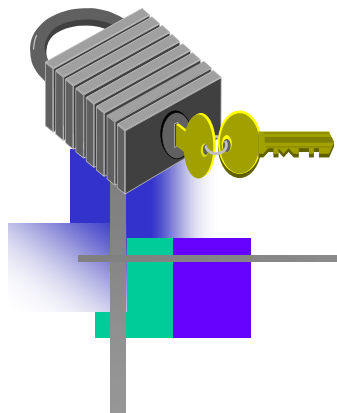


Public Key Notation

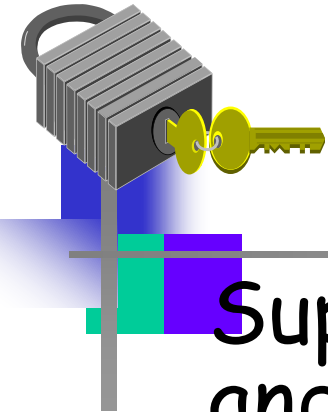
- **Sign** message M with Alice's **private key**: $[M]_{\text{Alice}}$
- **Encrypt** message M with Alice's **public key**: $\{M\}_{\text{Alice}}$
- Then

$$\{[M]_{\text{Alice}}\}_{\text{Alice}} = M$$

$$[\{M\}_{\text{Alice}}]_{\text{Alice}} = M$$



Sign and Encrypt vs Encrypt and Sign



Confidentiality and Non-repudiation?

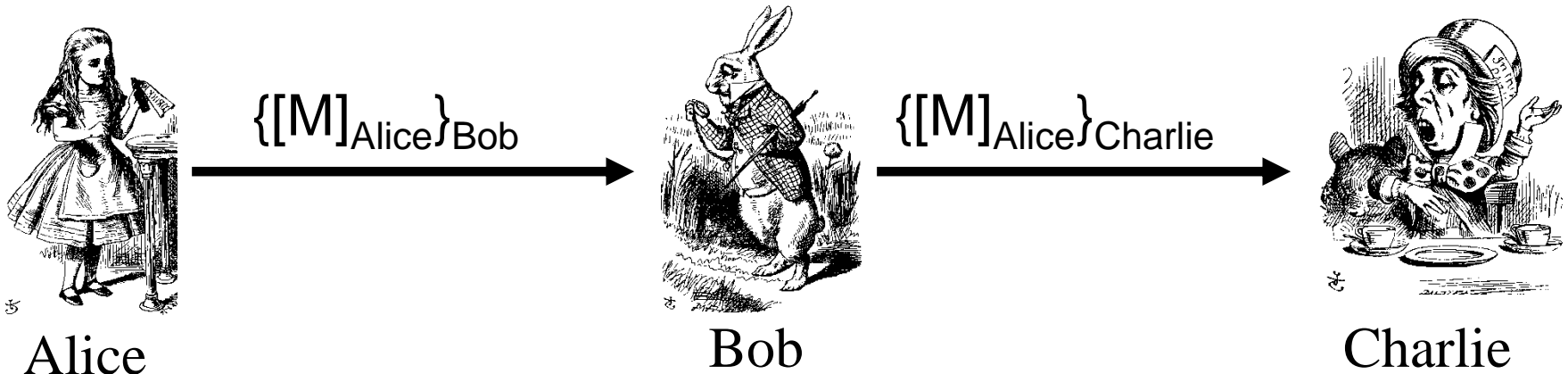
Suppose that we want confidentiality and integrity/non-repudiation

- Can public key crypto achieve both?
- Alice sends message to Bob
 - **Sign and encrypt** $\{[M]_{\text{Alice}}\}_{\text{Bob}}$
 - **Encrypt and sign** $[\{M\}_{\text{Bob}}]_{\text{Alice}}$
- Can the order possibly matter?



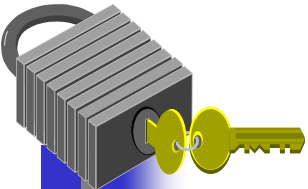
Sign and Encrypt

□ $M = \text{"I love you"}$



□ **Q:** What's the problem?

□ **A:** No problem — public key is public



Encrypt and Sign

□ $M = \text{“My theory, which is mine....”}$



Alice

$[\{M\}_{\text{Bob}}]_{\text{Alice}}$



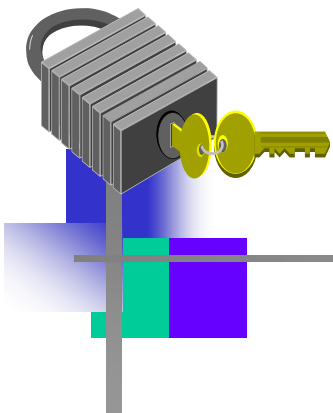
Charlie

$[\{M\}_{\text{Bob}}]_{\text{Charlie}}$

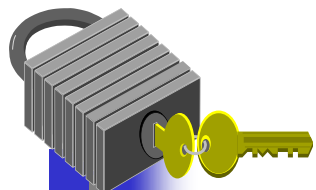


Bob

- **Note** that Charlie cannot decrypt M
- **Q:** What is the problem?
- **A:** No problem — public key is public



Public Key Infrastructure



Public Key Certificate

Certificate contains name of user and user's public key (and possibly other info)

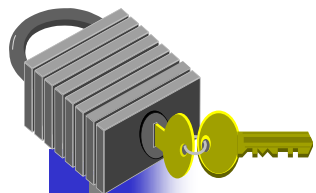
- It is *signed* by the issuer, a **Certificate Authority (CA)**, such as VeriSign

$$M = (\text{Alice}, \text{Alice's public key}), S = [M]_{CA}$$

$$\text{Alice's Certificate} = (M, S)$$

- Signature on certificate is verified using CA's public key:

