

DI-FCT-UNL

Segurança de Redes e Sistemas de Computadores
Network and Computer Systems Security

Mestrado Integrado em Engenharia Informática
MSc Course: Informatics Engineering
2º Semestre, 2018/2019

5. Public Key Cryptography and Digital Signatures

Symmetric vs. Asymmetric Encryption

Exercise: Remember the OSI X.800 Framework for Attack-Typology, Security Properties and Security Mechanisms

- Remember the target for symmetric encryption algorithms, secure hash functions, message authentication codes (CMACs and HMACs), and the related methods and techniques.
- What is the target for asymmetric encryption algorithms methods and techniques

Symmetric Encryption Mechanisms

	Release of message contents	Traffic analysis	Masquerade	Replay	Modification of messages	Denial of service
Peer entity authentication			Y			
Data origin authentication			Y			
Access control			Y			
Confidentiality	Y					
Traffic flow confidentiality		Y				
Data integrity				Y	Y	
Non-repudiation			Y			
Availability						Y

Cryptography methods, Algorithms, models, techniques

	Release of message contents	Traffic analysis	Masquerade	Replay	Modification of messages	Denial of service
Encipherment	Y					
Digital signature			Y	Y	Y	
Access control	Y	Y	Y	Y		Y
Data integrity				Y	Y	
Authentication exchange	Y		Y	Y		Y
Traffic padding		Y				
Routing control	Y	Y				Y
Notarization			Y	Y	Y	

**Symmetric Crypto:
for Confidentiality**

Service	Encipherment	Digital signature	Access control	Data integrity	Authentication exchange	Traffic padding	Routing control	Notarization
Peer entity authentication	Y	Y			Y			
Data origin authentication	Y	Y						
Access control			Y					
Confidentiality	Y						Y	
Traffic flow confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Non-repudiation		Y		Y				Y
Availability				Y	Y			

Symmetric Encryption Mechanisms

	Release of message contents	Traffic analysis	Masquerade	Replay	Modification of messages	Denial of service
Peer entity authentication			Y			
Data origin authentication			Y			
Access control			Y			
Confidentiality	Y					
Traffic flow confidentiality		Y				
Data integrity				Y	Y	
Non-repudiation			Y			
Availability						Y

Key Dist. Protocols
(No Peer-Authent. Assumptions,
Shared Master Keys
KDCs
No Perfect
Backward/Forward Secrecy)

Service	Enciph-erment	Digital signature	Access control	Data integrity	Authenti-cation exchange	Traffic padding	Routing control	Notari-zation
Peer entity authentication	Y	Y			Y			
Data origin authentication	Y	Y						
Access control			Y					
Confidentiality	Y						Y	
Traffic flow confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Non-repudiation		Y		Y				Y
Availability				Y	Y			

Cryptography methods,
Algorithms, models, techniques

Release of message contents	Traffic analysis	Masquerade	Replay	Modification of messages	Denial of service
Encipherment	Y				
Digital signature		Y	Y	Y	
Access control	Y	Y	Y		Y
Data integrity			Y	Y	
Authentication exchange	Y	Y	Y		Y
Traffic padding		Y			
Routing control	Y	Y			Y
Notarization		Y	Y	Y	

Sec Hashing + MAC Mechanisms

	Release of message contents	Traffic analysis	Masquerade	Replay	Modification of messages	Denial of service
Peer entity authentication			Y			
Data origin authentication			Y			
Access control			Y			
Confidentiality	Y					
Traffic flow confidentiality		Y				
Data integrity				Y	Y	
Non-repudiation			Y			
Availability						Y

Sec. Hashing Algorithms + MAC models, constructions and techniques

	Release of message contents	Traffic analysis	Masquerade	Replay	Modification of messages	Denial of service
Encipherment	Y					
Digital signature			Y	Y	Y	
Access control	Y	Y	Y	Y		Y
Data integrity				Y	Y	
Authentication exchange	Y		Y	Y		Y
Traffic padding		Y				
Routing control	Y	Y				Y
Notarization			Y	Y	Y	

Sec Hashing and MACs (HMACs and CMACs)

Service	Encipherment	Digital signature	Access control	Data integrity	Authentication exchange	Traffic padding	Routing control	Notarization
Peer entity authentication	Y	Y			Y			
Data origin authentication	Y	Y						
Access control			Y					
Confidentiality	Y						Y	
Traffic flow confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Non-repudiation		Y		Y				Y
Availability				Y	Y			

Sec Hashing + MAC Mechanisms

	Release of message contents	Traffic analysis	Masquerade	Replay	Modification of messages	Denial of service
Peer entity authentication			Y			
Data origin authentication			Y			
Access control			Y			
Confidentiality	Y					
Traffic flow confidentiality		Y				
Data integrity				Y	Y	
Non-repudiation			Y			
Availability						Y

Asymmetric Cryptography (Public Key Algorithms + Techniques + Constructions)

Service	Enciph-erment	Digital signature	Access control	Data integrity	Authenti-cation exchange	Traffic padding	Routing control	Notari-zation
Peer entity authentication	Y	Y			Y			
Data origin authentication	Y	Y						
Access control			Y					
Confidentiality	Y						Y	
Traffic flow confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Non-repudiation		Y		Y				Y
Availability				Y	Y			

	Release of message contents	Traffic analysis	Masquerade	Replay	Modification of messages	Denial of service
Encipherment	Y					
Digital signature			Y	Y	Y	
Access control	Y	Y	Y	Y		Y
Data integrity				Y	Y	
Authentication exchange	Y		Y	Y		Y
Traffic padding		Y				
Routing control	Y	Y				Y
Notarization			Y	Y	Y	

Public-Key Cryptography

- Probably most significant advance in the 3000 year history of cryptography ...
 - [https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))
 - https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange
 - https://en.wikipedia.org/wiki/Elliptic-curve_cryptography
- Clifford Cocks (British Intelligence/GCHQ first in 1973, declassif. in 1997, Whitfield Diffie & Martin Hellman, Stanford University (1976), Ron Rivest, Adi Shamir, Leonard Aldeman (1978), Neal Koblitz (1985) and Victor Miller (1985)
- **Foundations:** number theory concepts and functions (RSA, DSA, ElGammal, Diffie Hellman) and algebraic structures of elliptic curves over finite fields (ECC)
 - ... | But current research together w/innovative symmetric encryption techniques

Some Public-Key Methods

(Asymmetric Cryptographic Algorithms and Constructions)

Examples.

Note) Targeted for different uses:

Algorithm	Encryption/Decryption		Digital Signature		Key Exchange	
RSA	Yes	✓	Yes	✓	Yes	✓
Elliptic Curve	Yes	✓	Yes	✓	Yes	✓
Diffie-Hellman	No	✗	No	✗	Yes	✓
DSS	No	✗	Yes	✓	No	✗
ElGamal	Yes	✓	Yes	✓	No	✓

Outline

- **Public Key Cryptography**
 - Public Key Cryptography and Use of Public Key Methods
 - Properties of Public Key Algorithms
 - Public-Key Cryptography Algorithms
 - Digital Signatures

Outline

- **Public Key Cryptography**



- Public Key Cryptography and Use of Public Key Methods
- Properties of Public Key Algorithms
- Public-Key Cryptography Algorithms
- Digital Signatures

Public-Key Cryptography

Public-key (or Asymmetric) cryptography involves:

Two keys (or a **key-pair**):

- a **public-key**, known by anybody: can be used to **encrypt messages**, and **verify signatures**
- a **private-key**, known only to the recipient: used to **decrypt messages**, and **sign (create) digital signatures**

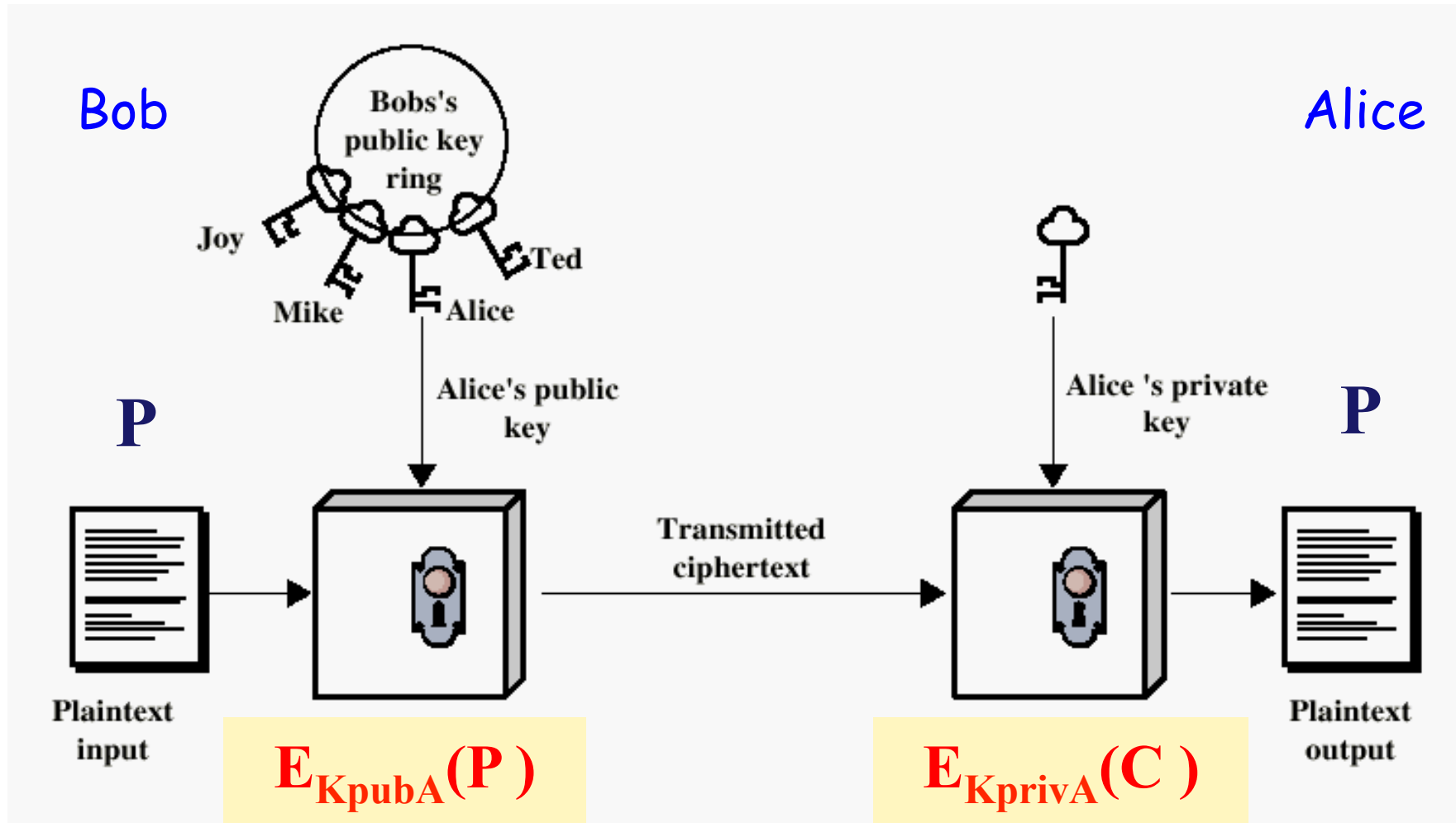
What we encrypt with one key, we can decrypt with the other key of the pair*

