ECE 4309

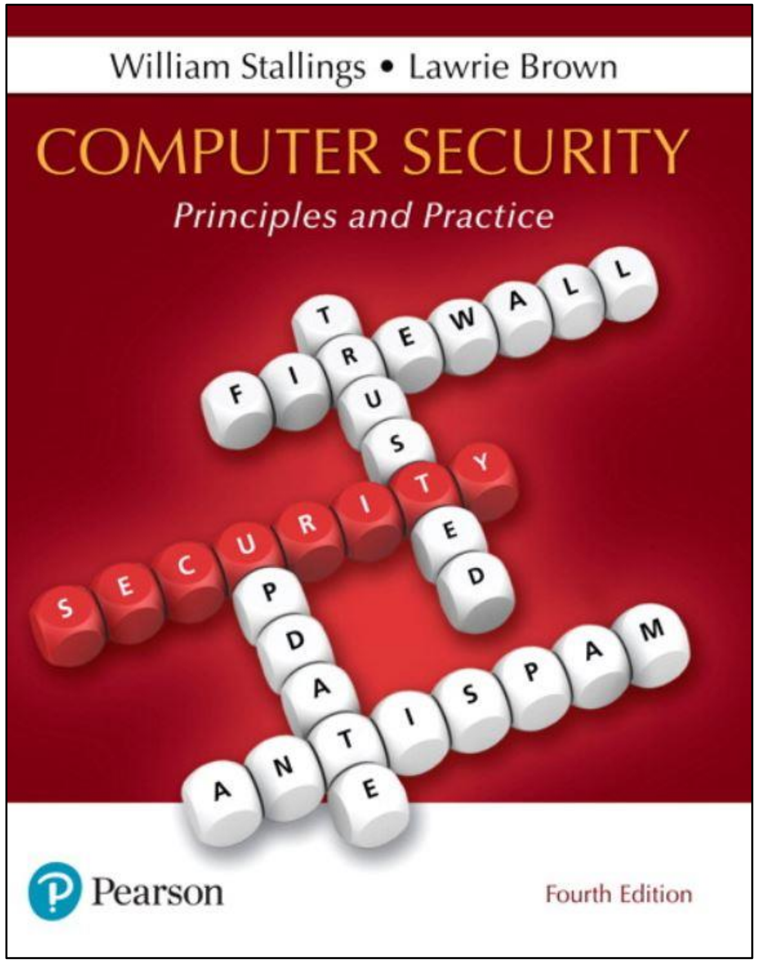
Basics of cryptography – part 3 - Public crytpography

Dr. Valerio Formicola

CalPolyPomona

Computer Security: Principles and Practice

Fourth Edition



Chapter 2 and 21

Cryptographic Tools

Public key encryption

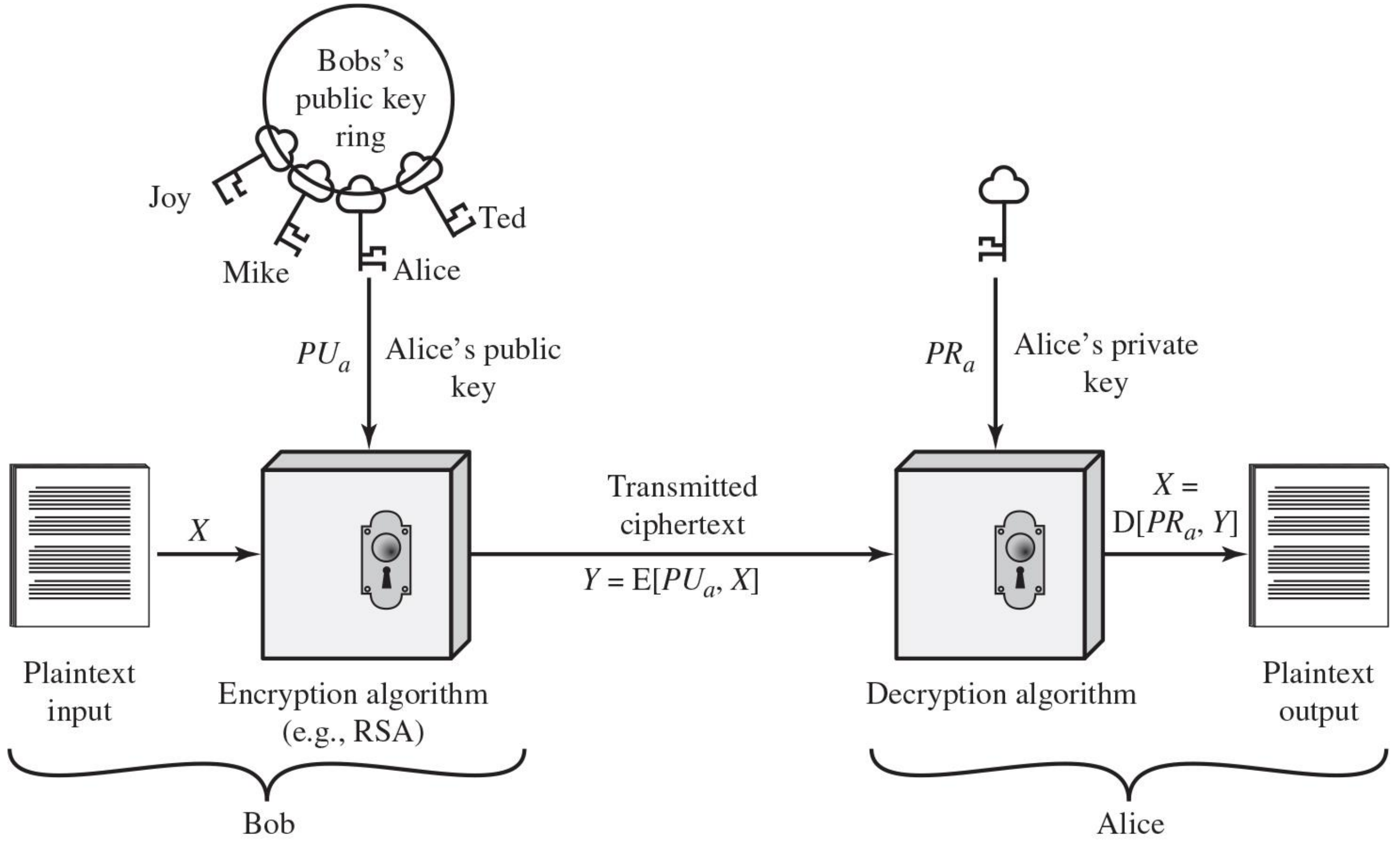
Public-Key Encryption Structure

- Publicly proposed by Diffie and Hellman in 1976:
  - They thought the benefit of a system like this before inventing it
- Based on mathematical functions
  - rather than algorithms and functions, like for shared key
- Asymmetric
  - Uses two separate keys
  - Public key and private key
  - Public key is made public for others to use
- Some form of protocol is needed for distribution
- Two schemas of operation:
  - **Confidentiality**: Ciphertext = Encrypt[PuplicKey, Message] -> Message = Decrypt[PrivateKey, Message]
  - **Authentication and Integrity**: Ciphertext = Encrypt[PrivateKey, Message] -> Message = Decrypt[PublicKey, Message]

Ingredients for public key cryptography

- Plaintext
    - Readable message or data that is fed into the algorithm as input
- Encryption algorithm
    - Performs transformations on the plaintext
- Public and private key
    - Pair of keys, one for encryption, one for decryption
- Ciphertext
    - Scrambled message produced as output
- Decryption key
    - Produces the original plaintext