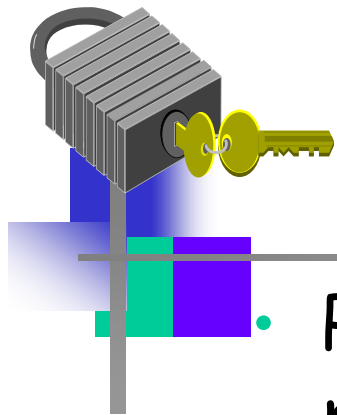
$$\text{Verify that } M = \{S\}_{CA}$$



Certificate Authority

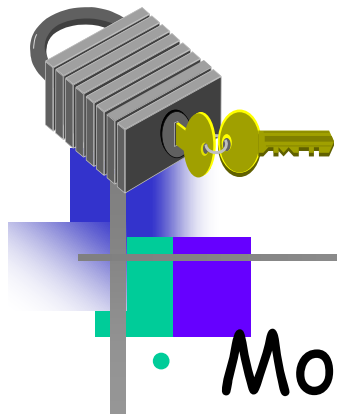
Certificate authority (CA) is a trusted 3rd party (TTP) — creates and signs certificates

- Verify signature to verify integrity & identity of **owner of corresponding private key**
 - Does **not** verify the identity of the **sender** of certificate — certificates are public keys!
- Big problem if CA makes a mistake (a CA once issued Microsoft certificate to someone else)
- A common format for certificates is X.509



PKI

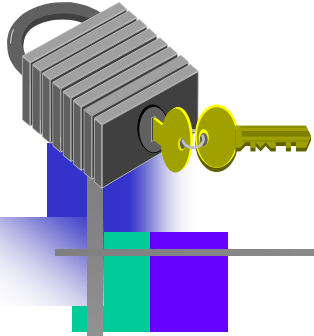
- Public Key Infrastructure (PKI): the stuff needed to securely use public key crypto
 - Key generation and management
 - Certificate authority (CA) or authorities
 - Certificate revocation lists (CRLs), etc.
- No general standard for PKI
- We mention 3 generic "trust models"



PKI Trust Models

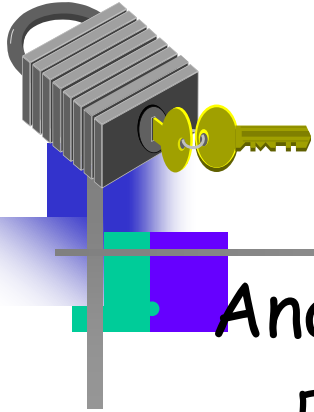
- Monopoly model

- One universally trusted organization is the *CA* for the known universe
- Big problems if *CA* is ever compromised
- Who will act as *CA*???
- System is useless if you don't trust the *CA*!



PKI Trust Models

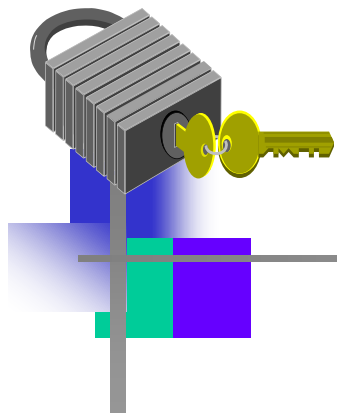
- Oligarchy
 - Multiple trusted CAs
 - This is approach used in browsers today
 - Browser may have 80 or more certificates, just to verify certificates!
 - User can decide which CAs to trust



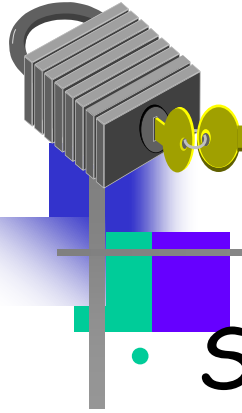
PKI Trust Models

Anarchy model

- Everyone is a CA...
- Users must decide who to trust
- This approach used in PGP: "Web of trust"
- Why is it anarchy?
 - Suppose a certificate is signed by Frank and you don't know Frank, but you do trust Bob and Bob says Alice is trustworthy and Alice vouches for Frank. Should you accept the certificate?
- **Many** other trust models and PKI issues

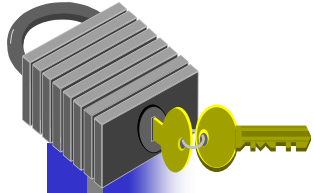


Confidentiality in the Real World



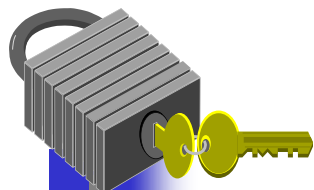
Symmetric Key vs Public Key

- Symmetric key +'s
 - **Speed**
 - No public key infrastructure (PKI) needed
- Public Key +'s
 - **Signatures** (non-repudiation)
 - No *shared* secret (but, private keys...)



Notation Reminder

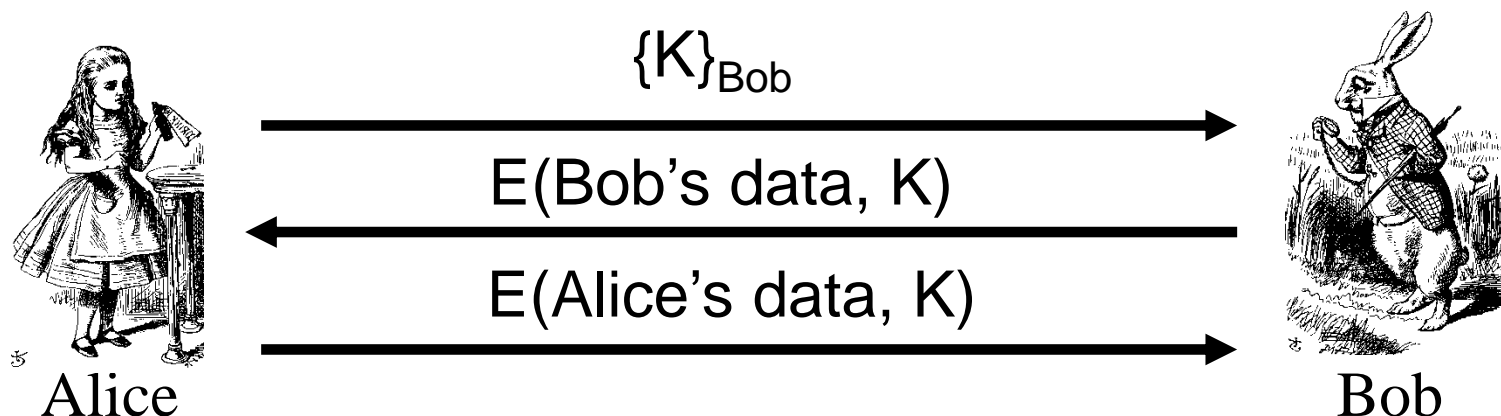
- Public key notation
 - Sign M with Alice's **private key**
 $[M]_{\text{Alice}}$
 - Encrypt M with Alice's **public key**
 $\{M\}_{\text{Alice}}$
- Symmetric key notation
 - Encrypt P with symmetric key K
 $C = E(P, K)$
 - Decrypt C with symmetric key K
 $P = D(C, K)$