Same function (**same computation**) for encryption and for decryption

(*) valid property for algorithms used for encryption/decryption

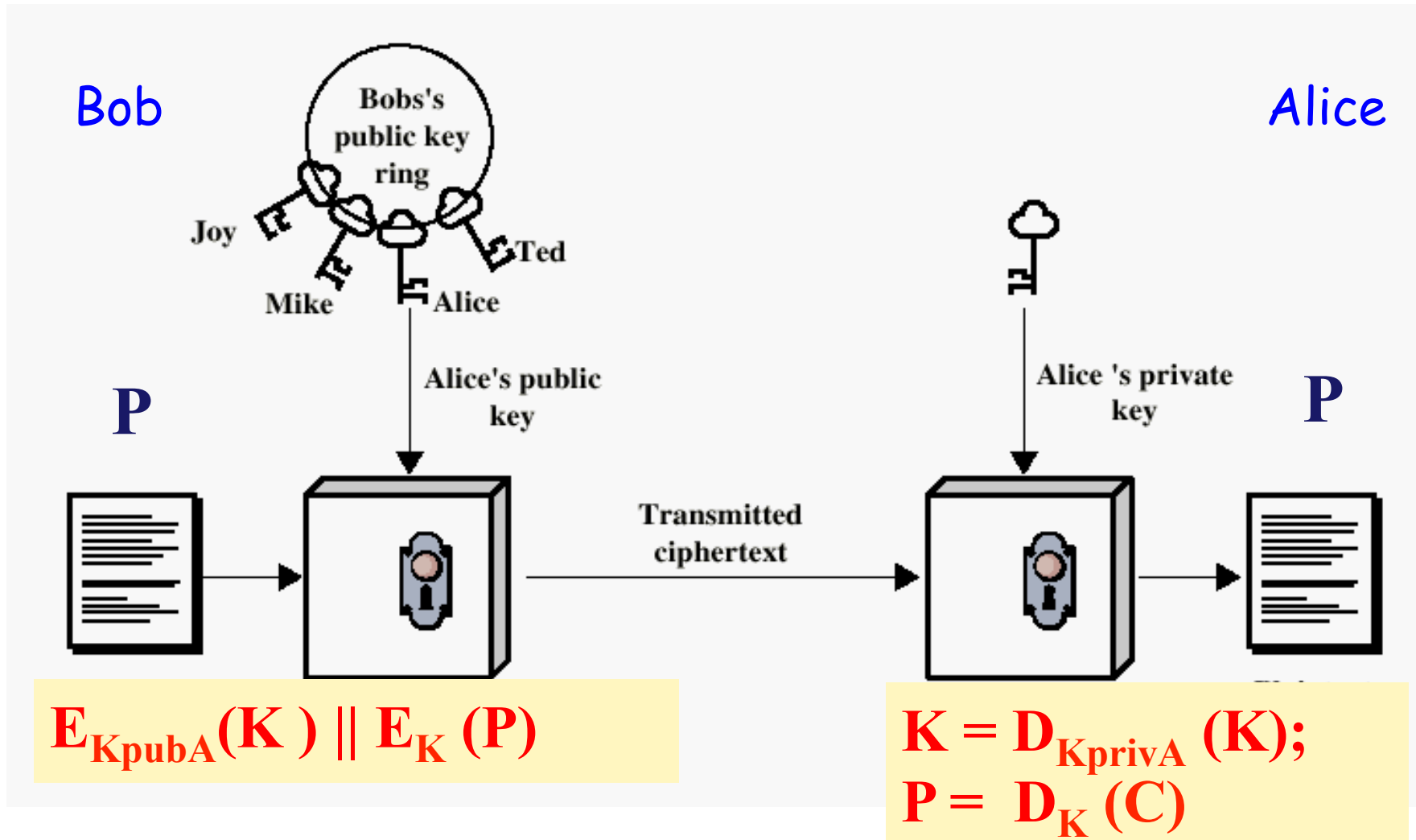
Encryption using a Public-Key System

Usage for Confidentiality



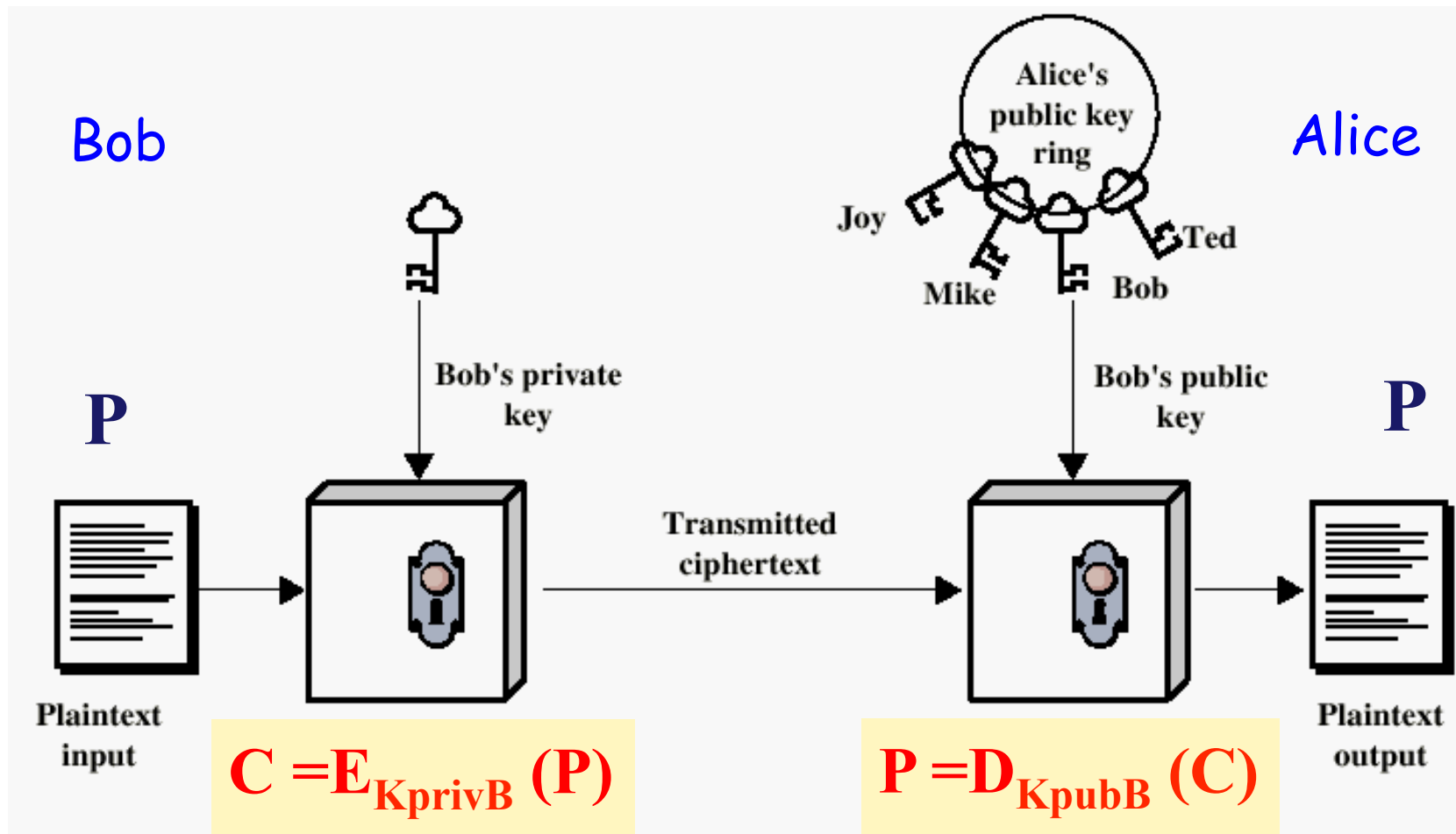
Encryption using a Public-Key System

Better Usage for Confidentiality ! Why ?



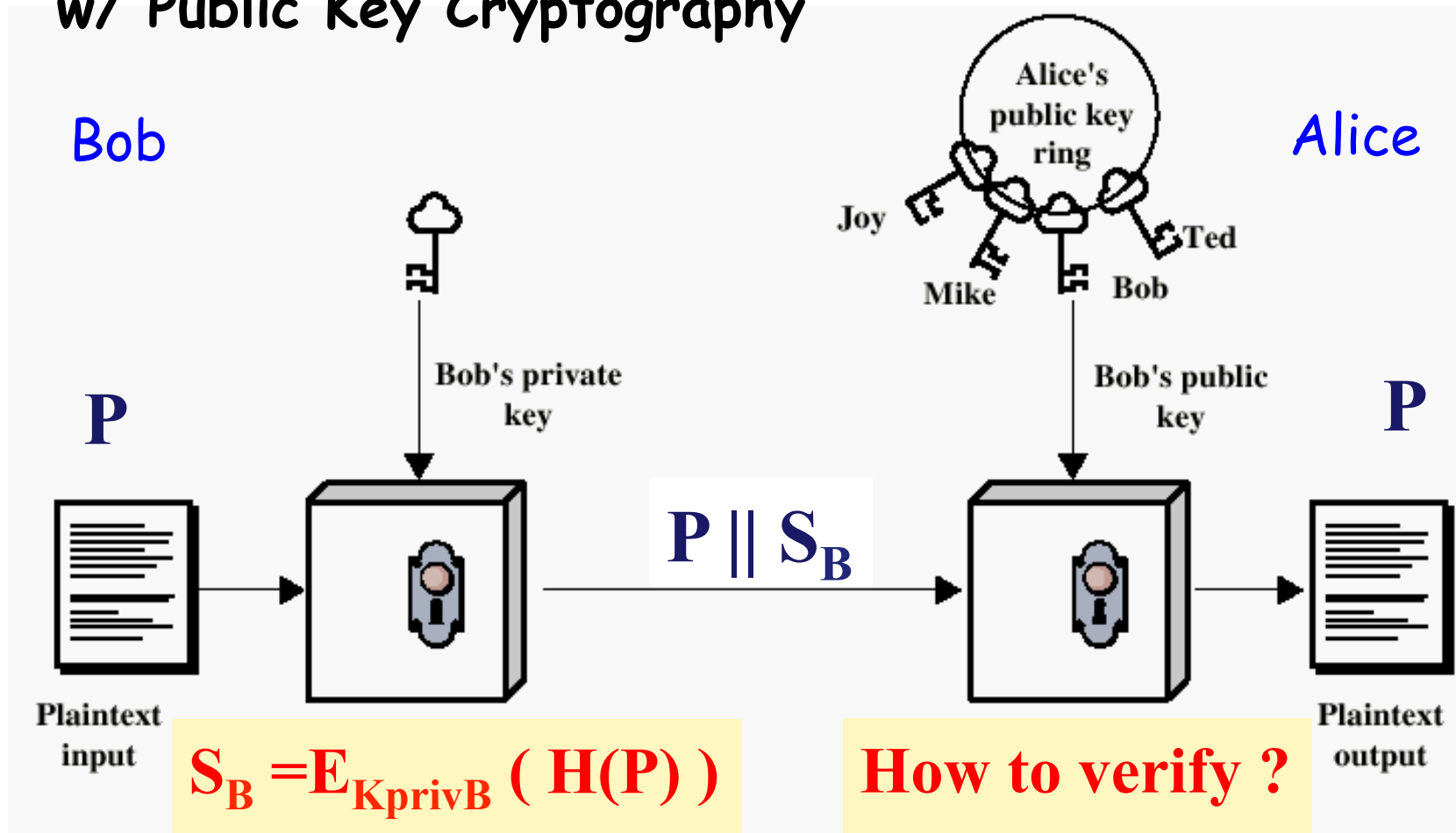
Authentication using Public-Key System

Usage for Authentication (Peer-Authentication)



Authentication using Public-Key System

Principle of Digital Signatures w/ Public Key Cryptography



Use and Programming w/ Asymmetric Algorithms

Demo w/ Java Programming (JCA/JCE)
(More detail in Lab 5)