Figure 2.6 Public-Key Cryptography: schema 1



(a) Encryption with public key

Schema 1: confidentiality between sender and receiver

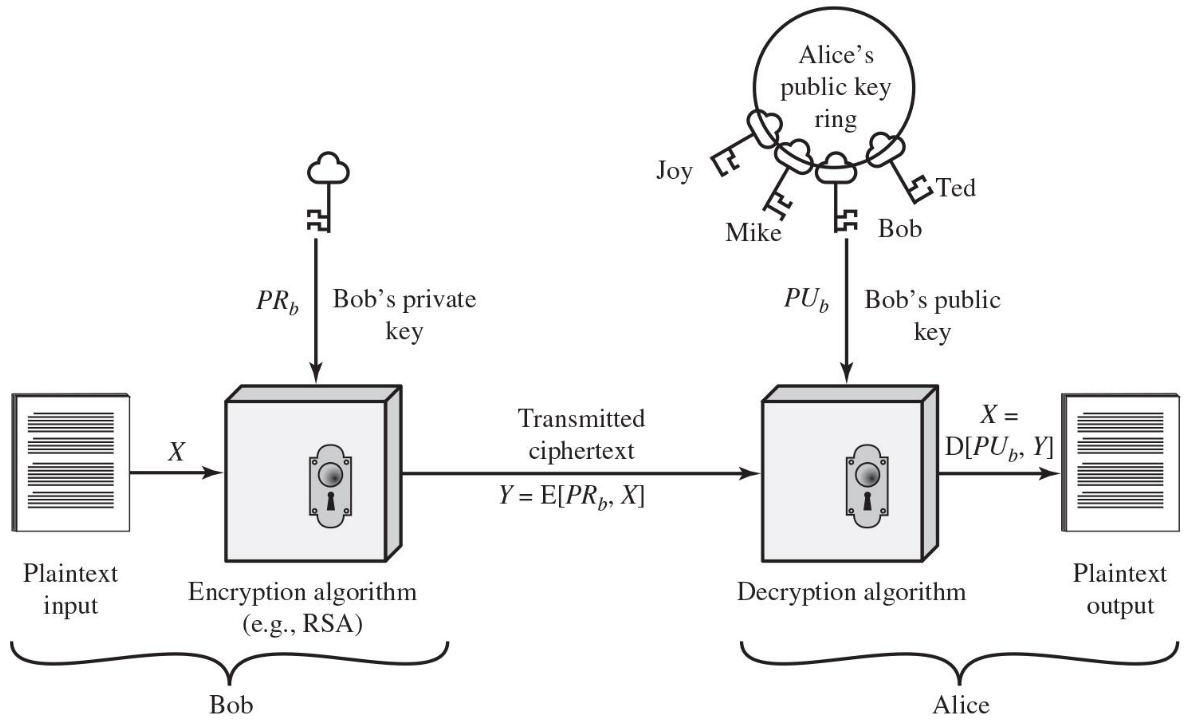
General step (for any use case): All public keys of users are available and accessible to anybody in a public **trusted** location

- Each user keeps his/her own private key locally stored and doesn't need to show to the others (*and should not show to the others*)
- At any time, a user can change the private key and publish the companion public key to replace the old public key in the public location.

- **Schema 1:** The sender (Bob) retrieves many public keys of other entities:
  - PU_{alice}, PU_{joe}, PU_{mrx}, PU_{mrsy}
- If Bob wishes to send a private message to Alice, Bob encrypts the message using Alice's public key PU_{alice}.
- When Alice receives the message, she decrypts it using her private key PR_{alice}. No other recipient can decrypt the message because only Alice knows Alice's private key PR_{alice}.
- Why confidentiality? Because Bob's message is encrypted with Alice's public and only Alice can look at the content, because she is the only one owning the private key of Alice.

Figure 2.6 Public-Key Cryptography (2 of 2)

- User encrypts data using his or her own private key
- Anyone who knows the corresponding public key will be able to decrypt the message



(b) Encryption with private key

Schema 2: authentication and integrity between sender and receivers

General step are the same than use case 1, i.e., all public keys are publicly available and each user owns its private key as a secret

- **Schema 2:** The sender (Bob) retrieves many public keys of other entities:
  - PU_{alice}, PU_{joe}, PU_{mrx}, PU_{mrsy}
- If Bob encrypts data using his or her own private key, anyone who knows the corresponding public key will then be able to decrypt the message.
- Why **authentication**? If a user is able to successfully recover the plaintext from Bob's ciphertext using Bob's public key (and it's easy to be done), this indicates that only Bob could have encrypted the plaintext, thus providing authentication.
- Why **integrity**? No one but Bob would be able to modify the plaintext because only Bob could encrypt the plaintext with Bob's private key

Asymmetric Encryption Algorithms (1 of 2)

- RSA (Rivest, Shamir, Adleman)
  - Developed in 1977
  - Most widely accepted and implemented approach to public-key encryption
  - Block cipher in which the plaintext and ciphertext are integers between 0 and $n-1$ for some n.
- Diffie-Hellman key exchange algorithm
  - Enables two users to securely reach agreement about a shared secret that can be used as a secret key for subsequent symmetric encryption of messages
  - Limited to the exchange of the keys

Table 2.3 Applications for Public-Key Cryptosystems

Depending on the application, the sender uses either the sender’s private key or the receiver’s public key, or both, to perform some type of cryptographic function.
In broad terms, we can classify the use of public-key cryptosystems into three categories: **digital signature, symmetric key distribution, and encryption of secret keys.**

Algorithm	Digital Signature	Symmetric Key Distribution	Encryption of Secret Keys
RSA	Yes	Yes	Yes
Diffie–Hellman	No	Yes	No
DSS	Yes	No	No
Elliptic Curve	Yes	Yes	Yes

Diffie-Hellman algorithm for secret key generation and exchange

Diffie-Hellman Key Exchange

- First published public-key algorithm
- By Diffie and Hellman in 1976 along with the exposition of public key concepts
- Used in a number of commercial products
- Practical method to exchange a secret key securely that can then be used for subsequent encryption of messages
- Security relies on difficulty of computing discrete logarithms

Figure 21.9 The Diffie-Hellman Key Exchange Algorithm

Global Public Elements

q	Prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Generation of Secret Key by User A

$K = (Y_B)^{X_A} \bmod q$

Generation of Secret Key by User B

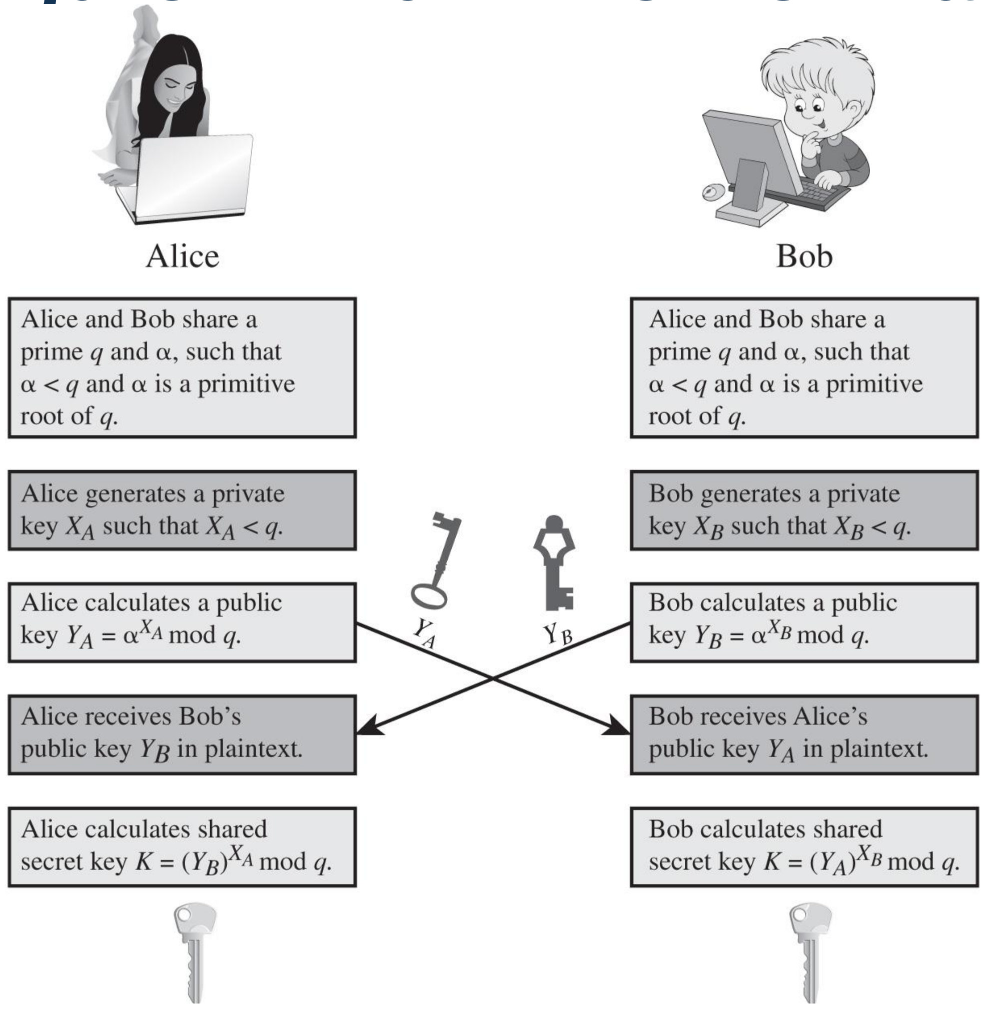
$K = (Y_A)^{X_B} \bmod q$

Sometime the *primitive root* is also called *generator.*
Hence, at the beginning, two parties exchange publicly a *Prime number* and a *Generator*