Real World Confidentiality

Hybrid cryptosystem

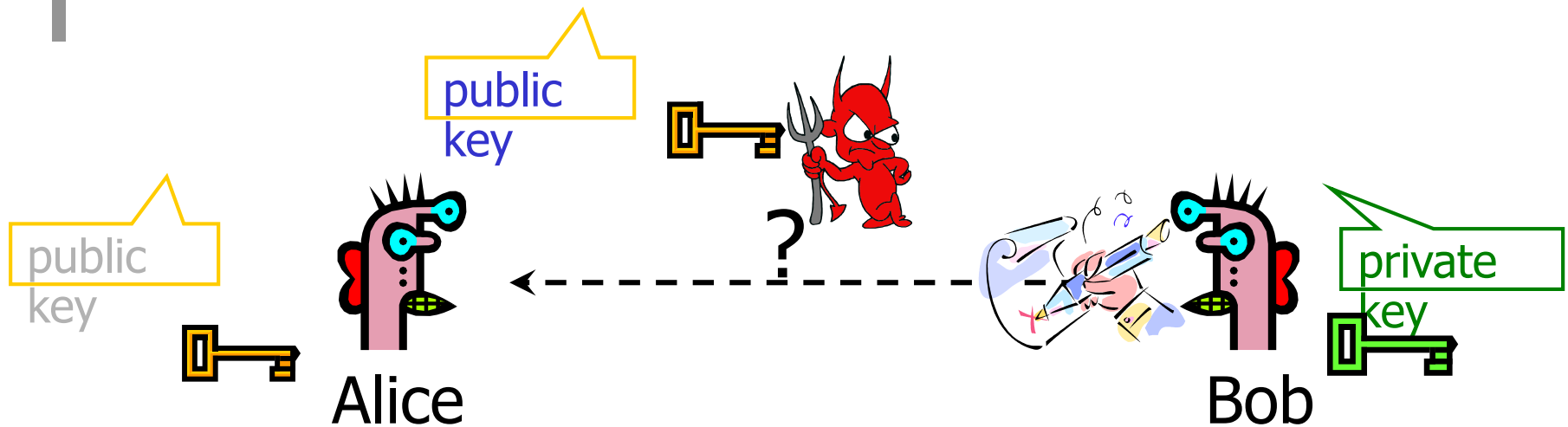
- Public key crypto to establish a key
- Symmetric key crypto to encrypt data...



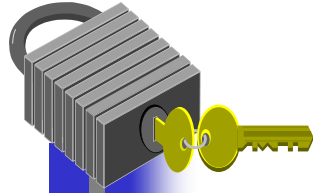
□ Can Bob be sure he's talking to Alice?



Digital Signatures: The basic idea

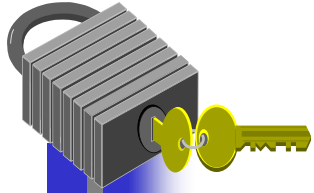


Vulnerable to man in the middle attack



Digital Signatures

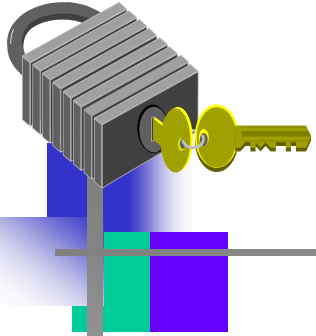
- Diffie-Hellman vulnerability can be overcome with the use of digital signatures and public-key certificates.
- Digital signatures provide the ability to:
 - verify author, date & time of signature
 - authenticate message contents
 - be verified by third parties to resolve disputes
- Digital signatures include authentication function with additional capabilities



Key Exchange Protocols

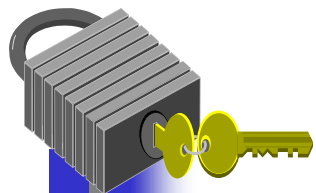
Consider a simple protocol that makes use of the Diffie-Hellman calculation:

- Suppose that user *A* wishes to set up a connection with user *B* and use a secret key to encrypt messages on that connection.
- User *A* can generate a one-time private key X_A , calculate Y_A , and send that to user *B*.
- User *B* responds by generating a private value X_B , calculating Y_B , and sending Y_B to user *A*. Both users can now calculate the key.