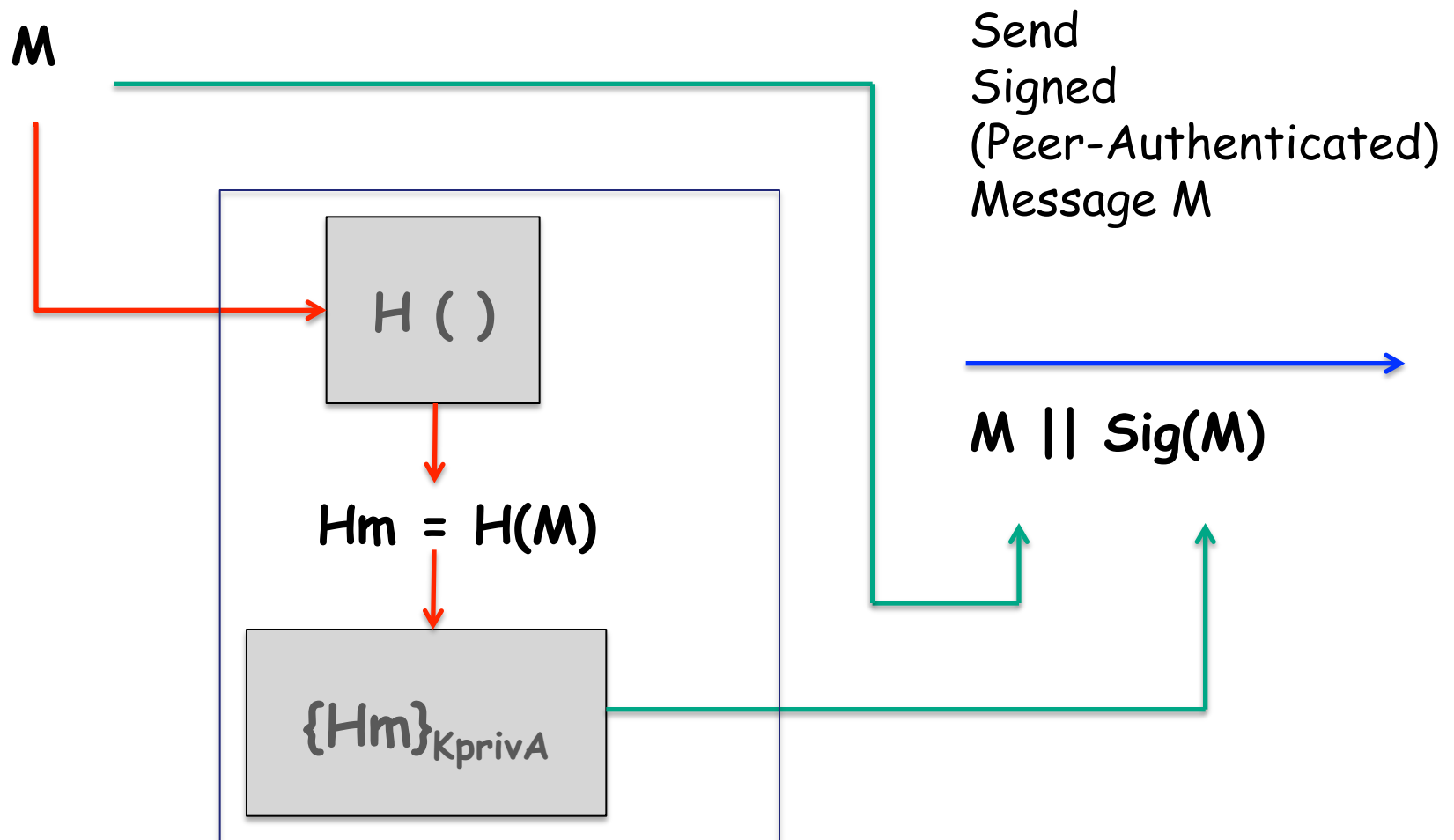
Keypair generation w/ openssl:

```
openssl genrsa -des3 -out private.pem 2048
openssl rsa -in private.pem -outform PEM -out public.pem
less public.pem
openssl rsa -in private.pem -out private_unencrypted.pem -outform PEM
less private.pem
less public.pem
```

Will see also in Lab the use of the Java keytool and how to manage public/private Keys in Java Programs

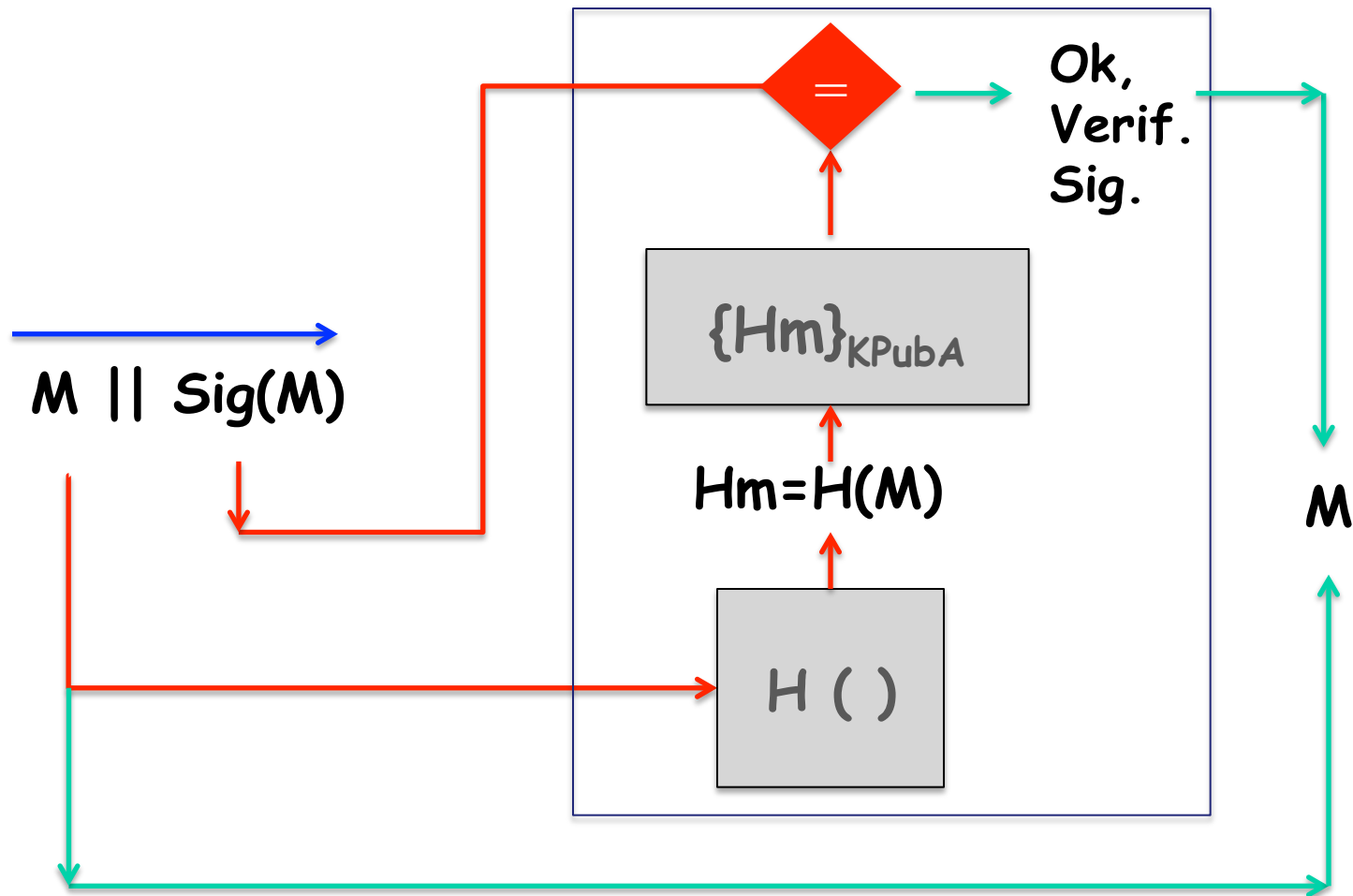
Base Scheme for a Digital Signature

A message sent with a Digital Signature Proof



Verification of Digital Signatures

Verification of the received M and Digital Signature



Construction for Confidentiality and Authentication

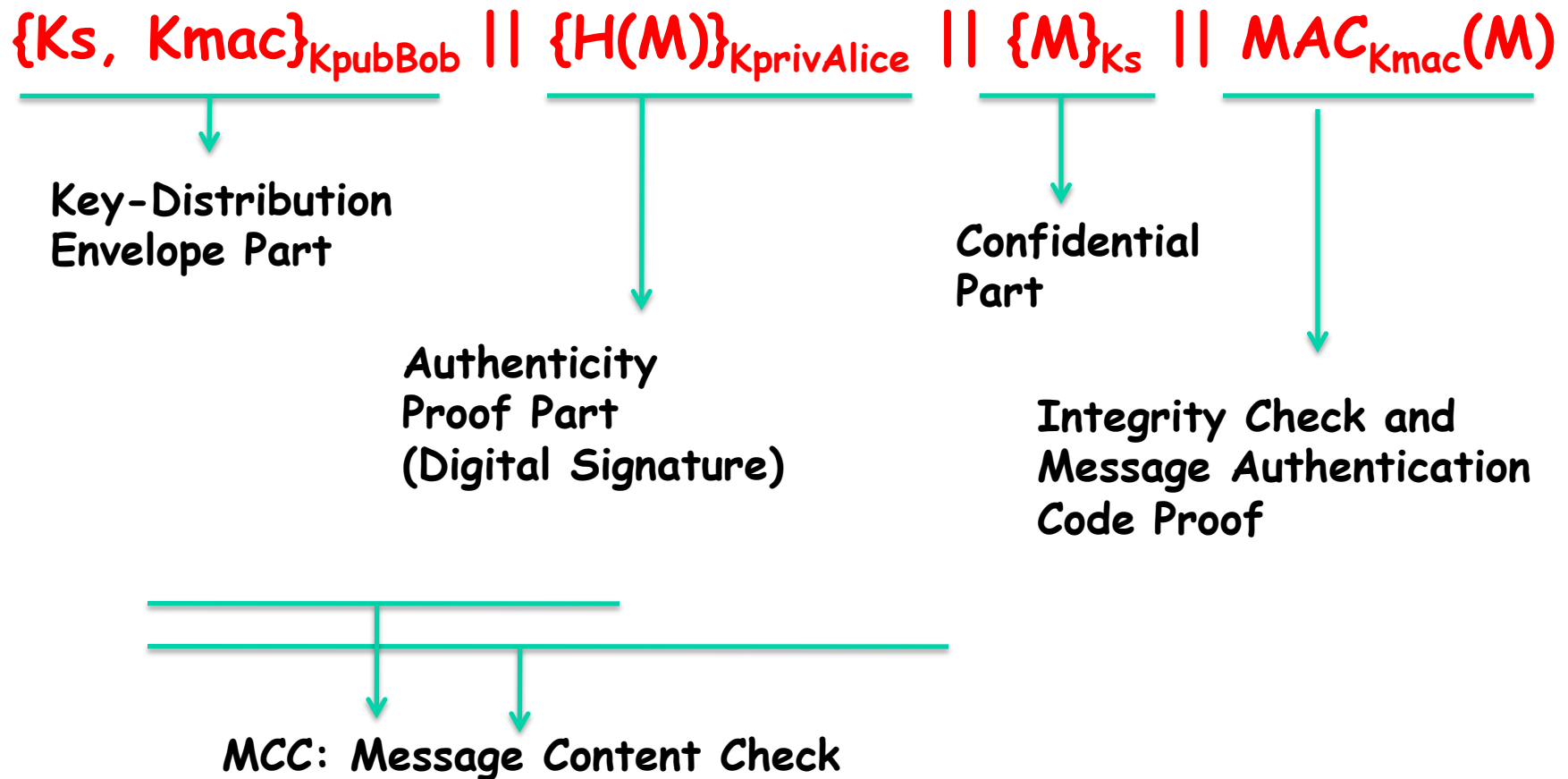
Authentication + Key Establishment + Confidentiality + Message Code Authentication + Integrity

- Verification by each principal, based on correct and non-repudiable associations (principal ID, PublicKey)
- Or (principal ID, Public Key) certified associations
- **Key exchange:** Two sides can cooperate to exchange a session key (or security association parameters): hybrid use of asymmetric and symmetric cryptography
 - Ex., Key generated by Alice and distributed to Bob:
 $\{ K_s \}_{K_{pubBob}} \parallel \{ H(M) \}_{K_{privAlice}} \parallel \{ M \}_{K_s}$

Note) Some Assym. Methods are only specifically targeted for Key-Exchange: ex., Diffie Hellman

Hybrid construction principle for secure channel establishment

Suppose Alice sends to Bob ...



Hybrid construction principle for secure channel establishment

Suppose Alice sends to Bob ...

$\{K_s, K_{mac}\}_{K_{pubBob}} \parallel \{H(M)\}_{K_{privAlice}} \parallel \{M\}_{K_s} \parallel MAC_{K_{mac}}(M)$

Is Ok for Secure Channel ?

Ok if we have unilateral Alice authentication, and keys generated by Alice ...

But what if we need Mutual Authentication ?