Diffie-Hellman Example

- Have
  - Prime number *q* = 353
  - Primitive root $\alpha = 3$
- A and B each compute their public keys
  - A computes $Y_A = 3^{97} \bmod 353 = 40$
  - B computes $Y_B = 3^{233} \bmod 353 = 248$
- Then exchange and compute secret key:
  - For A: $K = (Y_B)^{XA} \bmod 353 = 248^{97} \bmod 353 = 160$
  - For B $K = (Y_A)^{XB} \bmod 353 = 40^{233} \bmod 353 = 160$
- Attacker must solve:
  - $3^a \bmod 353 = 40$ which is hard
  - Desired answer is 97, then compute key as B does

Figure 21.10 Diffie-Hellman Key Exchange Protocol



Alice
Bob
Alice and Bob share a prime q and α, such that $\alpha < q$ and α is a primitive root of q.
Alice generates a private key X_A such that $X_A < q$.
Alice calculates a public key $Y_A = \alpha^{X_A} \bmod q$.
Alice receives Bob's public key Y_B in plaintext.
Alice calculates shared secret key $K = (Y_B)^{X_A} \bmod q$.
Alice and Bob share a prime q and α, such that $\alpha < q$ and α is a primitive root of q.
Bob generates a private key X_B such that $X_B < q$.
Bob calculates a public key $Y_B = \alpha^{X_B} \bmod q$.
Bob receives Alice's public key Y_A in plaintext.
Bob calculates shared secret key $K = (Y_A)^{X_B} \bmod q$.
Y_A
Y_B

Note 1: $(g^b \bmod p)^a = g^{ba} \bmod p$

i.e., exponentiation does not change module result

Note 2: $g^{ba} \bmod p \bmod p = g^{ba} \bmod p$

i.e., multiple module operations don't change the result

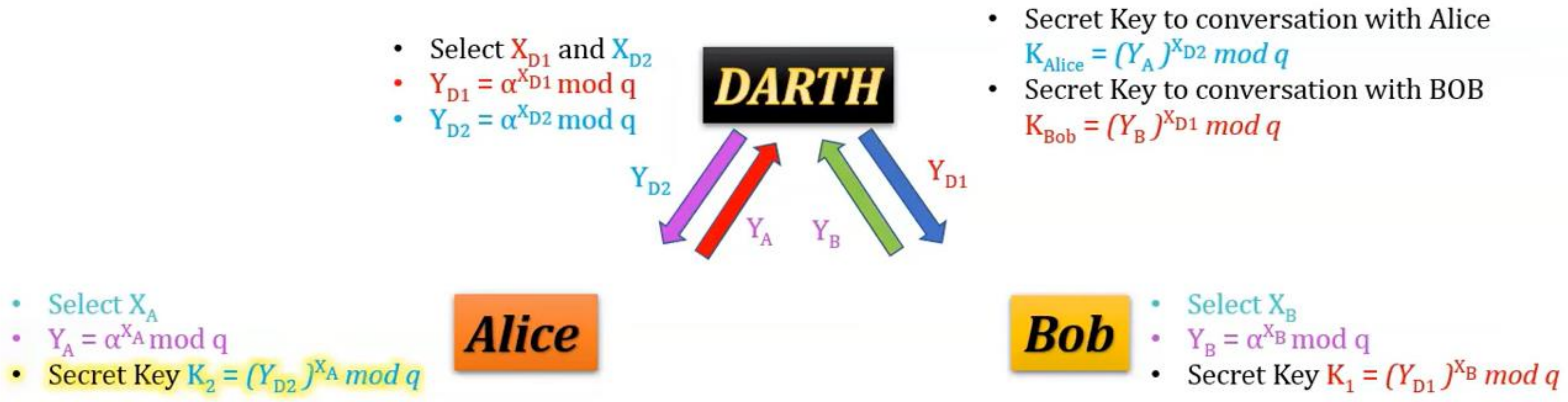
Attacking Diffie-Hellman mathematically

- Because X_A and X_B are private, an adversary only has the following ingredients to work with: q, α, Y_A, and Y_B.
- Thus, the adversary is forced to take a **discrete logarithm** to determine the key. For example, to determine the private key of user B, an adversary must compute: $X_B = \mathrm{dlog}_{\alpha, q} (Y_B)$. The adversary can then calculate the key K in the same manner as user B calculates it.

- The security of the Diffie-Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible.

- **Discrete Logarithm**: Given *a, b, p* with *a, b* non zero integers module *p*. The problem of finding *x*, such that
  $a^x = b \bmod p$
  is called **the discrete logarithm problem**, and we write: $x = dlog_{a,p}\, b$
  - (in our case of Diffie Hellman is $x = X_B$, , $a = \alpha$, $q = p$, $b = Y_B$)

Attacking Diffie-Hellman exchange protocol with Man-in-the-Middle



DARTH
• Select X_{D1} and X_{D2}
• $Y_{D1} = \alpha^{X_{D1}} \bmod q$
• $Y_{D2} = \alpha^{X_{D2}} \bmod q$
• Secret Key to conversation with Alice
$K_{Alice} = (Y_A)^{X_{D2}} \bmod q$
• Secret Key to conversation with BOB
$K_{Bob} = (Y_B)^{X_{D1}} \bmod q$
Y_{D2}
Y_A
Y_B
Y_{D1}
Alice
• Select X_A
• $Y_A = \alpha^{X_A} \bmod q$
• Secret Key $K_2 = (Y_{D2})^{X_A} \bmod q$
Bob
• Select X_B
• $Y_B = \alpha^{X_B} \bmod q$
• Secret Key $K_1 = (Y_{D1})^{X_B} \bmod q$

1. Alice sends Y_A to Bob, but Darth intercepts it (i.e., "*at some layer*" he acts as Bob)
2. Darth then sends Y_{D1} to Bob (lying that he is Alice)
3. Bob sends Y_B to Alice (who is actually Darth). Bob also calculates the shared key K_1 with Darth (i.e., the fake Alice)
4. Darth also calculates the shared key K_{Bob} which is actually the same of K_1. Now, Bob and Darth communicate with a secret key K_1.
5. Darth (acting as Bob) sends Y_{D2} to Alice. Darth also calculates the secret key K_{Alice} since he knows the Alice public key Y_A
6. Alice receives Y_{D2} from Darth (acting as Bob), and calculates the secret key K_2 who is actually the same of K_{Alice} . Now Alice and Darth share a secret key K_2
7. Now on, all the communications between Alice and Bob are intercepted and changed by Darth in both directions