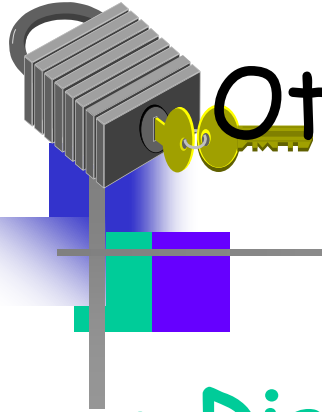
Key Exchange Protocols

- These public values together with global values for q and a are stored in a central directory
- Both users can now calculate the key
- If the central directory is trusted, this provides confidentiality and some authentication
- It does not protect against replay attacks



Man-in-the-Middle Attack

1. Darth prepares by creating two private / public keys
 2. Alice transmits her public key to Bob
 3. Darth intercepts this and transmits his first public key to Bob. Darth also calculates a shared key with Alice
 4. Bob receives the public key and calculates the shared key (with Darth instead of Alice)
 5. Bob transmits his public key to Alice
 6. Darth intercepts this and transmits his second public key to Alice. Darth calculates a shared key with Bob
 7. 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



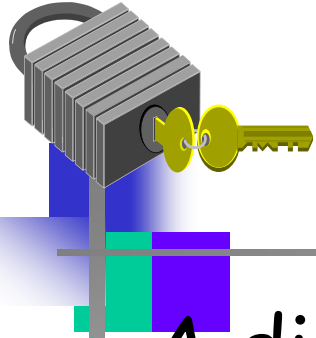
Other Public-Key Cryptographic Algorithms

- **Digital Signature Standard (DSS)**
 - Makes use of the **SHA-1**
 - Not for encryption or key exchange
- **Elliptic-Curve Cryptography (ECC)**
 - Good for smaller bit size
 - Low confidence level, compared with RSA
 - Very complex



Digital Signatures

- As E-commerce grows, so does the need for a high degree of authentication
- "Digital signature is a construct that authenticates both the origin and contents of a message in a manner that is provable to a disinterested third party."
(Bishop)



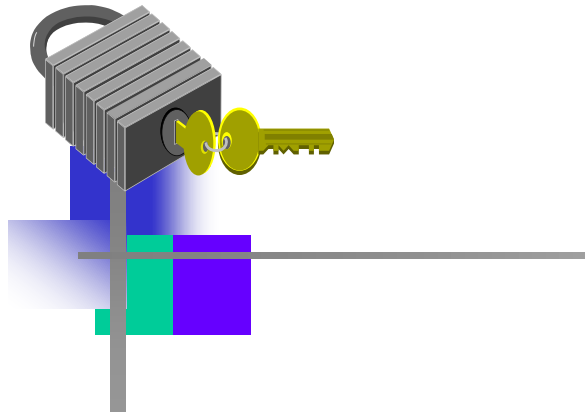
Digital signatures

- A digital signature is an encryption of a document with the creator's private key
- It is attached to a document that validates the creator of the document
- Any one can validate it by decrypting the signature with the claimed creator's public key

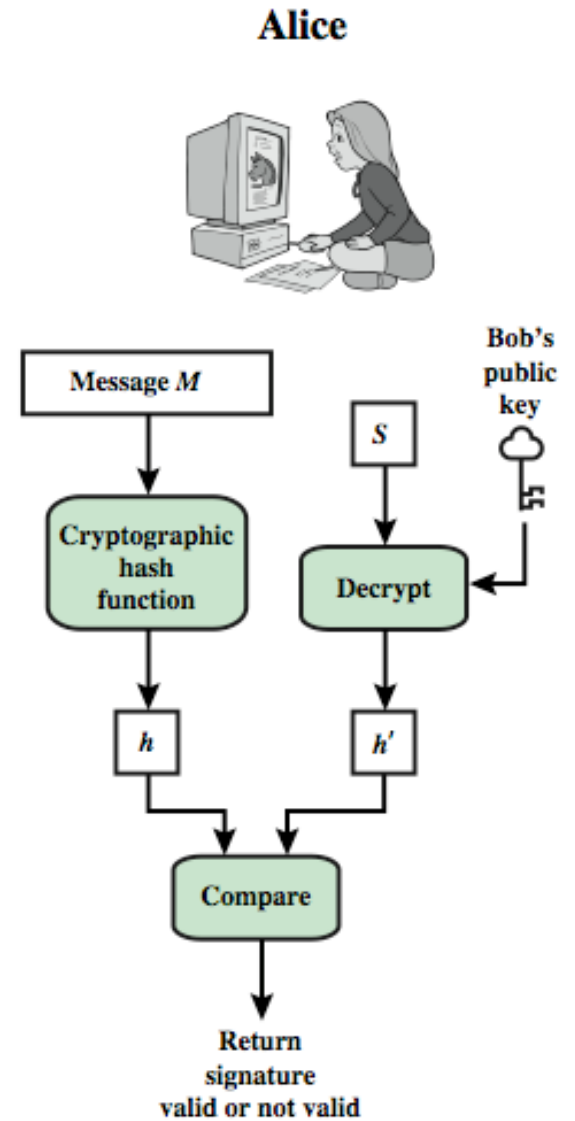
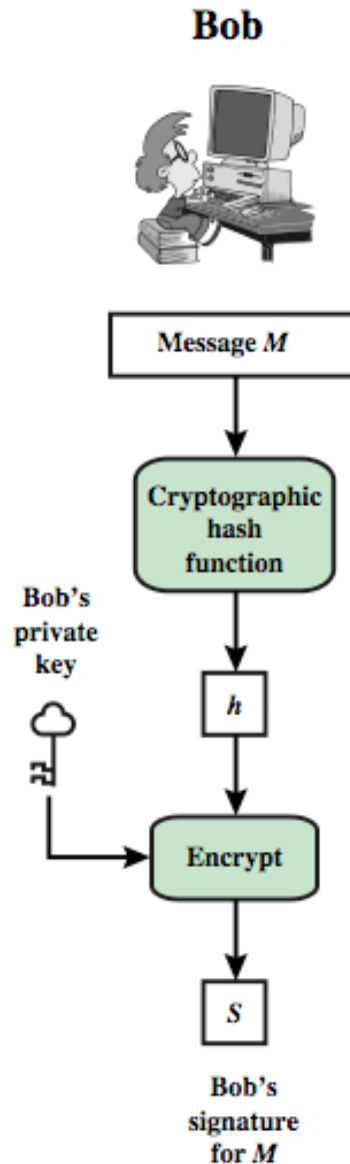


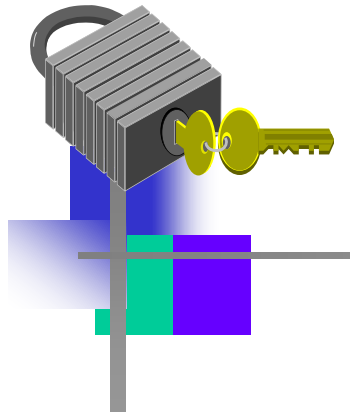
Digital Signatures: The Basic Idea

- Bob can sign a message using a digital signature generation algorithm.
- The inputs to the algorithm are the message and Bob's private key.
- Any other user, say Alice, can verify the signature using a verification algorithm, whose inputs are the message, the signature, and Bob's public key.