And what if we want contributive Keys

The RSA Cryptosystem

Authentication and Key Distribution
using Asymmetric Cryptography

The problem: Establishment of Secure Channel

- Alice and Bob want to establish a channel, with security properties as defined in the OSI X.800 Security Framework, to protect: Authentication (peer authentication and message authentication), Confidentiality, and Integrity
- Need to establish keys (and any other security association parameters) in a secure way
- Protection against the attacks-typology as defined in the OSI X.800 security framework

Needham-Schroeder with asymmetric cryptography

1. A -> PKC : A,B

2. PKC -> A : { KBpub, B } KPKCpriv

3. A -> B: { Na, A }KBpub

4. B obtains (in a trust way) the KpubA from the PKC

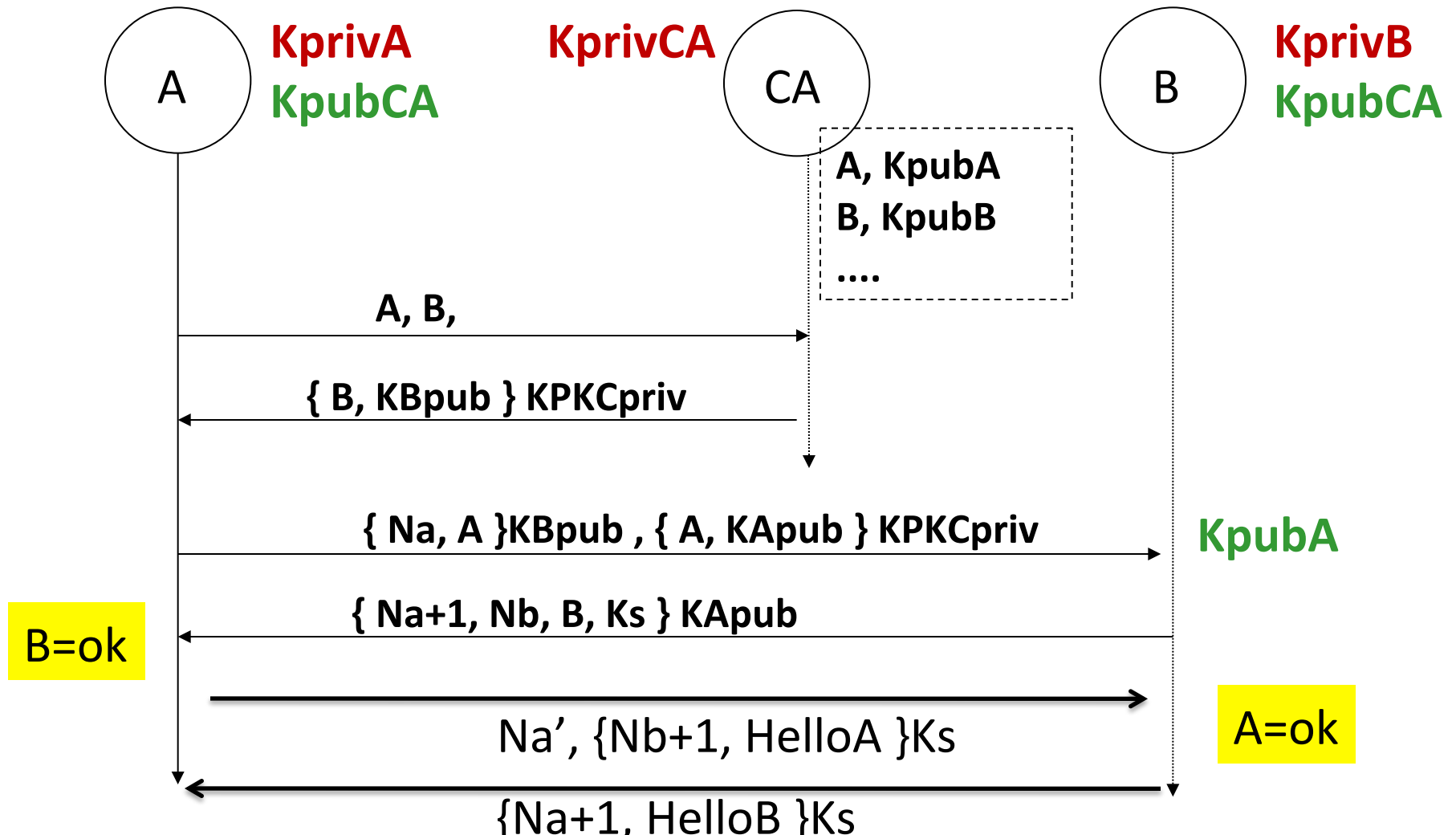
5. B -> A: { Na+1, Nb, B, Ks } KApub

In this case, B generates the session key

6. A -> B: { Nb+1 } Ks

**A replies with the response to the nonce Nb, which authenticates A.
Question: is B authenticated from the A viewpoint ?**

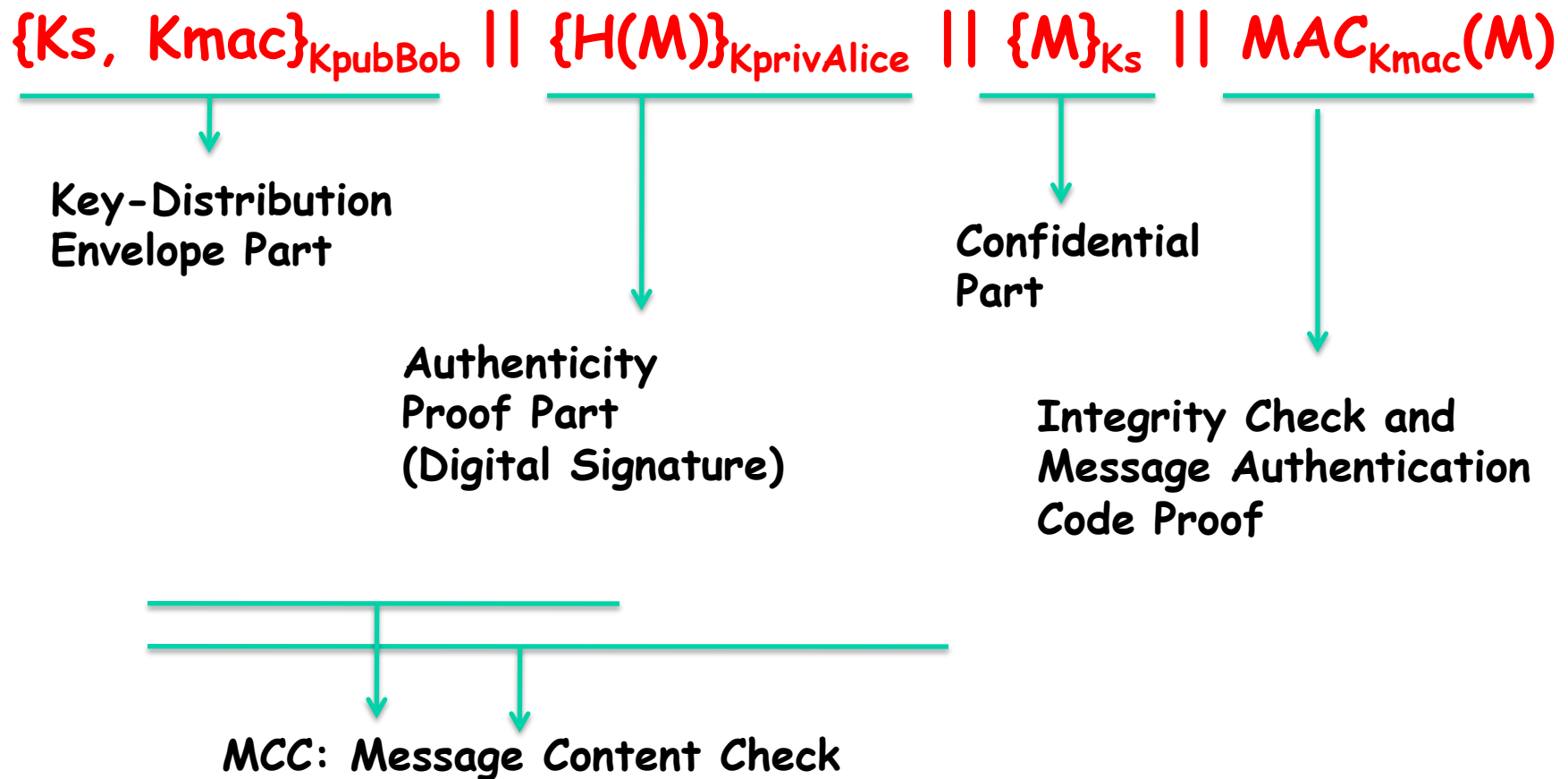
Needham Schroeder Method using Asymmetric Cryptography



Confidentiality with symmetric crypto and session key K_s (in this case generated by B)

Hybrid construction principle for secure channel establishment

Suppose Alice sends to Bob ...



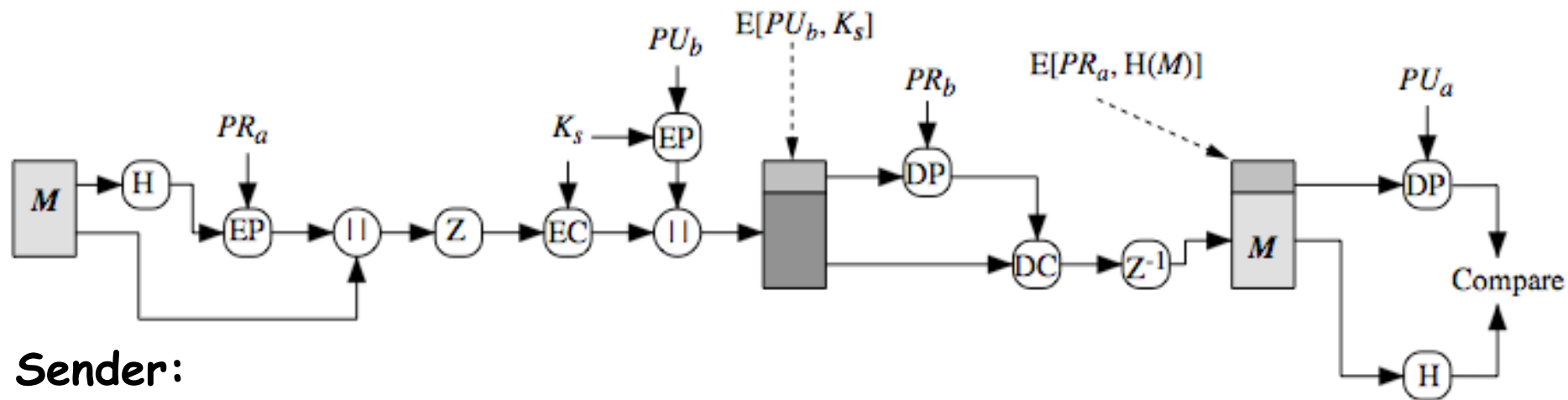
Advantages ...

- We don't need a KDC (and its trust assumptions as in the case of only using Symmetric Crypto)
 - Session Keys (generation and the quality of the key generation) is under the scrutiny of the involved principals
- Now, we only need a PKC (as a TCB)
 - The trust assumption now is that the PKC is used to distribute (correctly) the Public Keys of Principals
 - Correct mappings $\langle \text{Principal}, \text{Valid Public Key} \rangle$
 - This means that the PKC is trustable for "certifying" the **correct Public Keys** belonging to **correct principals**
 - **Better for PFS, PBS**
 - ...What if we want support Contributive Verifiable Keys ?

Hybrid use of different Crypto. Methods

Example for Secure Email:
Confidentiality + Authent. + Integrity

(Ex. Use in PGP
Pretty Good Privacy)



Sender:

$$E_{K_s} (\text{Comp} (\text{Sign}_{PR_a} (M) || M)) || E_{PU_b} (K_s)$$

Note: Compression after the signature and before encryption ! Why ?

What if there is a MiM issuing “fake” associations of public keys to correct principals ?

- Very important to obtain the Public Keys Signed by the PKC (used as a certification signing authority)
 - Can control the policy enforcements of received keys (Attributes + Policy Enforcements for Public Key Use)
- The ultimate TRUST: The Public Key of the PKC (as Public Keys of Certification Authorities - CAs)
 - Where are these public keys in your computer ?
- OK we will discuss later management of public keys and issues in the X509 Certification and Authentication Model
 - Certification Authorities and PKIs (Public Key Infrastructures)

Outline

- **Public Key Cryptography**

- Public Key Cryptography and Use of Public Key Methods
- Properties of Public Key Algorithms
- Public-Key Cryptography Algorithms
- Digital Signatures

Some Public-Key Methods

(Asymmetric Cryptographic Algorithms and Constructions)

Examples.