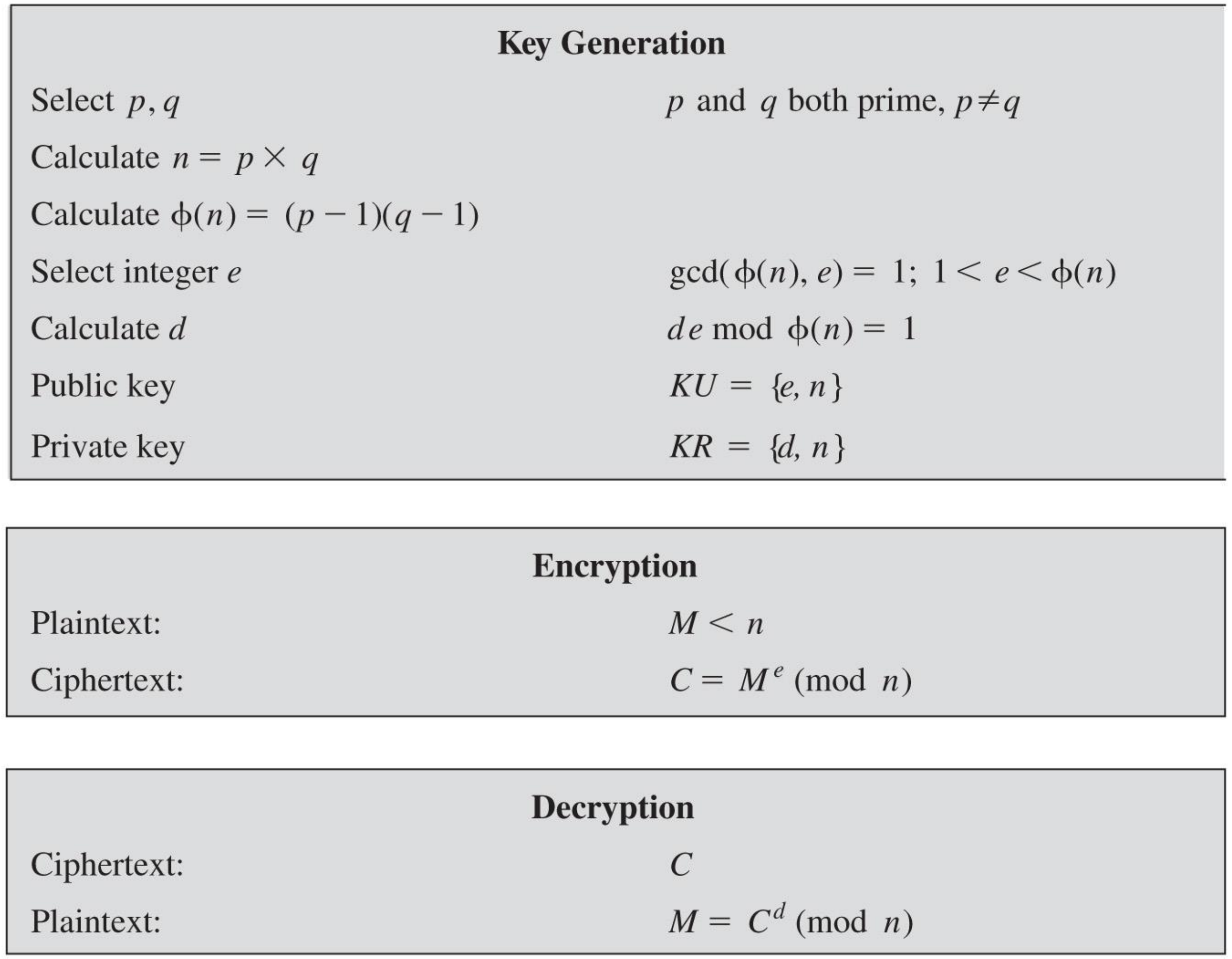
RSA

RSA Public-Key Encryption

- By Rivest, Shamir & Adleman of MIT in 1977
- Best known and widely used public-key algorithm
- Uses exponentiation of integers modulo a prime
- Encrypt: $C = M^e \bmod n$
- **Decrypt:** $M = C^d \bmod n = (M^e)^d \bmod n = M$
- Both sender and receiver know values of *n* and *e*
- Only receiver knows value of *d*
- Public-key encryption algorithm with public key

$PU = \{e, n\}$ and private key $PR = \{d, n\}$

Figure 21.7 The RSA Algorithm (for generation of a public-private key pair)



Key Generation

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$de \bmod \phi(n) = 1$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption

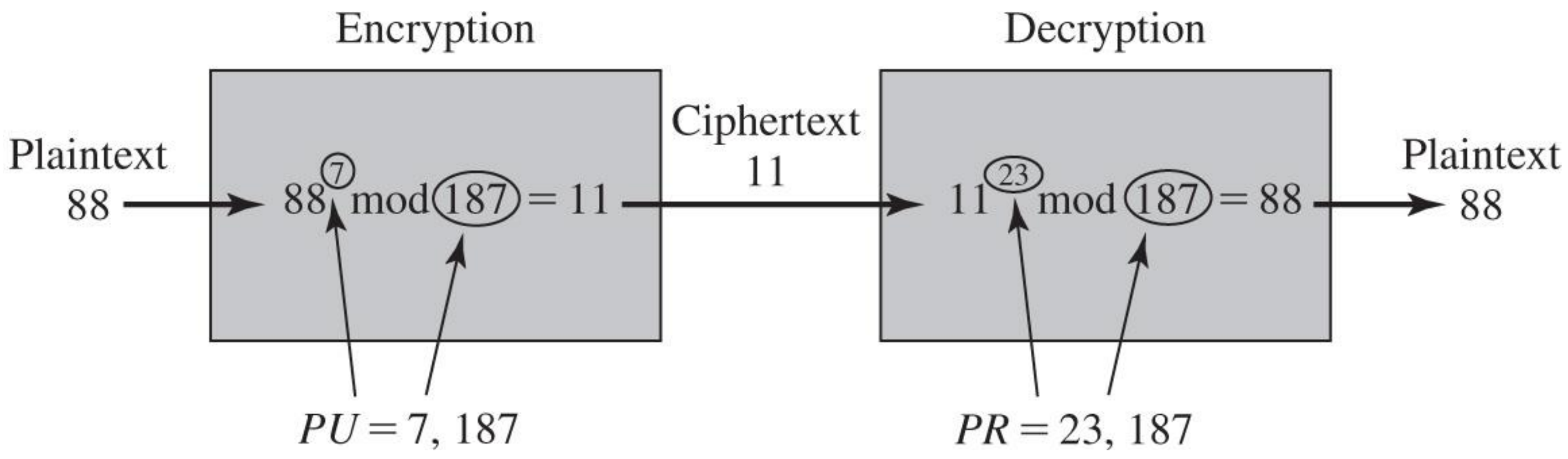
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption

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

Note: e is coprime of Φ(n)

Figure 21.8 Example of RSA Algorithm



Plaintext = 88
Public key = {7, 187}
Private key = {23,187}

Security of RSA

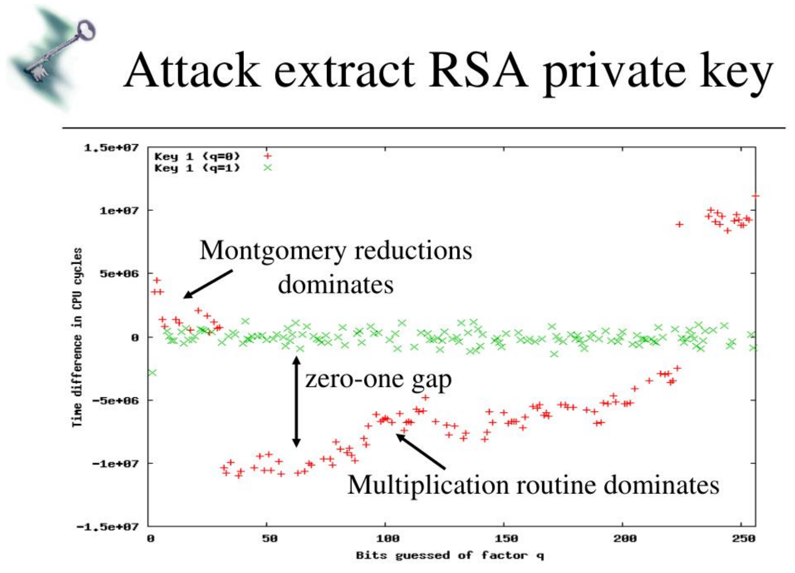
- **Brute force**
  - Involves trying all possible private keys. In general, for large keys we know it’s almost impossible to test all of them (statistically, half of it)
- **Mathematical attacks**
  - There are several approaches, all equivalent in effort to factoring the product of two primes
- **Timing attacks**
  - These depend on the running time of the decryption algorithm
- **Chosen ciphertext attacks**
  - This type of attack exploits properties of the RSA algorithm

Table 21.2 Progress in Factorization

Number of Decimal Digits	Number of Bits	Date Achieved
100	332	April 1991
110	365	April 1992
120	398	June 1993
129	428	April 1994
130	431	April 1996
140	465	February 1999
155	512	August 1999
160	530	April 2003
174	576	December 2003
200	663	May 2005
193	640	November 2005
232	768	December 2009

Timing Attacks

- Paul Kocher, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages
- Timing attacks are applicable not just to RSA, but also to other public-key cryptography systems
- This attack is alarming for two reasons:
  - It comes from a completely unexpected direction
  - It is a ciphertext-only attack



Timing Attack Countermeasures

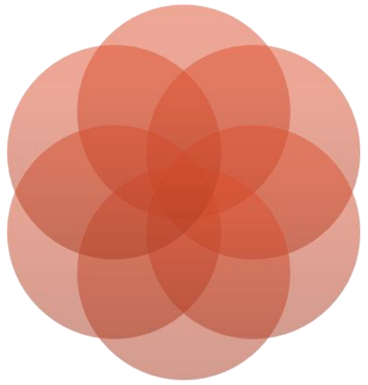
- Constant exponentiation time
  - Ensure that all exponentiations take the same amount of time before returning a result
  - This is a simple fix but does degrade performance