Components of the Digital Signature Model





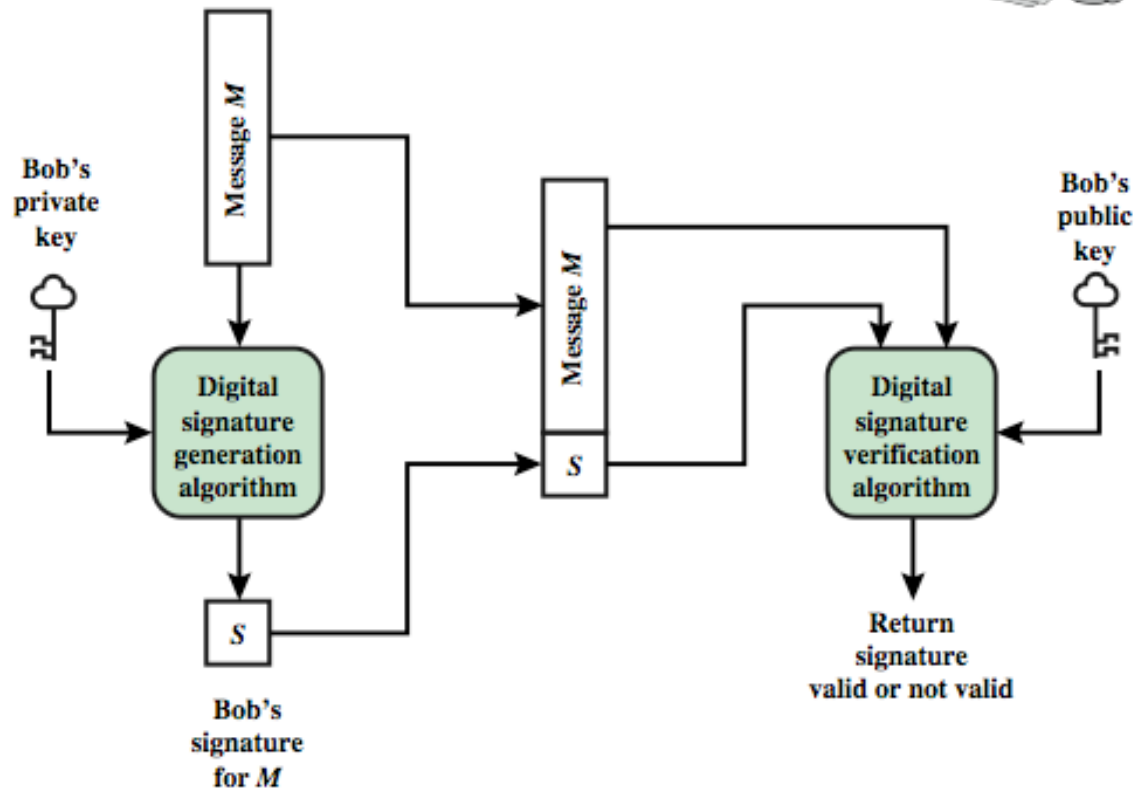
Digital Signature Model

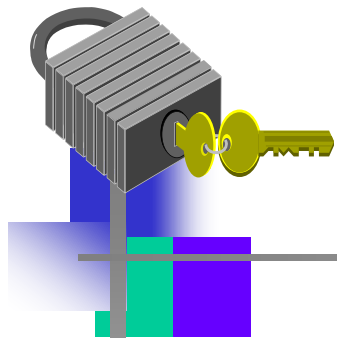
Bob



Transmit

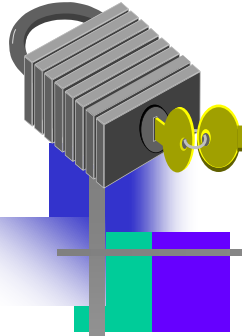
Alice





Digital Signature

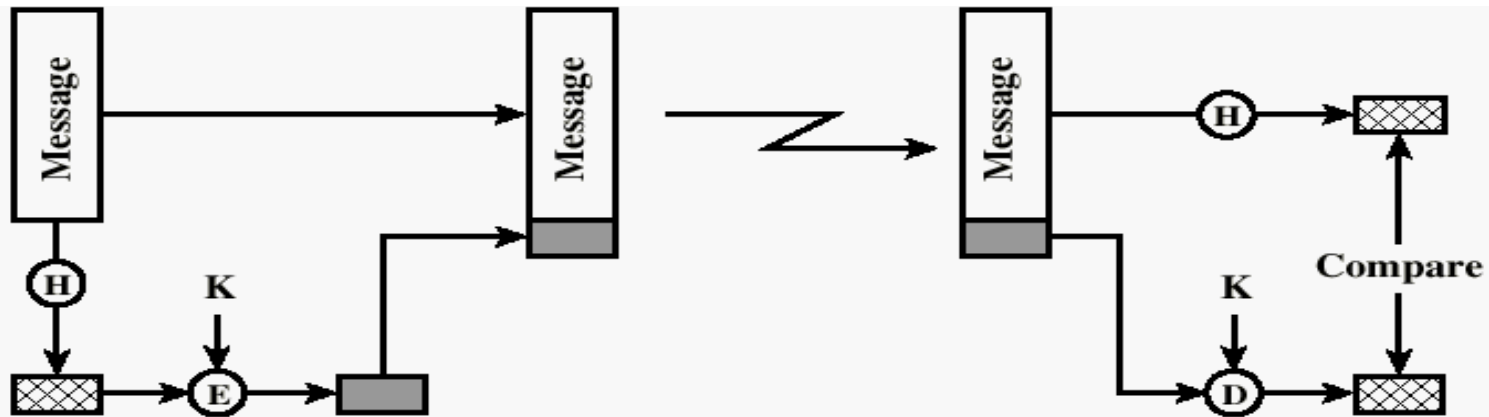
- Message Digest: The representation of text in the form of a single string of digits, created using a formula called a one-way hash function.
- Encrypting a message digest with a private key creates a digital signature, which is an electronic means of authentication.