Note) Targeted for different uses:

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes ✓	Yes ✓	Yes ✓
Elliptic Curve	Yes ✓	Yes ✓	Yes ✓
Diffie-Hellman	No ✗	No ✗	Yes ✓
DSS	No ✗	Yes ✓	No ✗
ElGamal	Yes ✓	Yes ✓	No ✓

Properties of Public-Key Cryptography (1)

1. Computationally feasible (easy) for a principal to generate a key pair

BOB: public key: K_{pubB} ; private key: K_{privB}

ALICE: public key: K_{pubA} ; private key: K_{privA}

2. Easy for a principal (A) to generate *ciphertext* using the public-key of other receiver (B)

$$C = \{M\}_{K_{pubB}}$$

3. Easy for a principal (B) to decrypt *ciphertext* using its private key

$$M = \{C\}_{K_{privB}} = \{ \{M\}_{K_{pubB}} \}_{K_{privB}}$$

Properties of Public-Key Cryptography (2)

4. Computationally infeasible to determine private key (K_{priv}) knowing the related public key (K_{pub})
5. Computationally infeasible to recover message M , knowing K_{pub} and ciphertext C
6. Either of the two keys can be used for encryption, with the other used for decryption

$$M = \{ \{M\}_{K_{pub}} \}_{K_{priv}} = \{ \{M\}_{K_{priv}} \}_{K_{pub}}$$

What means "easy" or "unfeasibility"

- **Easy:** something solved in polynomial time as a function of input length
Input: n bits \Rightarrow function proportional to n^a , with a = **fixed constant**
Functions of class P (Prime Numbers and Properties of Prime Numbers)
- **Unfeasibility:** if the effort to compute grows faster than polynomial time
 - Prime Factorization of Big Numbers (Big Integers) + Computation of Discrete Logarithm Problem with very large exponents

Outline

- **Public Key Cryptography**

- Public Key Cryptography and Use of Public Key Methods
- Properties of Public Key Algorithms
- ▶ - Public-Key Cryptography Algorithms
- Digital Signatures

Some Public-Key Methods

(Asymmetric Cryptographic Algorithms and Constructions)

Examples.

Note) Targeted for different uses:

Algorithm	Encryption/Decryption		Digital Signature		Key Exchange	
RSA	Yes	✓	Yes	✓	Yes	✓
Elliptic Curve	Yes	✓	Yes	✓	Yes	✓
Diffie-Hellman	No	✗	No	✗	Yes	✓
DSS	No	✗	Yes	✓	No	✗
ElGamal	Yes	✓	Yes	✓	No	✓

RSA: Rivest, Shamir & Adleman, MIT, 1977 (1)

- Best known & widely used and implemented public-key scheme
 - Used as a block cipher or digital signatures
 - Digital signatures combining secure hash functions and standardized computations: ex., PKCS#N standards
 - Hybrid use with symmetric crypto: digitally signed and confidential symmetric key-envelopes, combined with symmetric encryption

RSA: Rivest, Shamir & Adleman, MIT, 1977 (2)

- Based on exponentiation in a finite (Galois) field over integers modulo a prime
 - Feasible to compute $Y = X^K \bmod N$ (knowing K , X and N)
 - Impossible (computationally) to compute X from Y , N and K
 - Exponentiation takes $O((\log n)^3)$ operations (feasible) : Discrete Logarithm Problem !
- Uses large integers (eg. **1024**, preference now 2048, 4096 bits)
- Security due to cost of factoring large numbers
 - factorization takes $O(e^{\log n \log \log n})$ operations (hard)

Security vs Practical Use (ex. RSA)

Security considerations

- **Math Attacks:**

- Factoring the product of two big primes

SP 800-131A EU Regulations for Security (Transitions: Recommendation for Transitioning of Cryptographic Algorithms and Key Lengths, 2015): **Use of 2048 bit keys for RSA**

EU Agency for Network and Information Security : Algorithms, Key Size and Parameters Report), Nov 2014): **progressive use of 3072 bit keys for RSA**

Security vs Practical Use (ex. RSA)

Security considerations

- **Timing Attacks**
 - **Inference of Key Sizes** from running time of decryption
 - **Can be masked if needed, introducing random delay**
- **Chosen Ciphertext Attacks**
 - Selection of Data Blocks to be processed by the Private Key for the purpose of cryptanalysis
 - These attacks must be avoided using **Padding**
 - **Avoidance of the “low exponentiation problems”**: use of large keys

Math of RSA

- Number theory
 - Prime numbers, prime factorization
 - Relatively primes and its properties:
 - Ex., GCD
 - Fermat theorem
 - Euler theorem and Euler Totient Function $\phi(n)$
 - Primality testing
 - Miller-Rabin algorithm and prime distribution or estimation
 - CRT (Chinese Remainder Theorem)
 - Modular arithmetic
 - Primitive roots of integers and primes
 - Discrete logarithms (inverse of exponentiation)
 - Find i , such that $b = a^i \pmod{p}$,
or $i = \text{dlog}_a b \pmod{p}$

DH, El Gamal, DSS (or DSA)

- **Diffie-Hellman**
 - Exchange a secret key securely (secret key establishment) or key-agreement
 - Unfeasible solution of discrete logarithms (computational time and complexity)
- **El Gamal**
 - Block Cipher
 - Unfeasible solution of discrete logarithms (computational time and complexity, comparable to RSA)
- **Digital Signature Standard (DSS or DSA)**
 - Initially Make use of the SHA-1 (now, usage of other Hash functions (SHA-2 and SHA-3))
 - For digital signatures (only), not for encryption or key exchange
 - Also implementable with different asymmetric algorithms

Elliptic Curve Cryptography

- **Elliptic-Curve Cryptography (ECC)**
 - Good for smaller bit size
 - Low confidence level yet, compared with RSA
 - A Recent (in going) Story of Weak vs. Strong Curves
 - Complexity, Reputation is growing ... Interest in Mobile Computing and Constrained-Devices

Why ?

- Majority of public-key crypto (RSA, D-H and others) use either integer or polynomial arithmetic (intractability of math problems): very large numbers/polynomials, factorization problem of big numbers composed of two or more large prime factors
 - **Imposes a significant load in storing and processing keys and messages**

Elliptic Curve Cryptography

- **Elliptic-Curve Cryptography (ECC)**
 - Good for smaller bit size
 - Low confidence level yet, compared with RSA
 - A Recent (in going) Story of Weak vs. Strong Curves
 - Complexity, Reputation is growing ... Interest in Mobile Computing and Constrained-Devices
- ECC appears as an alternative for offering same security with smaller bit sizes
- Newer, not as well (crypt)analyzed // Ongoing Research
- Standardization problem: different ECC curves and their characteristics

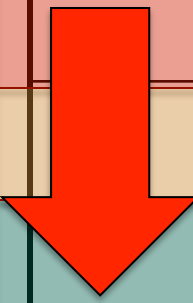
Other Public-Key Algorithms ...

- Many other public-key algorithms:
Knapsack, Pohlig-Hellman, Rabin, McEliece, LUC, Finite Automaton
- Many other public-Key signature algorithms and constructions:
DSA variants, GOST, Discrete Logarithm Variants, Ong-Schnorr-Shamir, ESIGN, ...
- See the Algorithms available in the JCE Framework for Programming
- Foundation (for specific interest):
Bruce Schneier, Applied Cryptography, Wiley, 2006

Comparable Key Sizes for Equivalent Security: RSA vs. ECC vs. Symmetric Encryption

Computational effort for cryptanalysis

Symmetric scheme (key size in bits)	ECC-based scheme (size of n in bits)	RSA, DSA (modulus size in bits)
56	112	512
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360



The RSA Cryptosystem

RSA Internals and Structure

The RSA Algorithm - Key Generation

Key pair generation (summary and simple example)

1. Select p, q p and q both prime (secrets)
2. Calculate $n = p \times q$
3. Calculate $\Phi(n) = (p - 1)(q - 1)$
4. Select integer e $\gcd(\Phi(n), e) = 1; 1 < e < \Phi(n)$
5. Calculate d $d = e^{-1} \bmod \Phi(n)$
6. Public Key $K_{\text{pub}} = \{e, n\}$
7. Private key $K_{\text{priv}} = \{d, n\}$

1) Ex., 7, 17 2) $n = 7 \times 17 = 119$ 3) $\phi(n) = 6 \times 16 = 96$

4) $e = 5$, $\gcd(96, 5) = 1$, com $1 < 5 < 96$

5) $5 \times d = 1 \bmod 96$, com $d < 96$ $d = 77$
 $5 \times 77 = 385$, notar que $4 \times 96 + 1 = 385$

$K_{\text{pub}} = (5, 119)$

$K_{\text{priv}} = (77, 119)$

The RSA Algorithm: Encryption/Decryption

Encryption: $C = \{P\}_{K_{pub}}$

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

Decryption: $P = \{C\}_{K_{priv}}$

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

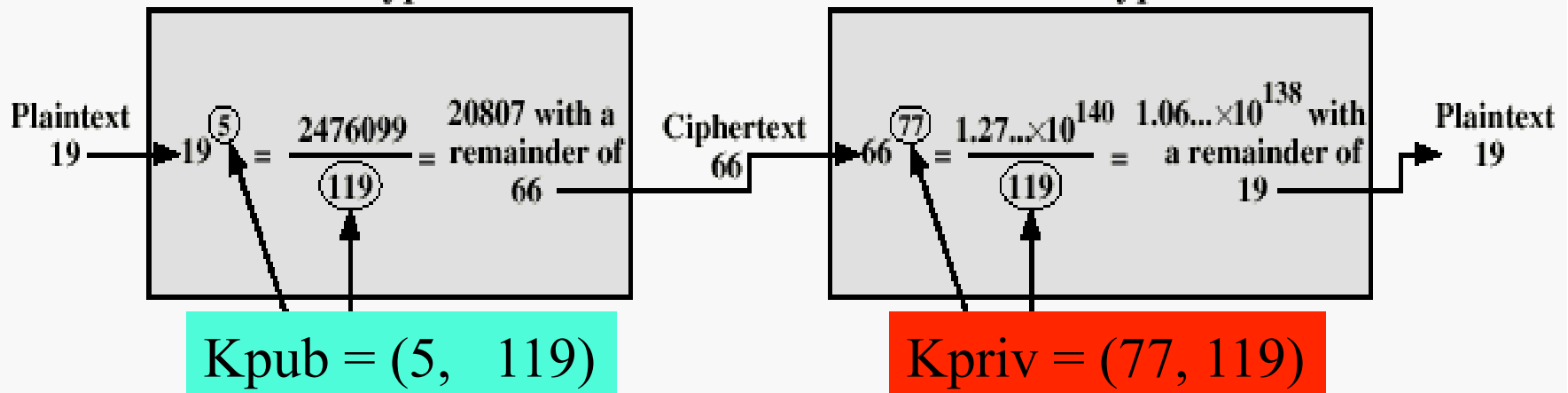
Ex., $M = 19$

Encryption

$C = 66$

Decryption

$M = 19$



Another RSA Simple Example - Key Setup

1. Select primes: $p=17$ & $q=11$ (secrets)
2. Compute $n = pq = 17 \times 11 = 187$
3. Compute $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$
4. Select e : $\gcd(e, 160) = 1$; choose $e=7$
5. Determine d : $de = 1 \pmod{160}$
and $d < 160$ Value is $d=23$ since $23 \times 7 = 161 = 10 \times 160 + 1$

Publish public key $K_{\text{pub}} = \{7, 187\}$

Keep secret private key $K_{\text{priv}} = \{23, 187\}$

Another RSA Example - Encrypt/Decrypt

Given message $M = 88$ (note that $88 < 187$, then it is ok)

- Encryption:

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

- Decryption:

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

Other sources to learn about RSA

- Summary of Math behind (see also additional slides in this presentation)
- Other sources: wikipedia article:
[https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem)), is ok
- Math background and practical issues
- Relevance of Padding and attacks against plain RSA (without padding):
 - Low encryption exponents e
 - Small values for plaintext values M ($M < N^{1/e}$)
 - **Causes: that m^e is strictly smaller than modulus N**
 - Problems of sharing similar exponents, using the CRT (The Coopersmith Attack)
 - Exploiting the deterministic nature of encryption (non semantically security)
 - Exploiting the multiplication homomorphism of the RSA encryption

Suggested Readings



Suggested Readings:

W. Stallings, Network Security Essentials - Applications and Standards, Chap 3., sections 3.4 to 3.6

The RSA Cryptosystem

Programming with the RSA Algorithm
Encryption/Decryption
Secure Key Envelopes
Digital RSA Signatures

Encryption/Decryption using Public Key Algorithms

- **See more (hands-on) in Labs** (Use of Public Key Algorithms for Encryption Decryption in Java JCE)

Key pair: $\langle K_{pub}, K_{priv} \rangle$

$$C = \{ M \}_{K_{pub}}$$

$$P = \{ C \}_{K_{priv}}$$

With no
Padding

- **Use of *Standardized Padding Methods* (ex., RSA-PKCS#1, RSA-OAEP, RFC 5756) for secure use in encryption/decryption**

Key pair: $\langle K_{pub}, K_{priv} \rangle$

$$C = \{ \text{Padding} || M \}_{K_{pub}}$$

$$P = \{ C \}_{K_{priv}}$$

With
Padding

Relevance of Padding : ex., PKCS#1 for RSA

- Form of structured, randomized values, added to plaintext M (on the left) before encryption assuring that:
 - The M value (as an integer) does not fall into the range of insecure plaintexts
 - M , once padded, will encrypt to one of a larger number of different possible *ciphertext* numbers !