- Random delay
  - Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack
  - If defenders do not add enough noise, attackers could still succeed by collecting additional measurements to compensate for the random delays

- Blinding
  - Multiply the ciphertext by a random number before performing exponentiation
  - This process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack

Requirements for Public-Key Cryptosystems

1. Computationally easy to create key pairs
2. Computationally easy for sender knowing public key to encrypt messages
3. Computationally easy for receiver knowing private key to decrypt ciphertext
4. Computationally infeasible for opponent to determine private key from public key
5. Computationally infeasible for opponent to otherwise recover original message
6. Useful if either key can be used for each role, i.e., it is possible to encrypt/decrypt using the public/private key



Other relevant Asymmetric Encryption Algorithms

- Digital Signature Standard (DSS)
  - Provides only a digital signature function with SHA-1
  - Cannot be used for encryption or key exchange
- Elliptic curve cryptography (ECC)
  - Security like RSA, but with much smaller keys

Applications of public key

Digital Signatures

- NIST FIPS PUB 186-4 defines a digital signature as:
  - ”The result of a cryptographic transformation of data that, when properly implemented, provides a mechanism for verifying **origin authentication**, **data integrity** and **signatory non-repudiation**.”
- Thus, a digital signature is a data-dependent bit pattern, generated by an agent as a function of a file, message, or other form of data block
- The receiver can access the digital signature to verify that:
  1. the data block has been signed by the alleged signer
  2. the data block has not been altered since the signing
  3. The signer cannot repudiate the signature, hence content and creation of the message
- FIPS 186-4 specifies the use of one of three digital signature algorithms:
  - Digital Signature Algorithm (DSA)
  - RSA Digital Signature Algorithm
  - Elliptic Curve Digital Signature Algorithm (ECDSA)

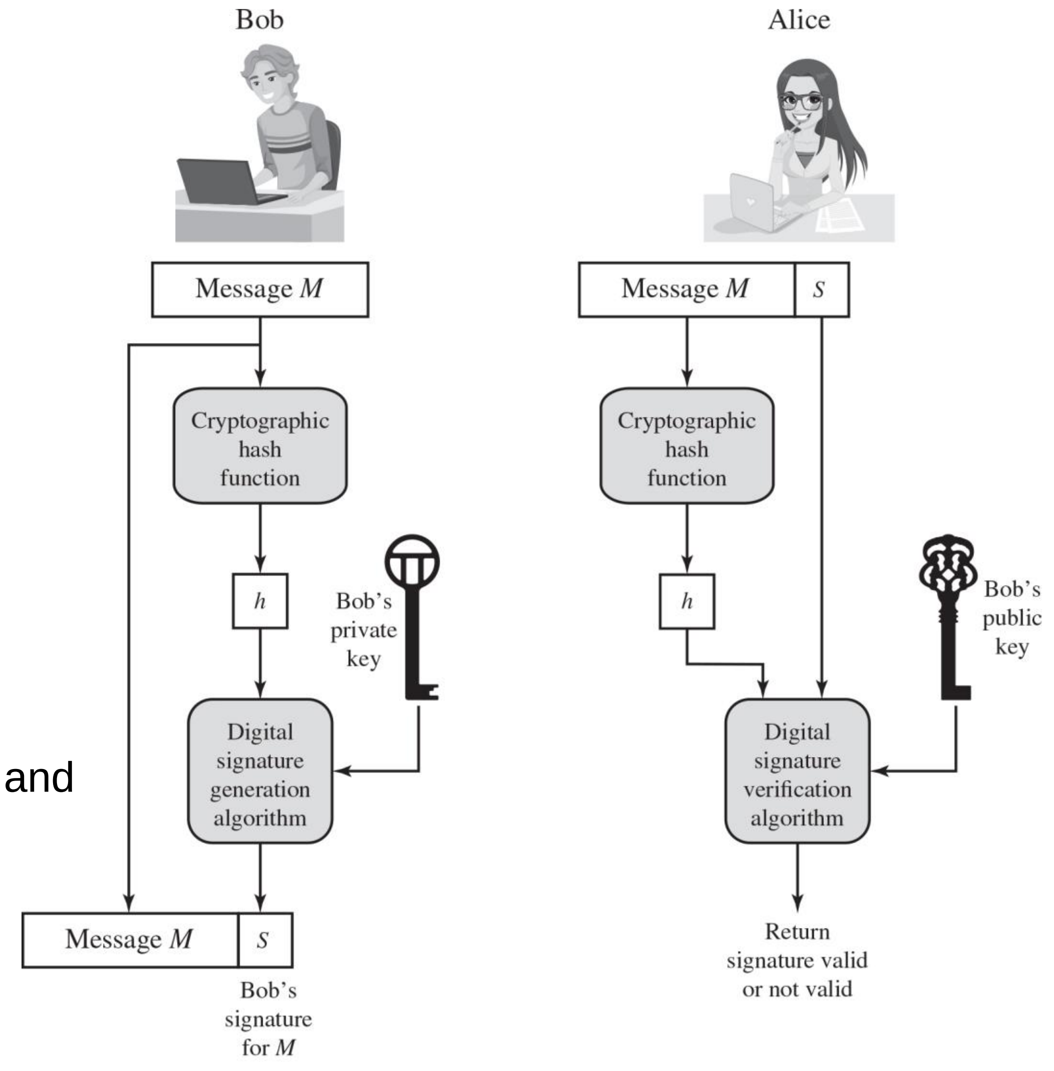
Figure 2.7 Simplified Depiction of Essential Elements of Digital Signature Process

Observation: digital signature does not provide confidentiality

Example of hash: SHA-512
Example of Public key algorithm: RSA

Question: make it sense for Bob to use his keys to encrypt the *content* of messages for achieving confidentiality?

Ans: **No**, because if he uses his private key, everybody will be able to decrypt with his public key and see the content.
On the other hand, if he uses his public key, only him can see the content…
Hence, we need to find another way to achieve confidentiality, i.e., to decide which key use for encryption of *content*



Bob
Alice
Message M
Message M
S
Cryptographic hash function
Cryptographic hash function
h
h
Bob's private key
Bob's public key
Digital signature generation algorithm
Digital signature verification algorithm
Message M
S
Bob's signature for M
Return signature valid or not valid

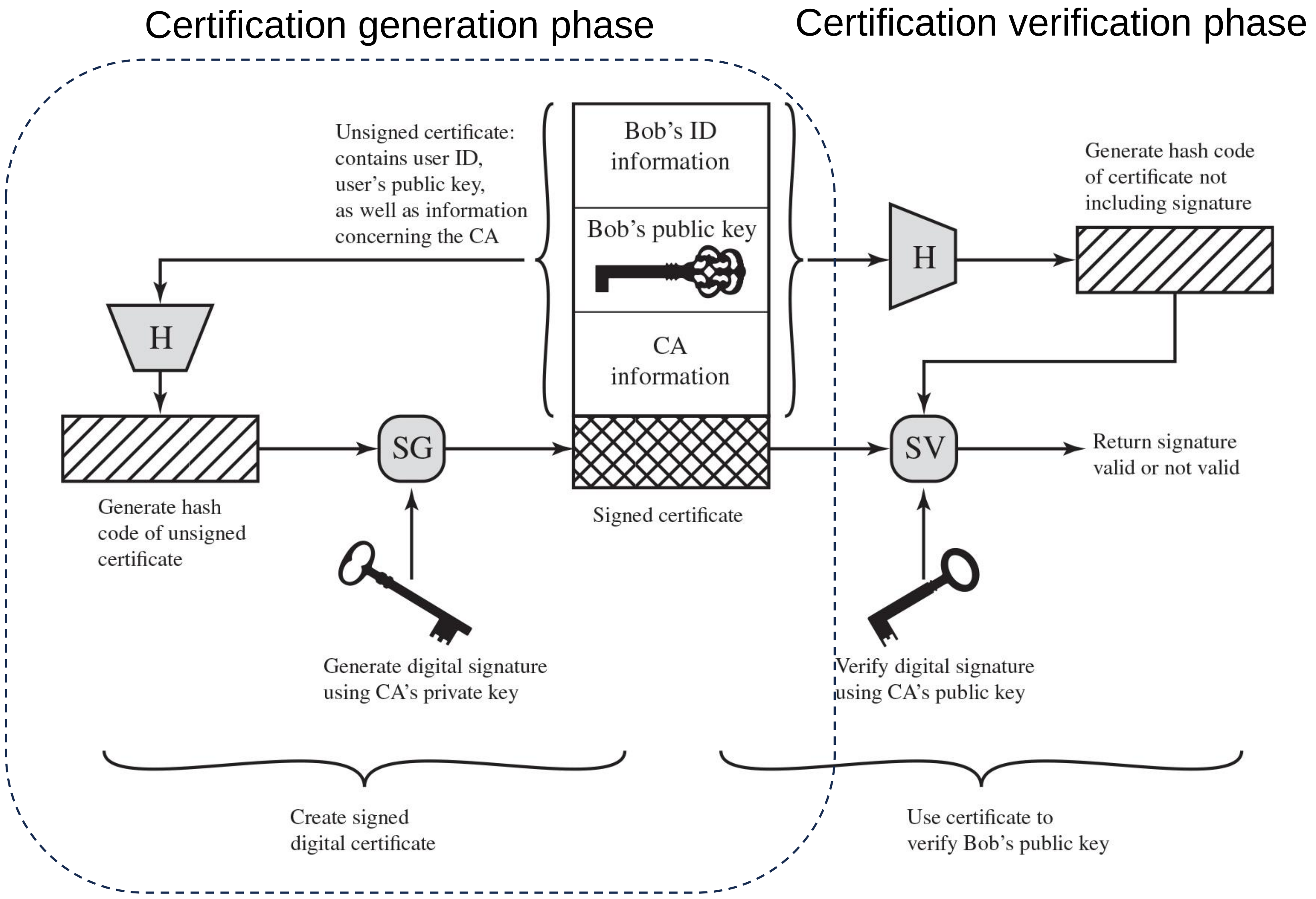
(a) Bob signs a message (b) Alice verifies the signature