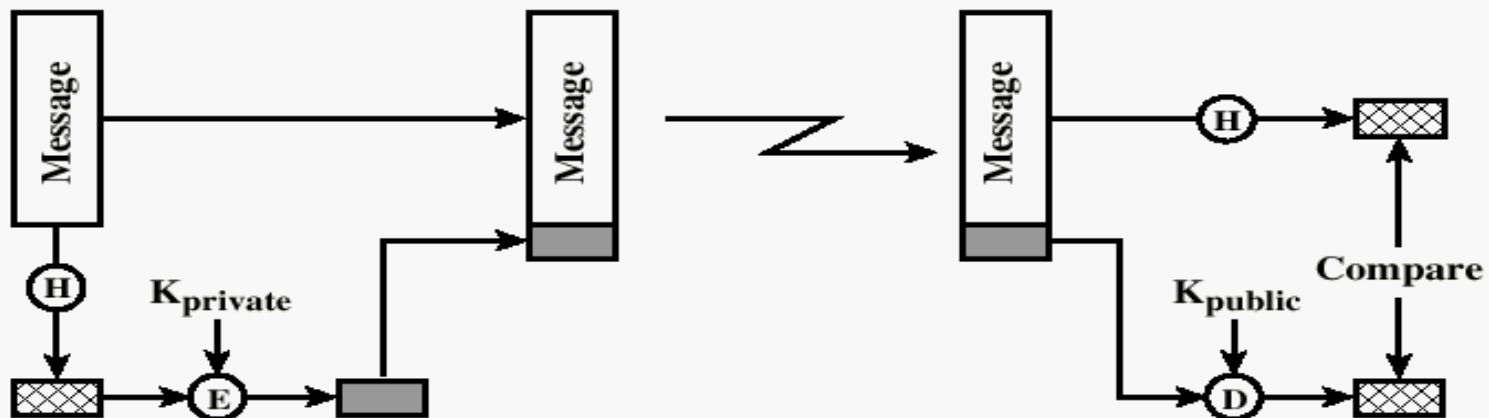
Digital signatures on hashes

- A more efficient way for a digital signature is by creating an authenticator of the document first (a hash)
- Then sign the hash (i.e. encrypt the hash using private key)
- If M is the message (or document), $\text{Hash}(M) = H$
- $\text{sig}_{\text{PV}(A)}(H)$ represents signing H
- i.e. encrypting H with A 's private key

Typical Digital Signature Approach



(a) Using conventional encryption

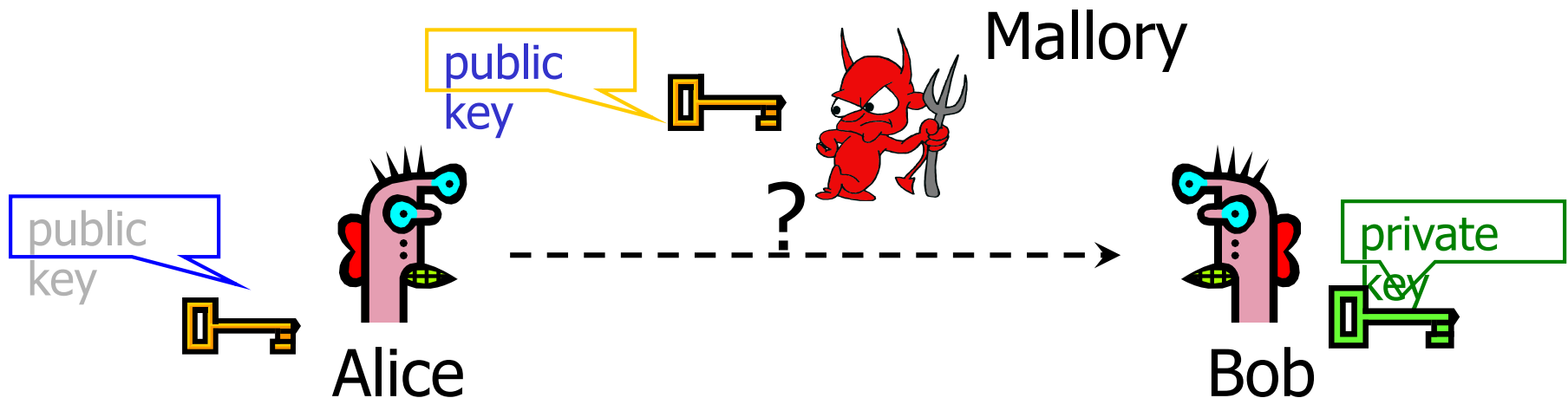


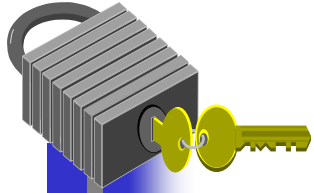
(b) Using public-key encryption



Key management

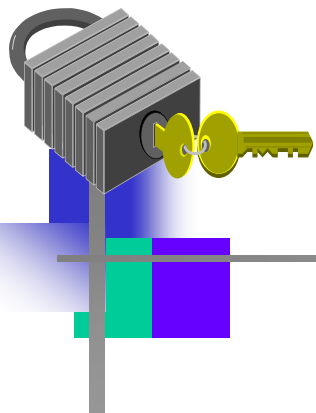
- Distribution of public keys - major problem
 - What's the issue?
 - Can't we just trust Mallory if she claims a key as her public key?