Possible: PKCS#1 (RSA Security inc., Recommendation/Standard):

- But (up to version 1.5) is not recommended today as a way to add high enough level of security, should be replaced wherever possible
- PKCS#1 - also incorporates processing schemes for additional security in RSA-based digital signatures (to see later)
 - Called PKCS#1 PSS (Probabilistic Signature Scheme)
 - ... Some other PSS based schemes w/ patents expired in 2009 and 2019

RSA PKCS#1 (v1.5)

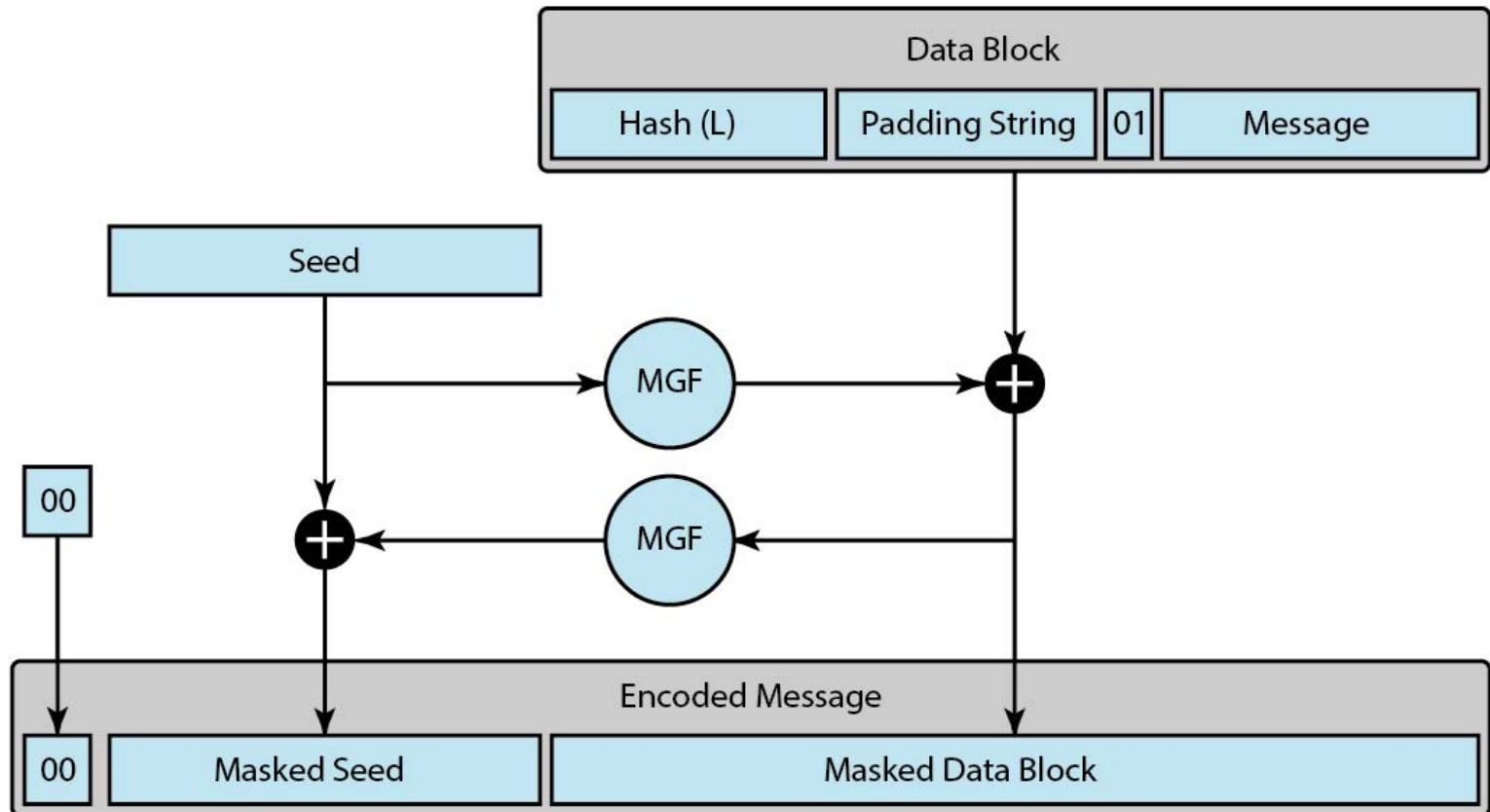
PKCS #1 (v1.5): Padding Formats and Usage

- ♦ Sign: $01 \parallel \text{ff} \dots \text{ff} \parallel 00 \parallel \text{DER}(\text{HashAlgID}, \text{Hash}(M))$
- ♦ Encrypt: $02 \parallel \text{pseudorandom PS} \parallel 00 \parallel M$
- ♦ Ad hoc design
- ♦ Widely deployed, incorporated in many Internet standards, such as:
 - PKIX profile
 - TLS
 - IPSEC
 - S/MIME

Example: RSA-OAEP

Optimal Asymmetric Encryption Padding

- Published at Eurocrypt 2000 (Coron et al.,)
Crypto 1998 (Bleichenbacher et al)



RSA Digital Signatures

See more (hands-on) in Labs (Use of Public Key Algorithms for Encryption Decryption in Java JCE)

Use of *Standardized Padding Methods* for secure Digital Signatures

Key pair: $\langle K_{\text{pub}}, K_{\text{priv}} \rangle$

$$\text{Sig}(M) = \{ H(\text{Padding} || M) \}_{K_{\text{priv}}}$$

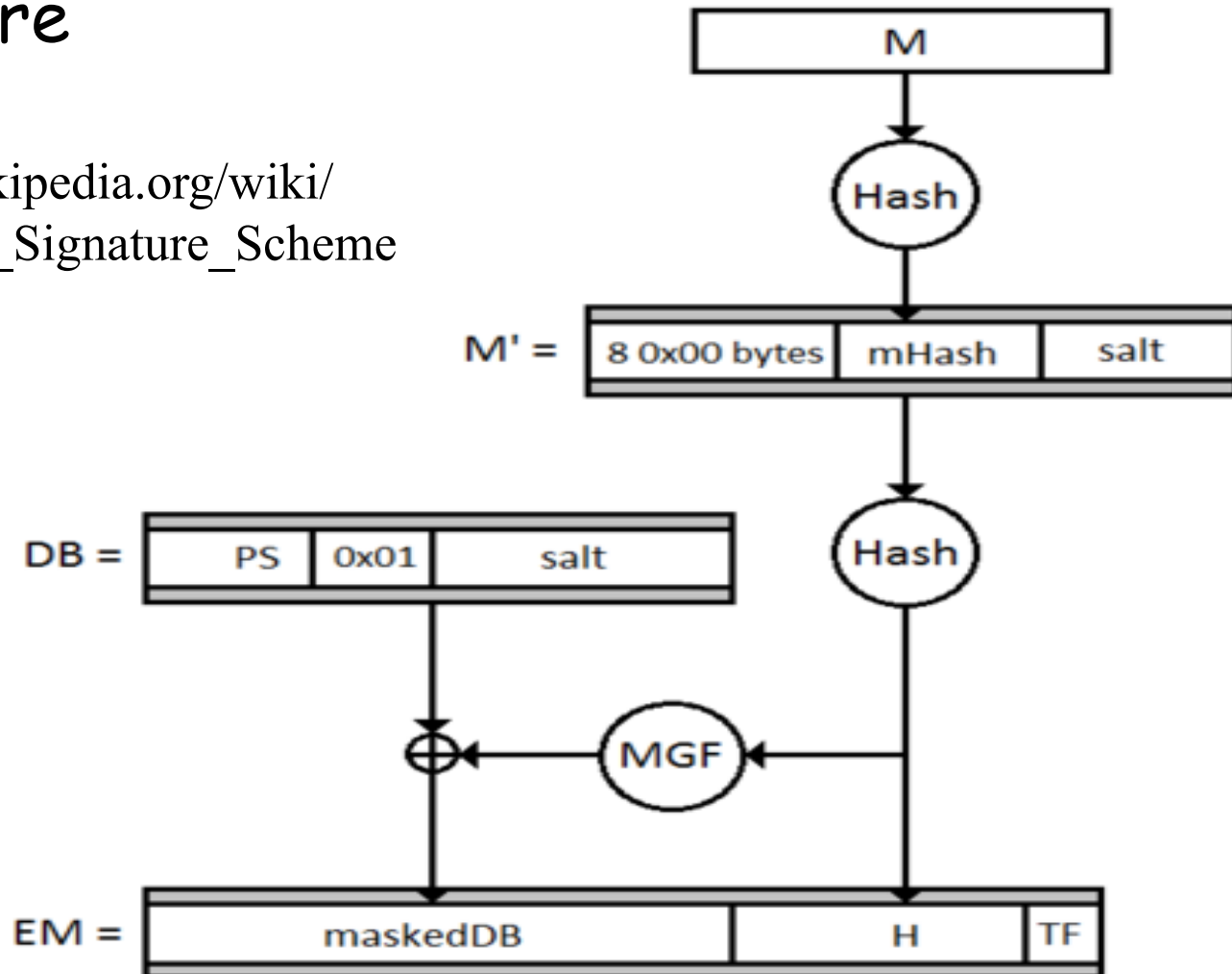
Ex: RSA-PKCS#1, RSA-PSS

RSA PSS (aka PKCS#1 v2, RFC 5756)

Signature

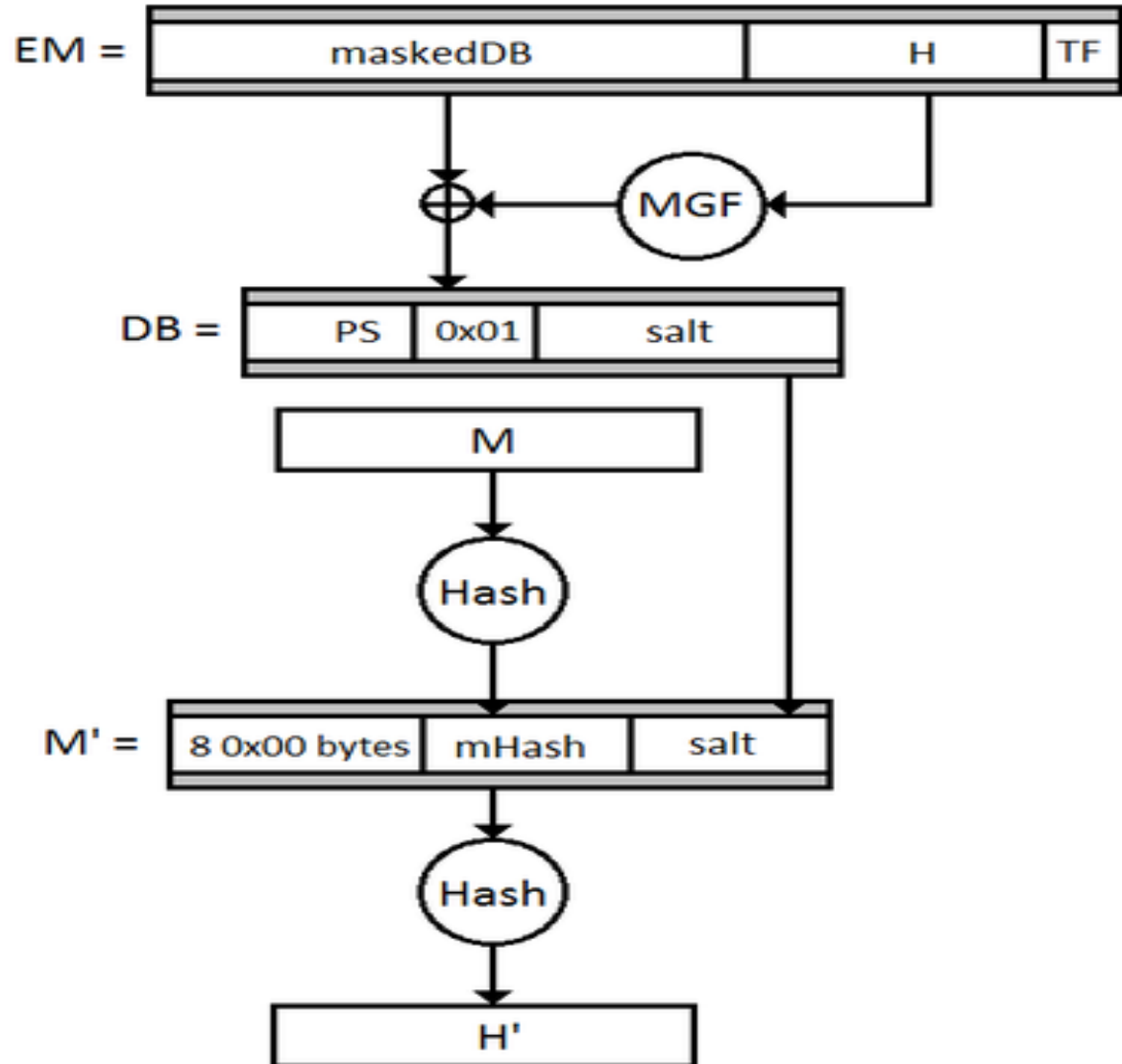
See, ex:

https://de.wikipedia.org/wiki/Probabilistic_Signature_Scheme



RSA PSS (aka PKCS#1 v2, RFC 5756)

Signature Verification



See more (hands-on) in LABs (Java, JCE)

- RSA Enc/Dec w/ PKCS#1 and OAEP
 - PKCS#1, PSS Padded Digital Signatures w/ RSA
 - ECCDSA Digital Signatures
-
- See also the Public Key Crypto Alg. And Digital Signatures' standardized constructions in the JAVA/JCE framework for programming

The RSA Cryptosystem

ElGamal Algorithm

ElGamal (Taher ElGamal, 1985)

Security only based on the complexity of the computation of modular logarithms (using big numbers)

It is an *homophonic* cipher

Summary:

Public values: p // big prime number

g // primitive root of p (or \mathbb{Z}_p)

Private Key: x // $0 \leq x \leq p$

Public Key: y // $y = g^x \bmod p$

To Encrypt: Generate a random value $k \leq p-1$

$$c1 = g^k \bmod p; \quad c2 = P \cdot y^k \bmod p$$

To Decrypt: $P = c2 \cdot (c1^x)^{-1} \bmod p, \quad c1^x \cdot (c1^x)^{-1} \equiv 1 \pmod{p}$

DSA (or DSS)

DSA - Digital Signature Algorithm
(or DSS - Digital Signature Standard)

DSA

DSA, (Aug/1991) : Digital Signature Standard promoted by NIST under the designation: DSS - Digital Signature Standard (Standard FIPS 186-3, June 2009, 186-4 rev 2013)

(A variant of Schnorr and El Gamal Crypto. but specifically targeted for digital signatures only : similar to El Gamal Signatures)

Ref:

https://en.wikipedia.org/wiki/Digital_Signature_Algorithm

<https://csrc.nist.gov/publications/detail/fips/186/4/final>

DSA Parameterizations

$H(\)$: Secure hash function

- SHA 1, SHA 2 initially promoted in the standardization of DSS signature constructions

Two prime numbers: p (L bits) and q (N bits):

$p-1$ must be multiple of q

Must choose g , such that $g^q = 1 \pmod p$

Shared Parameters: p , q and g

DSA Keys

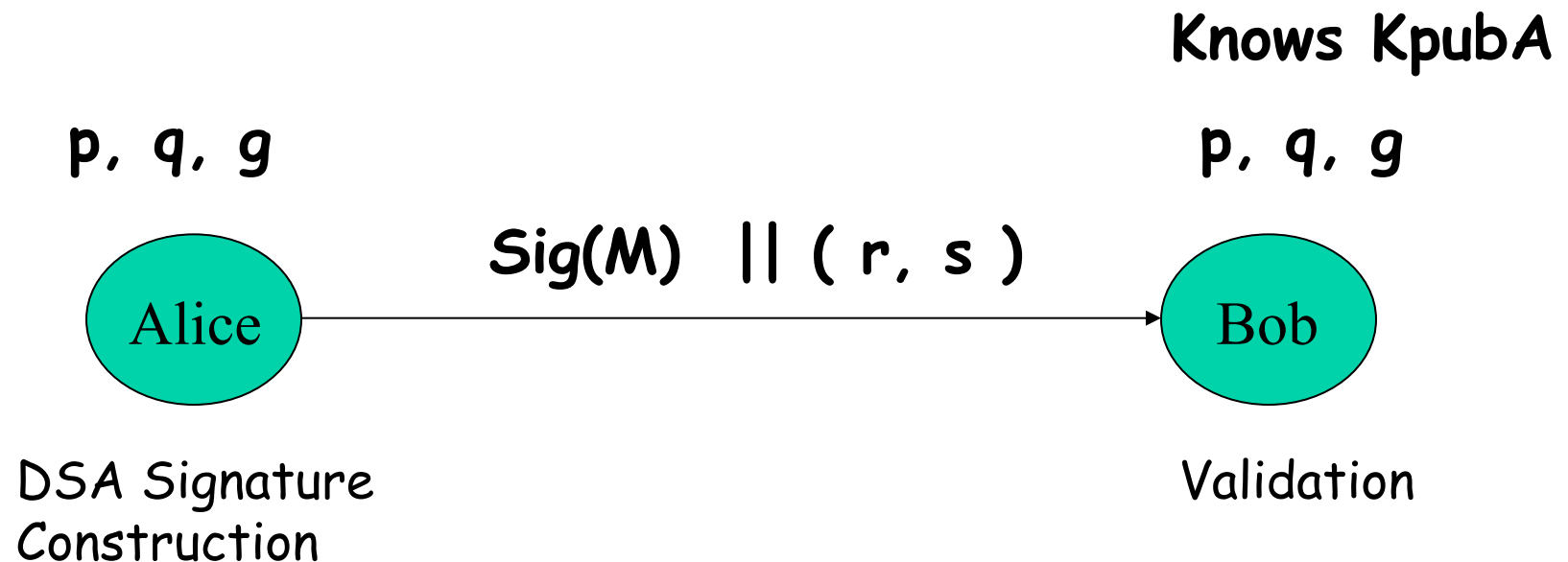
Key pair (Kpriv, Kpub) in group (primitive root) g

- Kpriv, chosen as a secret random: x
in such a way that $1 < Kpriv < q$
- Kpub, chosen as: $Kpub = y = g^{Kpriv} \bmod p$

A Key pair is (x, y) given (p, q, g)

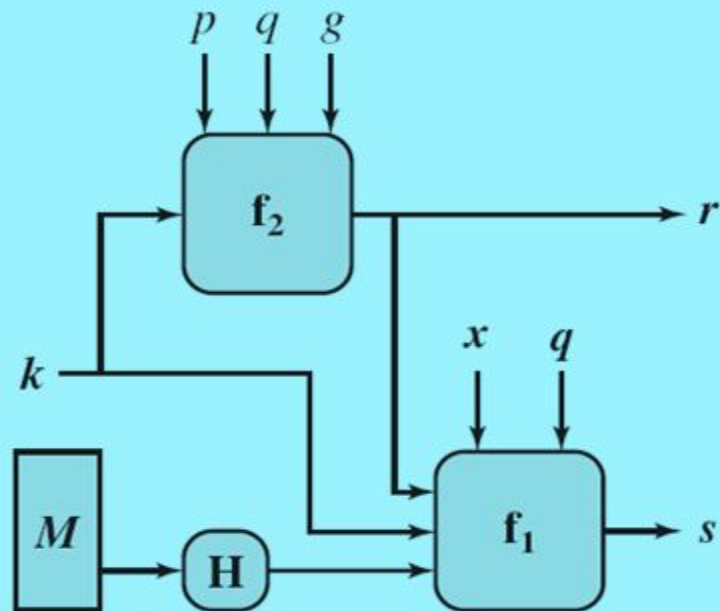
DSA: Shared Parameters and Signatures

Shared Parameters: p, q, g



Digital Signatures

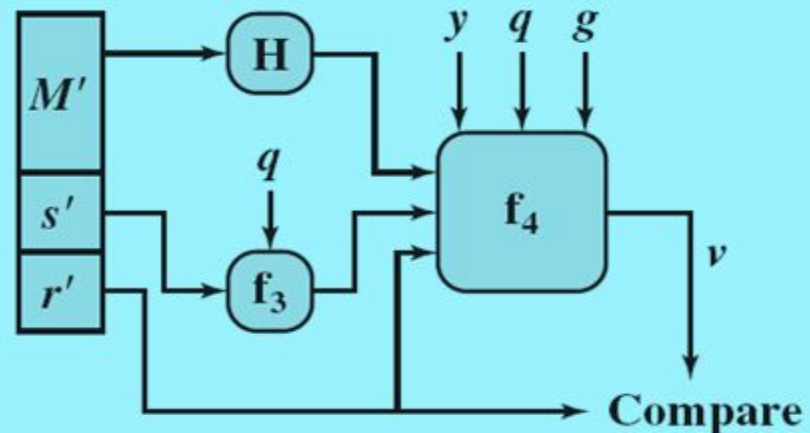
Digital Signature Algorithm (DSA)



$$s = f_1(H(M), k, x, r, q) = (k^{-1} (H(M) + xr)) \bmod q$$

$$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$$

(a) Signing



$$w = f_3(s', q) = (s')^{-1} \bmod q$$

$$v = f_4(y, q, g, H(M'), w, r') \\ = ((g^{H(M')w} \bmod q) y^{r'w} \bmod q) \bmod p) \bmod q$$

(b) Verifying

DSA Signature Construction

1. Generate a random per-message value k , with $1 < k < q$
2. Compute $r = (g^k \bmod p) \bmod q$
if $r=0$, regenerate the random k
3. Compute $s = k^{-1} (H(M) + xr) \bmod q$
if $s=0$, regenerate the random k
4. If $s \neq 0 \Rightarrow$ the $Sig(M) = (r, s)$

So we can send M , r, s

↓
 $Sig(M)$

DSA Signature Verification

Given M and (r, s)

1. We must reject a signature

if $0 < r < q$ or $0 < s < q$

2. Compute $w = s^{-1} \bmod q$

3. Compute $u1 = H(M) \cdot w \bmod q$

4. Compute $u2 = r \cdot w \bmod q$

5. Compute $v = (g^{u1} g^{u2} \bmod p) \bmod q$

6. If $v = r$ the signature is valid

DSA: Practical Observations

- DSA Signatures tend to be slowly compared with RSA (for keys with the same size)
- But sizes of signatures are shorter:
(you can see practical evidences in Lab exercises)
 - In RSA, proportional to the key sizes and related modulo N
(See the RSA algorithm)
 - In DSA, depending on the parameters, can appear usually with 40 bytes but the standard representation (ASN.1) expands the signature to 44 - 48 bytes, plus 3 bytes for bitstring encoding. So you can expect: 47 to 51 bytes

DSA Security Conditions

Decide on a key length L and N . This is the primary measure of the cryptographic strength of the used key

The original DSS constrained L to be a multiple of 64, between 512 and 1,024 (inclusive).

NIST 800-57 recommends lengths of 2048 (or 3072) for keys with security lifetimes extending beyond 2010 (or 2030), using correspondingly longer N

FIPS 186-3 specifies L and N length pairs of (1024, 160), (2048, 224), (2048, 256), and (3072, 256). N must be less than or equal to the output length of the used hash H .