A solution for confidential communication

- Any participant can send his or her public key to any other participant or broadcast the key to the community at large.
- Although this approach is convenient, it has a major weakness. Anyone can forge such a public announcement. That is, some user could pretend to be Bob and send a public key to another participant or broadcast such a public key. Until such time as Bob discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for Bob and can use the forged keys for authentication.
- The solution to this problem is the **public-key *certificate***.

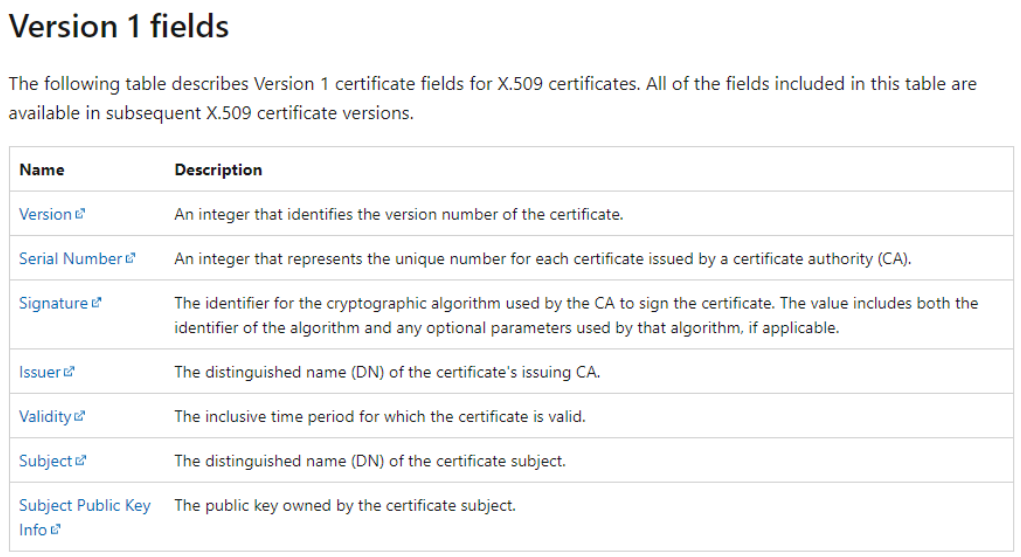
Figure 2.8 Public-Key Certificate Use

Note:
generation
and *verification*
phases happen usually in two
different moments
(even if, here, represented
in the same figure)


Certification generation phase
Certification verification phase
Unsigned certificate: contains user ID, user's public key, as well as information concerning the CA
Bob's ID information
Bob's public key
CA information
H
Generate hash code of unsigned certificate
SG
Signed certificate
Generate digital signature using CA's private key
Create signed digital certificate
H
Generate hash code of certificate not including signature
SV
Return signature valid or not valid
Verify digital signature using CA's public key
Use certificate to verify Bob's public key

Examples of standards for digital certificates and applications

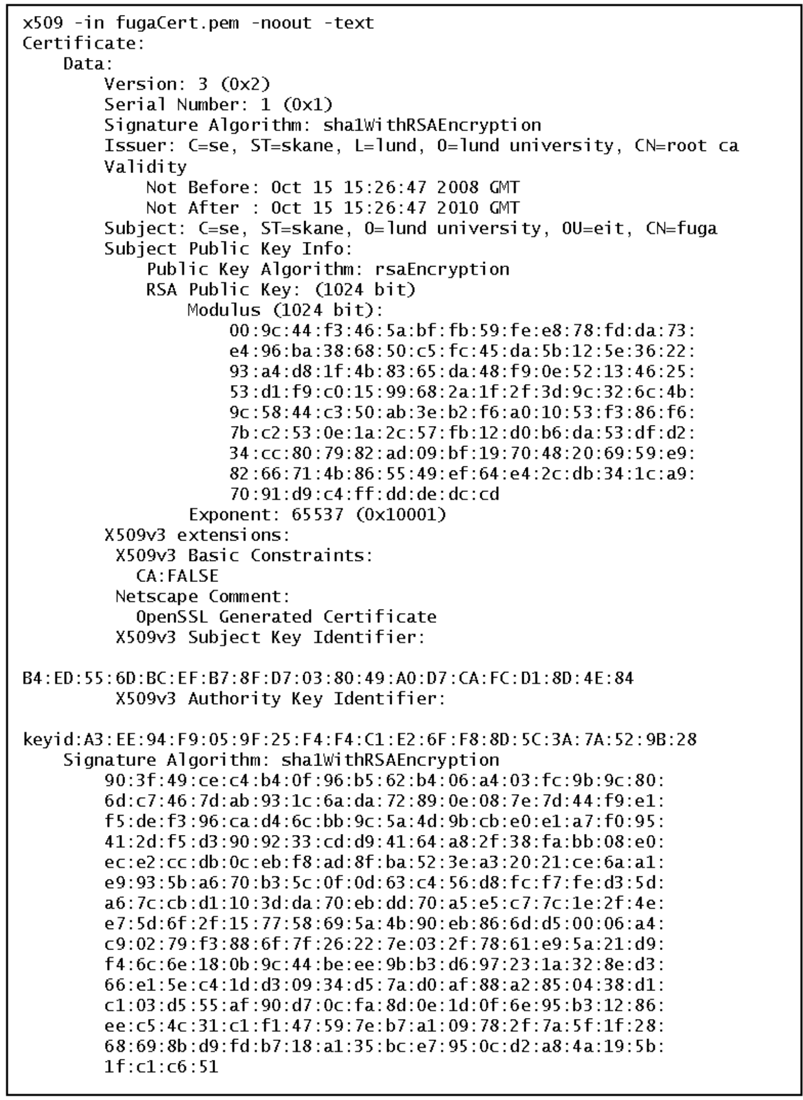
- Very common standard: X.509 (currently ver. 3)
- Very common applications: IPSec, TLS, SSH, S/MIME (emails)
- Purposes of certificates:
    - providing assurance for the public keys of Computers/machines
    - Individual users
    - Email addresses
    - Developers (code-signing certificates)



Version 1 fields

The following table describes Version 1 certificate fields for X.509 certificates. All of the fields included in this table are available in subsequent X.509 certificate versions.

Name	Description
Version	An integer that identifies the version number of the certificate.
Serial Number	An integer that represents the unique number for each certificate issued by a certificate authority (CA).
Signature	The identifier for the cryptographic algorithm used by the CA to sign the certificate. The value includes both the identifier of the algorithm and any optional parameters used by that algorithm, if applicable.
Issuer	The distinguished name (DN) of the certificate's issuing CA.
Validity	The inclusive time period for which the certificate is valid.
Subject	The distinguished name (DN) of the certificate subject.
Subject Public Key Info	The public key owned by the certificate subject.