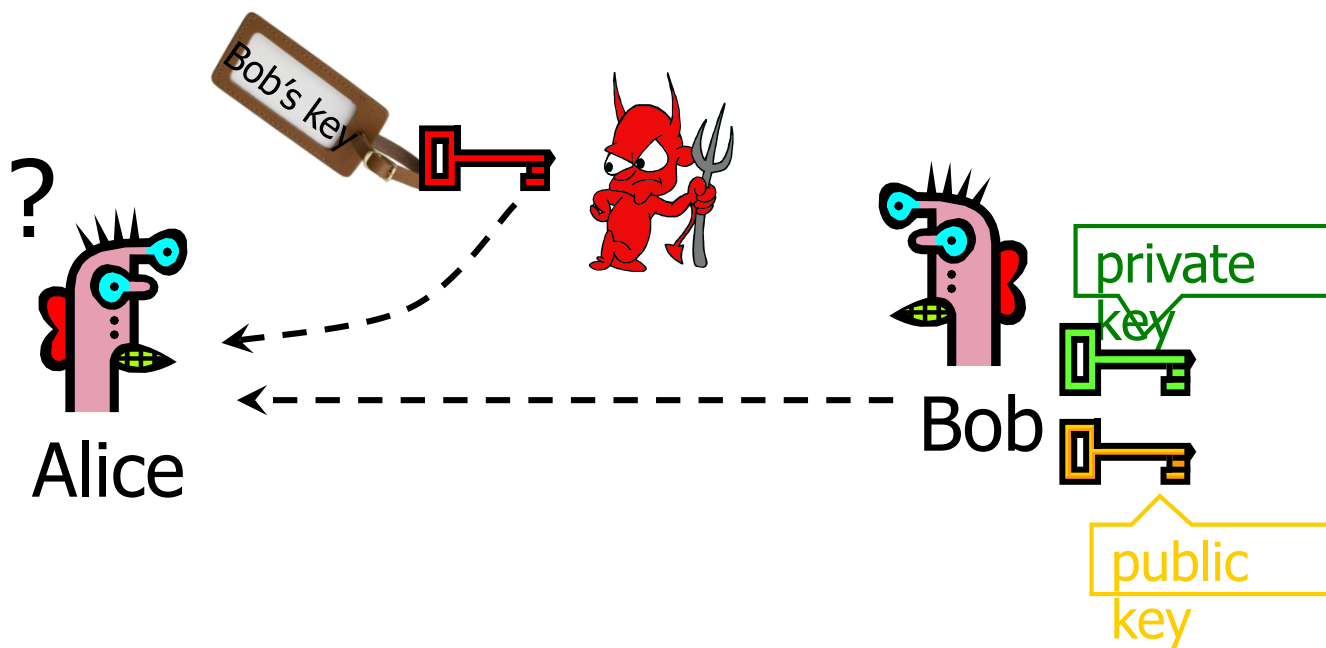


Public keys to exchange secret keys

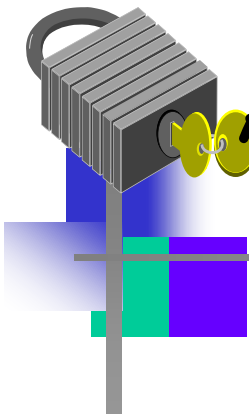
- Using public-keys to exchange secret keys
 - why exchange secret keys?
 - aren't public keys sufficient?



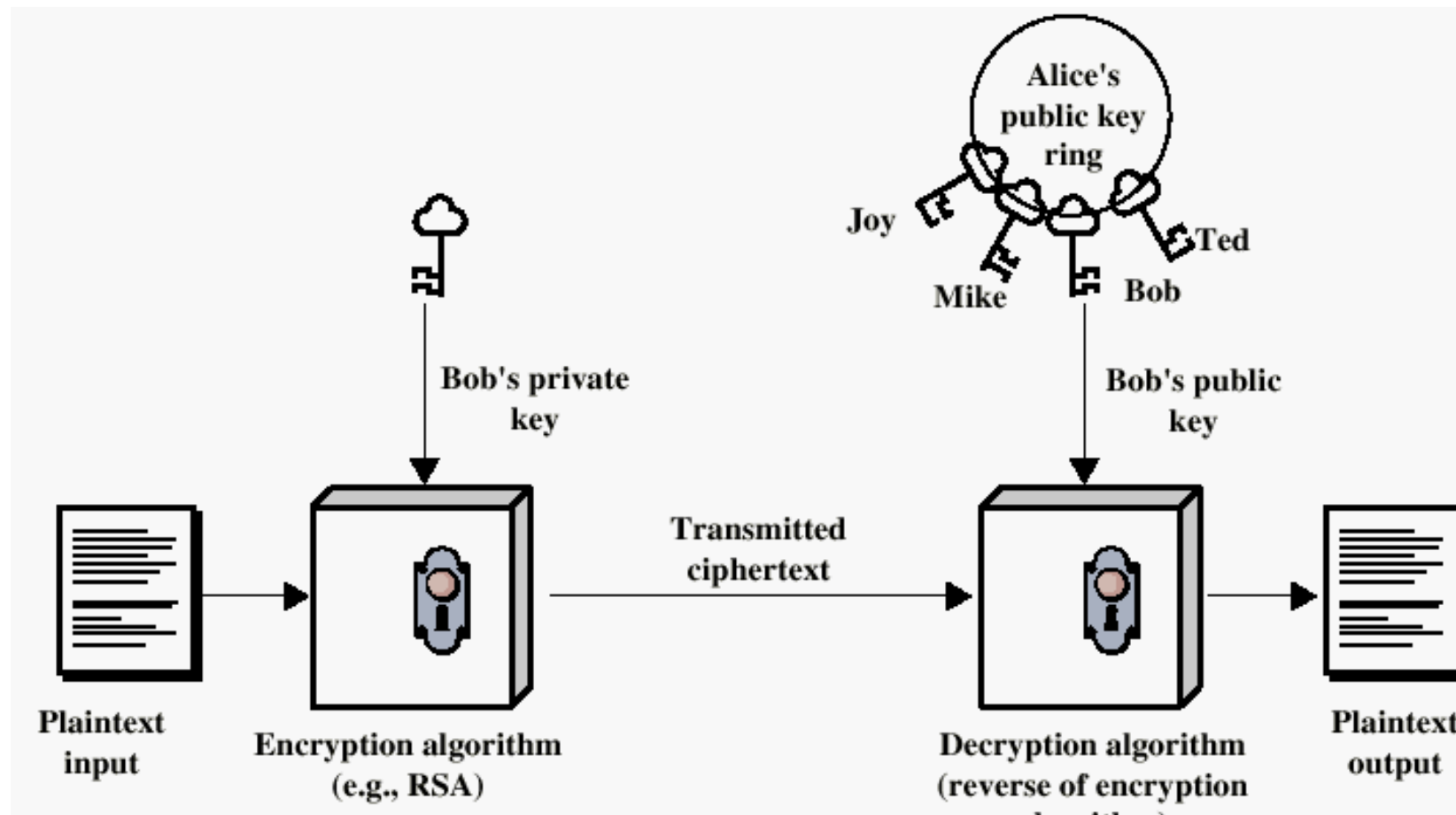
Authenticity of public keys

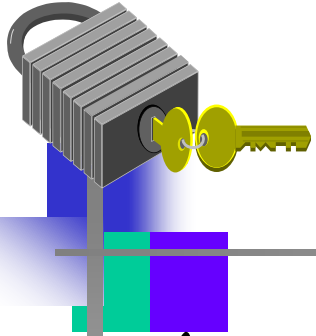


Problem: How does Alice know that the public key she received is really Bob's public key?



Authentication using Public-Key System



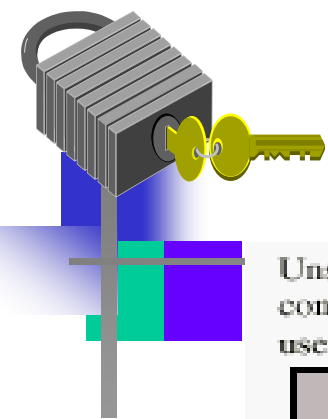


Public-key certificates

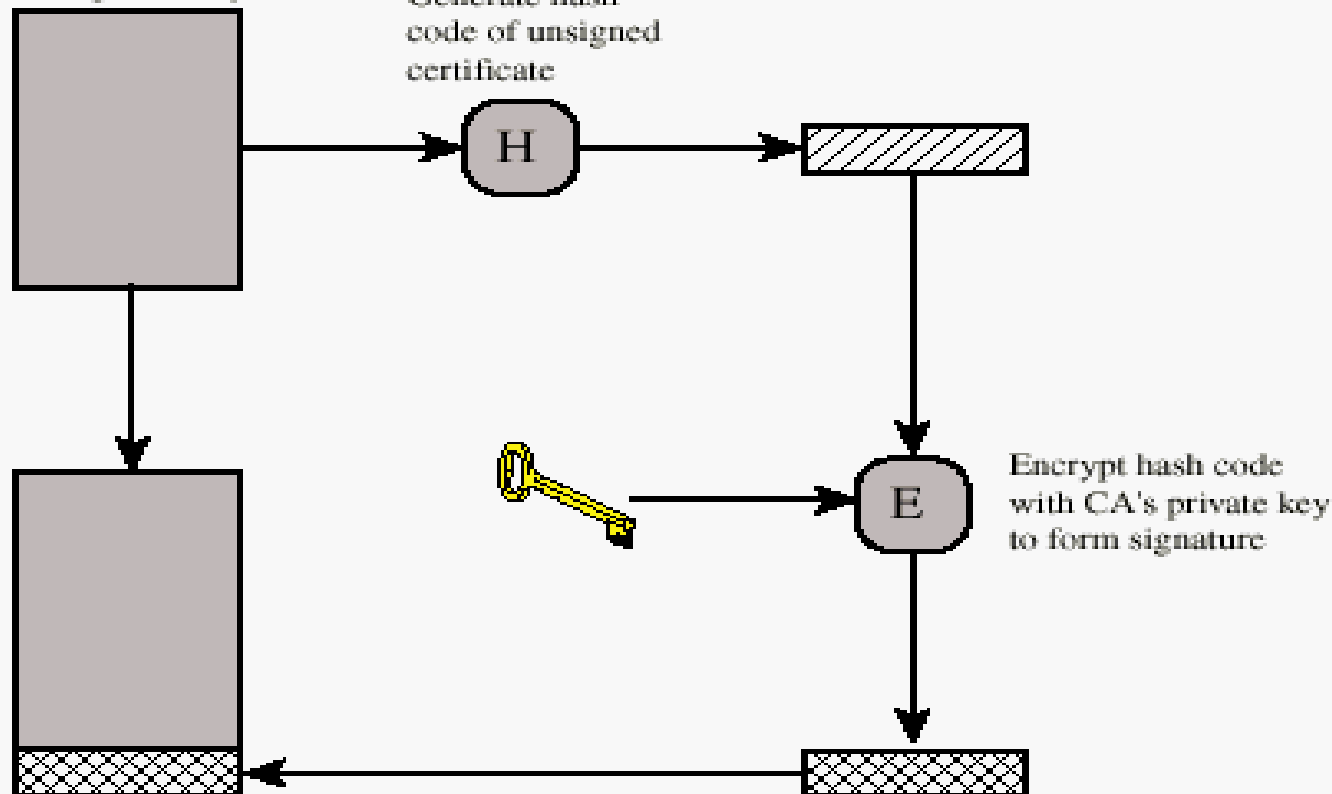
- Anyone can forge public-keys
- Therefore, use public-key certificates
- A **public-key certificate** is a public-key that was signed by a trusted third party (called a certificate authority or *CA*)
- See figure on next slide

Key Management

Public-Key Certificate Use



Unsigned certificate:
contains user ID,
user's public key



Signed certificate:
Recipient can verify
signature using CA's
public key.