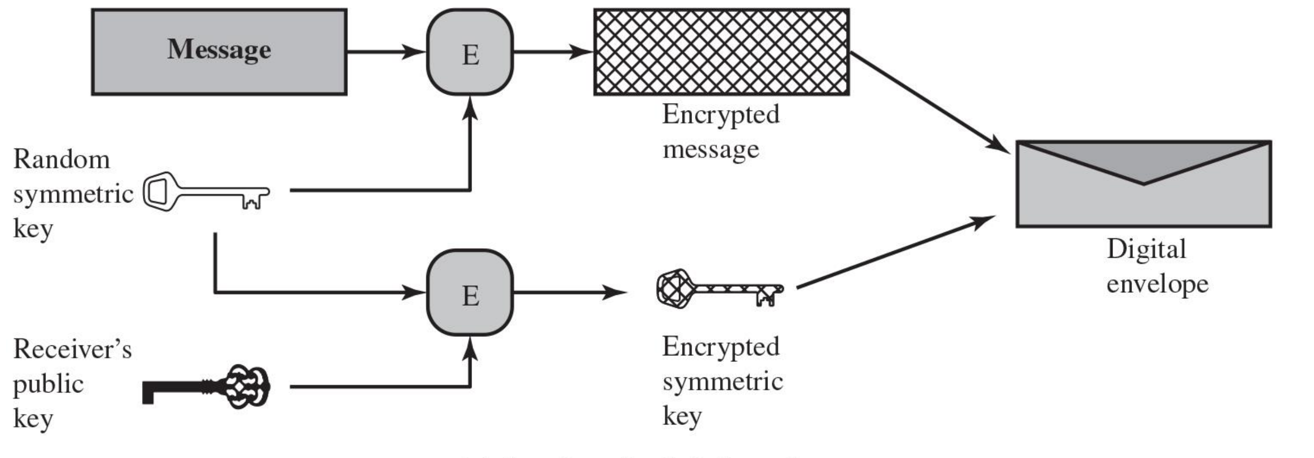
```
x509 -in fugaCert.pem -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=se, ST=skane, L=lund, O=lund university, CN=root ca
        Validity
            Not Before: Oct 15 15:26:47 2008 GMT
            Not After : Oct 15 15:26:47 2010 GMT
        Subject: C=se, ST=skane, O=lund university, OU=eit, CN=fuga
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:9c:44:f3:46:5a:bf:fb:59:fe:e8:78:fd:da:73:
                    e4:96:ba:38:68:50:c5:fc:45:da:5b:12:5e:36:22:
                    93:a4:d8:1f:4b:83:65:da:48:f9:0e:52:13:46:25:
                    53:d1:f9:c0:15:99:68:2a:1f:2f:3d:9c:32:6c:4b:
                    9c:58:44:c3:50:ab:3e:b2:f6:a0:10:53:f3:86:f6:
                    7b:c2:53:0e:1a:2c:57:fb:12:d0:b6:da:53:df:d2:
                    34:cc:80:79:82:ad:09:bf:19:70:48:20:69:59:e9:
                    82:66:71:4b:86:55:49:ef:64:e4:2c:db:34:1c:a9:
                    70:91:d9:c4:ff:dd:de:dc:cd
                Exponent: 65537 (0x10001)
        X509v3 extensions:
         X509v3 Basic Constraints:
           CA:FALSE
         Netscape Comment:
           OpenSSL Generated Certificate
         X509v3 Subject Key Identifier:

B4:ED:55:6D:BC:EF:B7:8F:D7:03:80:49:A0:D7:CA:FC:D1:8D:4E:84
         X509v3 Authority Key Identifier:

keyid:A3:EE:94:F9:05:9F:25:F4:F4:C1:E2:6F:F8:8D:5C:3A:7A:52:9B:28
    Signature Algorithm: sha1WithRSAEncryption
        90:3f:49:ce:c4:b4:0f:96:b5:62:b4:06:a4:03:fc:9b:9c:80:
        6d:c7:46:7d:ab:93:1c:6a:da:72:89:0e:08:7e:7d:44:f9:e1:
        f5:de:f3:96:ca:d4:6c:bb:9c:5a:4d:9b:cb:e0:e1:a7:f0:95:
        41:2d:f5:d3:90:92:33:cd:d9:41:64:a8:2f:38:fa:bb:08:e0:
        ec:e2:cc:db:0c:eb:f8:ad:8f:ba:52:3e:a3:20:21:ce:6a:a1:
        e9:93:5b:a6:70:b3:5c:0f:0d:63:c4:56:d8:fc:f7:fe:d3:5d:
        a6:7c:cb:d1:10:3d:da:70:eb:dd:70:a5:e5:c7:7c:1e:2f:4e:
        e7:5d:6f:2f:15:77:58:69:5a:4b:90:eb:86:6d:d5:00:06:a4:
        c9:02:79:f3:88:6f:7f:26:22:7e:03:2f:78:61:e9:5a:21:d9:
        f4:6c:6e:18:0b:9c:44:be:ee:9b:b3:d6:97:23:1a:32:8e:d3:
        66:e1:5e:c4:1d:d3:09:34:d5:7a:d0:af:88:a2:85:04:38:d1:
        c1:03:d5:55:af:90:d7:0c:fa:8d:0e:1d:0f:6e:95:b3:12:86:
        ee:c5:4c:31:c1:f1:47:59:7e:b7:a1:09:78:2f:7a:5f:1f:28:
        68:69:8b:d9:fd:b7:18:a1:35:bc:e7:95:0c:d2:a8:4a:19:5b:
        1f:c1:c6:51
```

Certification Authorities (CA)

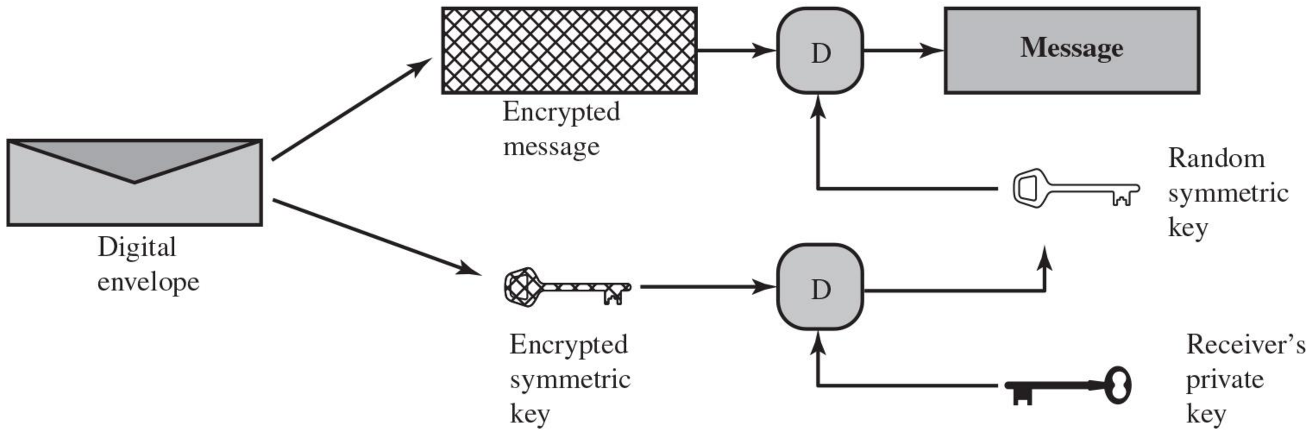
- Certificate authorities (CAs) are the glue that binds the public key infrastructure together. These neutral organizations offer notarization services for digital certificates. To obtain a digital certificate from a reputable CA, you must prove your identity to the satisfaction of the CA. The following list includes some of the major CAs who provide widely accepted digital certificates:
    - Symantec, IdenTrust, Amazon Web Services, GlobalSign, Comodo, Certum, GoDaddy, DigiCert, Secom, Entrust, Actalis, Trustwave

- Observations:
    - A certificate not signed from a CA, can be dangerous, so be careful to accept a certificate not signed from a CA
    - Even worst if you trust and add an unknown CA to your system. Usually, our systems are pre-configured with trusted Cas
    - Usually, CAs don’t expose their main certificate directly (*root certificate).* Rather, there is a chain of intermediate certificates to verify before arriving to the root certificate.
    - Locally, some organizations might use certificates not signed by a CA, but still trusted for internal use

Figure 2.9 Digital Envelopes



(a) Creation of a digital envelope



(b) Opening a digital envelope

Example of digital envelope:
PGP – Pretty Good Privacy
OpenPGP

PGP is an encryption program that provides cryptographic privacy and authentication for data communication. PGP is used for signing, encrypting, and decrypting texts, e-mails, files, directories, and whole disk partitions and to increase the security of e-mail communications.

Random Numbers

Uses include generation of:

- Keys for public-key algorithms
- Stream key for symmetric stream cipher
- Symmetric key for use as a temporary session key or in creating a digital envelope
- Handshaking to prevent replay attacks
- Session key

Random Number Requirements

Randomness

- Criteria:
  - Uniform distribution
    - Frequency of occurrence of each of the numbers should be approximately the same
  - Independence
    - No one value in the sequence can be inferred from the others

Unpredictability

- Each number is statistically independent of other numbers in the sequence
- Opponent should not be able to predict future elements of the sequence on the basis of earlier elements

Random Versus Pseudorandom

- Cryptographic applications typically make use of algorithmic techniques for random number generation
  - Algorithms are deterministic and therefore produce sequences of numbers that are not statistically random
- Pseudorandom numbers are:
  - Sequences produced that satisfy statistical randomness tests
  - Likely to be predictable
- True random number generator (TRNG):
  - Uses a nondeterministic source to produce randomness
  - Most operate by measuring unpredictable natural processes
    - e.g., radiation, gas discharge, leaky capacitors
  - Increasingly provided on modern processors

Practical Application: Encryption of Stored Data

- Common to encrypt transmitted data
- Much less common for stored data
  - There is often little protection beyond domain authentication and operating system access controls
  - Data are archived for indefinite periods
  - Even though erased, until disk sectors are reused data are recoverable
- Approaches to encrypt stored data:
  - Use a commercially available encryption package
  - Back-end appliance
  - Library based tape encryption
  - Background laptop/PC data encryption

Conclusions about Cryptography

- Classified along three independent dimensions:
  - The type of operations used for transforming plaintext to ciphertext
    - Substitution – each element in the plaintext is mapped into another element
    - Transposition – elements in plaintext are rearranged
    - Mathematical – Exploiting mathematical functions like RSA and D-H
  - The number of keys used
    - Sender and receiver use same key – symmetric
    - Sender and receiver each use a different key - asymmetric
  - The way in which the plaintext is processed
    - Block cipher – processes input one block of elements at a time
    - Stream cipher – processes the input elements continuously

Observations and misconceptions

- Symmetric key is not necessarily worst than a public key algorithm, it mostly depends on the length of the key and the mathematical functions required to be broken
- Because of the computational overhead of current public-key encryption schemes, there seems no foreseeable likelihood that symmetric encryption will be abandoned
- For public-key key distribution, some form of protocol is needed, often involving a central agent, and the procedures involved are no simpler or any more efficient than those required for symmetric encryption.

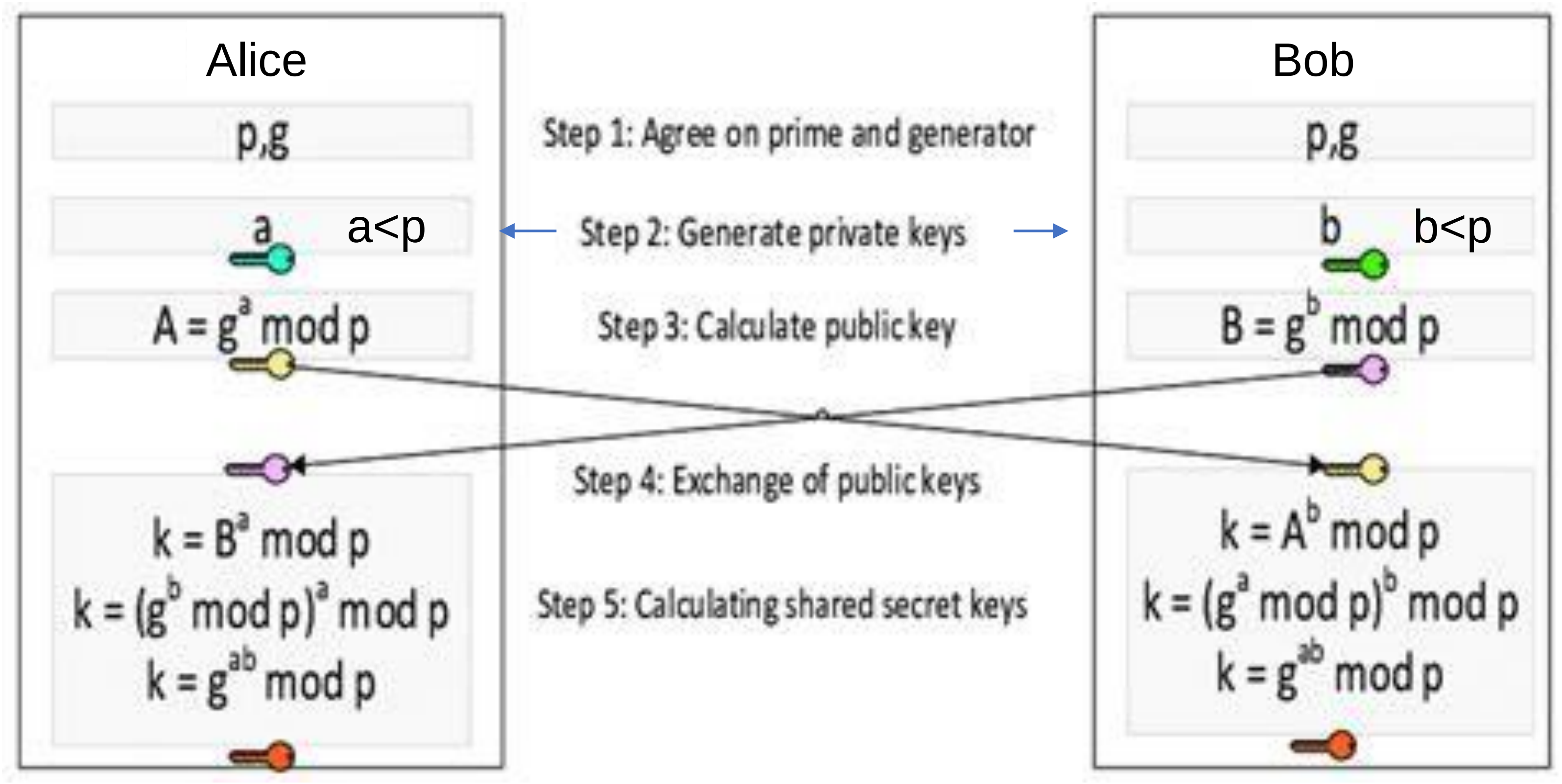
Summary (1 of 2)

- Confidentiality with symmetric encryption
  - Symmetric encryption
  - Symmetric block encryption algorithms
  - Stream ciphers
- Message authentication and hash functions
  - Authentication using symmetric encryption
  - Message authentication without message encryption
  - Secure hash functions
  - Other applications of hash functions
- Random and pseudorandom numbers
  - The use of random numbers
  - Random versus pseudorandom

Summary (2 of 2)

- Public-key encryption
    - Structure
    - Applications for public-key cryptosystems
    - Requirements for public-key cryptography
    - Asymmetric encryption algorithms
- Digital signatures and key management
    - Digital signature
    - Public-key certificates
    - Symmetric key exchange using public-key encryption
    - Digital envelopes
- Practical Application: Encryption of Stored Data

Appendix: Another example of Diffie-Hellman


Alice
Bob
p,g
p,g
Step 1: Agree on prime and generator
a
a<p
b
b<p
Step 2: Generate private keys
A = g^a mod p
B = g^b mod p
Step 3: Calculate public key
Step 4: Exchange of public keys
k = B^a mod p
k = (g^b mod p)^a mod p
k = g^ab mod p
k = A^b mod p
k = (g^a mod p)^b mod p
k = g^ab mod p
Step 5: Calculating shared secret keys

Note 1: $(g^b \bmod p)^a = g^{ba} \bmod p$

i.e., exponentiation does not change module result

Note 2: $g^{ba} \bmod p \bmod p = g^{ba} \bmod p$

i.e., multiple module operations don't change the result