Elliptic Curve Cryptography

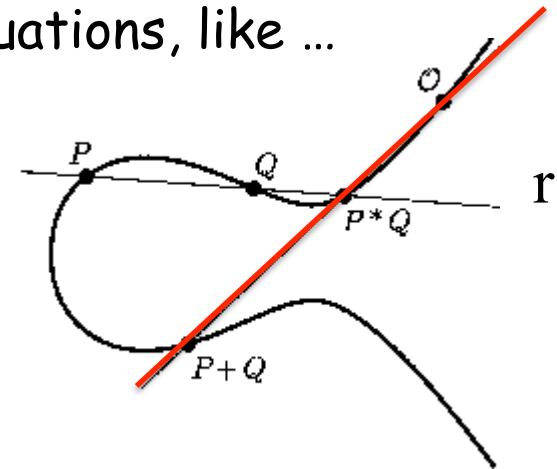
ECC - Elliptic Curve Cryptography

ECC Foundations

- For current cryptographic purposes, an *elliptic curve* is a plane-curve over a finite field (rather than the real numbers) which consists of the points satisfying for equations, like ...

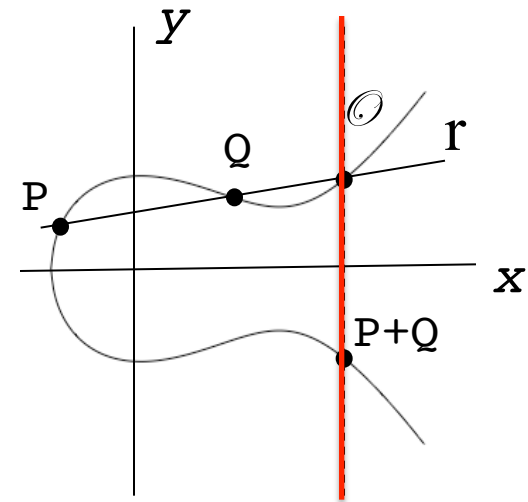
Ex1: $y^2 = x^3 + ax + b$

and a given point at infinity \mathcal{O}



Ex2: $y^2 = x^3 - x + 1$

and a given point at infinity \mathcal{O}



Curve and \mathcal{O} , together with the group operation of elliptic curves form a so-called Abelian Group, where the point at infinity is the identity element. The structure of the group is inherited from the divisor group of the underlying algebraic variety

ECC

- It is assumed that finding the discrete logarithm of a random elliptic curve element with respect to a publicly known base point (or generation point) is infeasible:
 - this is known as the "**elliptic curve discrete logarithm problem**" (ECDLP).
- The use and security of elliptic curve cryptography depends on:
 - the ability to compute POINT MULTIPLICATION
 - the inability to compute the multiplicand given the original and product points.
 - The "size" of the elliptic curve (**ECC Order**) determines the difficulty of the problem.

In general:

Elliptic Curves: Addition + Duplication

Ex., considering point in infinite \mathcal{O} and curve:

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

Addition $C = A+B$

- Interception point P of r with the curve, and obtain the symmetric (on axe x) of P

Duplication $C = 2A$

- The sum of a point with the same point
 - Similar to addition, but now r is the tangent to the considered point

Multiplication of any number d by a Point P in a curve:

$$dP = 2^3P + 2^2P + 2^1P = 2(2(2P)) + 2(2P) + 2P \text{ (ex., if } d=14\text{)}$$

- Decomposition of d using 2^i factors:
- We have 3 duplications and 3 sums in this case

Elliptic Curves as Discrete and Modular Universes

- ECC Points only have integer coordinates
- Modulo q : coordinates of Points must belong to a set $[0, q-1]$
- q value is variable ... Can be P^K
 - P a prime (Prime corpus F_P)
 - K an integer constant (can be 2^m) with m prime (Binary Corpus F_{2^m})

ECDLP or Discrete Logarithm Problem in ECCs

- Given a curve and a point G of the curve (*called base or generator point*), and give a point P in the curve in such a way that P is multiple of G i.e., $\Rightarrow P = dG$
- To compute the discrete logarithm of P given G , we must find the value of d
 - d is a "private key" // an integer value
 - P is a "public key" // i.e., a point in a curve (x,y)

Why we can not have d knowing P ?

- The problem is the number of points in the curve. The number of points is considered "the order of the curve"
- Many points \Rightarrow More Security !

ECC Order (or #E ... or number of ECC points)

- An ECC can have several finite groups of points. Let's take n as the dimension of such groups

$$n P = \mathcal{O}$$

$$\text{Co-Factor } h = (\#E / n)$$

$$h = 1 \Rightarrow \#E = n$$

When we choose an ECC curve for cryptography, we must choose a **Generator Point G** (see last slide), in such a way that:

- **G must belong to the group with the big n in the curve**
- Then.... The value of n determines the key size of an ECC

Practical use of ECC: ECDH and ECDSA

- In practice ECCs are not used for encryption and decryption ! The use is for:
 - **Diffie-Hellman Key Agreements (called ECDH)**
 - DH computation with ECCs
 - Will see later (after explaining the DH Methods for Key-Agreements)
 - **And also for digital signatures (ex., ECDSA)**
 - DSA Signatures computed with ECC
 - See also DSA (or DSS)

Choice of Elliptic Curves

- Complex task - involves the choice of an **ECC curve**, and the value for the **Generator Point P**
 - Maximize $n \Rightarrow$ High security but lower efficiency
- In general we use pre-studied and pre-established curves
 - Standardized and promoted by different organizations or published in scientific cryptography venues)
 - Ex:

NIST Standardization

IETF Standardization

NIST Defined Secure ECCs (*)

NIST Promoted Secure ECCs:

- Ten Promoted Pseudo-Random Curves: 5 types with different dimensions of the modulo q and 2 types of corpus: prime and binary corpus
- Names from NIST relate to the number of bits of the divisor (modular corpus):

Curves in Prime Corpus

P-192, P-224, P-256, P-384, P-521

$$y^2 = x^3 - 3x + b, \text{ co-factor } h = 1$$

Curves in binary Corpus

B-163, B-233, B-283, B-409, B-571

$$y^2 + xy + x^3 + x^2 + b, \text{ co-factor } h = 2$$

The value b corresponds to the hash of an input "random" value to avoid a choice for b that may introduce vulnerabilities

Some on-gong concerns on ECC security came from Edward Snowden revelations on " ... possible intentional vulnerabilities introduced by NSA in a pseudo-random generator based on Elliptic Curves"

NIST Defined Efficient ECCs

NIST Efficient ECCs (or Koblitz Curves)

Curves with Binary Corpus:

K-163, K-233, K-283, K-409, K-571

$$y^2 + xy = x^3 + ax^2 + 1$$

Coefficient $a = 1$ (for co-factor $h = 2$) or $a = 0$ (for cofactor $h = 4$)

Elliptic Curve Cryptography

(and ongoing standardization and concerns)

- The U.S. NIST (National Institute for Standards and Technology) has endorsed elliptic curve cryptography (Standard Suite B set of recommended algorithms, specifically ECDH for key exchange and ECDSA for digital signature).
- The U.S. NSA (National Security Agency) allows their use for protecting information classified up to "top-secret" with 384-bit keys.
- However, in August 2015, the NSA announced that it plans to replace Suite B in the future with a new cipher suite, due to concerns about quantum computing attacks on ECC.

IETF ECCs: Curve25519 and Curve448

Curve25519 - Initially proposed by Daniel Bernstein

$$y^2 = x^3 + 486662 x^2 + x$$

$$\text{Modulo} = 2^{255} - 19$$

$$\text{Generator } Gx = 9$$

$$\text{Order} > 2^{255}, \text{ co-factor } h=8$$

- In the conducted studies this curve has an high-ranking (compared with the NIST proposed Curves) in efficiency (computation time) and memory occupation

Curve448 - also recommended by the IETF

$$y^2 = x^3 + 156326 x^2 + x$$

$$\text{Modulo} = 2^{448} - 2^{224} - 1$$

$$\text{Generator } Gx = 5$$

$$\text{Order} \sim 2^{446}, \text{ co-factor } h=4$$

Hands-On with ECC

- See the LAB Materials
- Programming with ECCs in Java

The RSA Cryptosystem

ECDSA - Digital Signature Algorithm
Using ECCs

ECDSA: Signature

- Given an ECC $E(n, G)$ // order n , Generator G
- Private Key: x , with $0 < x < n$
- Public Key: y , with $y = x G$

Signature generation:

1. Choose a random k , with $1 \leq k \leq n-1$
2. Compute the multiplicative inverse of k : $(k \cdot k^{-1}) \bmod n = 1$
3. Compute $K = k G$
4. Compute $r = K_x \bmod n$ $(r > 0, r \leq n)$
5. Compute $s = k^{-1} (\text{SHA-1}(M) + r \cdot x) \bmod n$ $(s \leq n)$
6. Send M , r and s (as signature of M)

ECDSA: Signature

- Given an ECC $E(n, G)$ // order n , Generator G
- Private Key: x , with $0 < x < n$
- Public Key: y , with $y = xG$

Signature validation:

1. Confirm that $0 < r \leq n$ and $s \leq n$
2. Confirm that y is a point in the given ECC, and $(n, y) \neq \mathcal{O}$
3. Compute $w = (s^{-1}) \bmod n$
4. Compute $u_1 = w \cdot \text{SHA-1}(M) \bmod n$
5. Compute $u_2 = w \cdot r \bmod n$
6. $P = u_1 \cdot G + u_2 \cdot Y$
 P different than \mathcal{O} , $r \equiv P_x \bmod n$

The RSA Cryptosystem

Blind Signatures

Blind Signatures

- Specific applicability: signatures preserving the anonymity (ex., digital cash, electronic voting)
- Idea: Alice first pre-process data with obscurity (transforming it in an apparently random value, not related to the original), signs this value and sends to a requested signer

$$m = H(DOC), \quad m' = F_{obsc}(m)$$

- Requested signer signs the obscurity value and return the signature to Alice

$$Ax(m') = E_{kx}^{-1}(m')$$

- Alice sends the obtained signature to be validated by others

$$Ax(m) = F_{obsc}^{-1}(Ax(m'))$$

Chaum Blind Signatures

- Alice first pre-process data with obscurity (transforming it in an apparently random value, not related to the original), signs this value and sends to a requested signer

Computes a *random* k , and $m' = (k^e \cdot m) \bmod n$

- Requested signer signs the obscurity value and return the signature to Alice

$$Ax(m') = (m')^d \bmod n$$

- Alice sends the obtained signature to be validated by others

$$Ax(m) = k^{-1} \cdot Ax(m') \bmod n$$

Demo: $k^{-1} (k^e \cdot m)^d \bmod n = (k^{-1} k^{ed} \cdot m^d) \bmod n = (k^{-1} k m^d) \bmod n = m^d \bmod n$

The RSA Cryptosystem

Diffie-Hellman

Diffie-Hellman Key Exchange

- First public-key type scheme:
 - Diffie & Hellman in 1976 along with the exposition of public key concepts
 - note: now know that Williamson (UK CESG) secretly proposed the concept in 1970
- Practical method for public exchange of a secret key k between A and B
 - Never exposing k
 - Without any previous shared secret between A and B
 - Use for shared secret key-establishment without any previous shared secret
 - Ideal support for PFS and PBS warranties
- Used in many security standard protocols and in a several commercial products

Diffie-Hellman Key Exchange

- D-H is a public-key scheme for use as a key (or secret) distribution scheme
 - cannot be used to exchange an arbitrary message (not an encryption method)
 - rather it can establish a common key, known only to the two participants
 - The common established key can be used as a shared and contributive secret key or shared key-material/seed to generate a key session
- Value of key depends on and only the participants (and their private and public D-H parameters)
 - D-H Private and public Numbers + Initial (non-secret) setup parameters

Diffie-Hellman Method and Math Behind

- Based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) - easy to compute
- Security relies on the difficulty of computing discrete logarithms (similar to factoring), hard to compute (or computationally unfeasible)

Diffie-Hellman: foundations (1)

- Global parameters:
 - q : a large prime integer
 - a : a primitive root mod q
 - In modular arithmetic, a primitive root **mod** q is any number a that:
 - Any number b (integer) relatively prime to q is congruent to a power $a^i \bmod q$
 - a is called the generator of a multiplicative group of integers modulo q
 - $a^i \bmod q$, where $0 \leq i \leq (q-1)$ generates all the integers between 1 and $q-1$, in some permutation order
 - For any integer $b < q$ with, there is a unique exponent integer i such that $b = a^i \bmod q$
- Such i is called the *index* or *the discrete logarithm of b for the base $a \pmod{q}$*

$$i = d_{\log a, q}(b)$$

Diffie-Hellman: foundations (2)

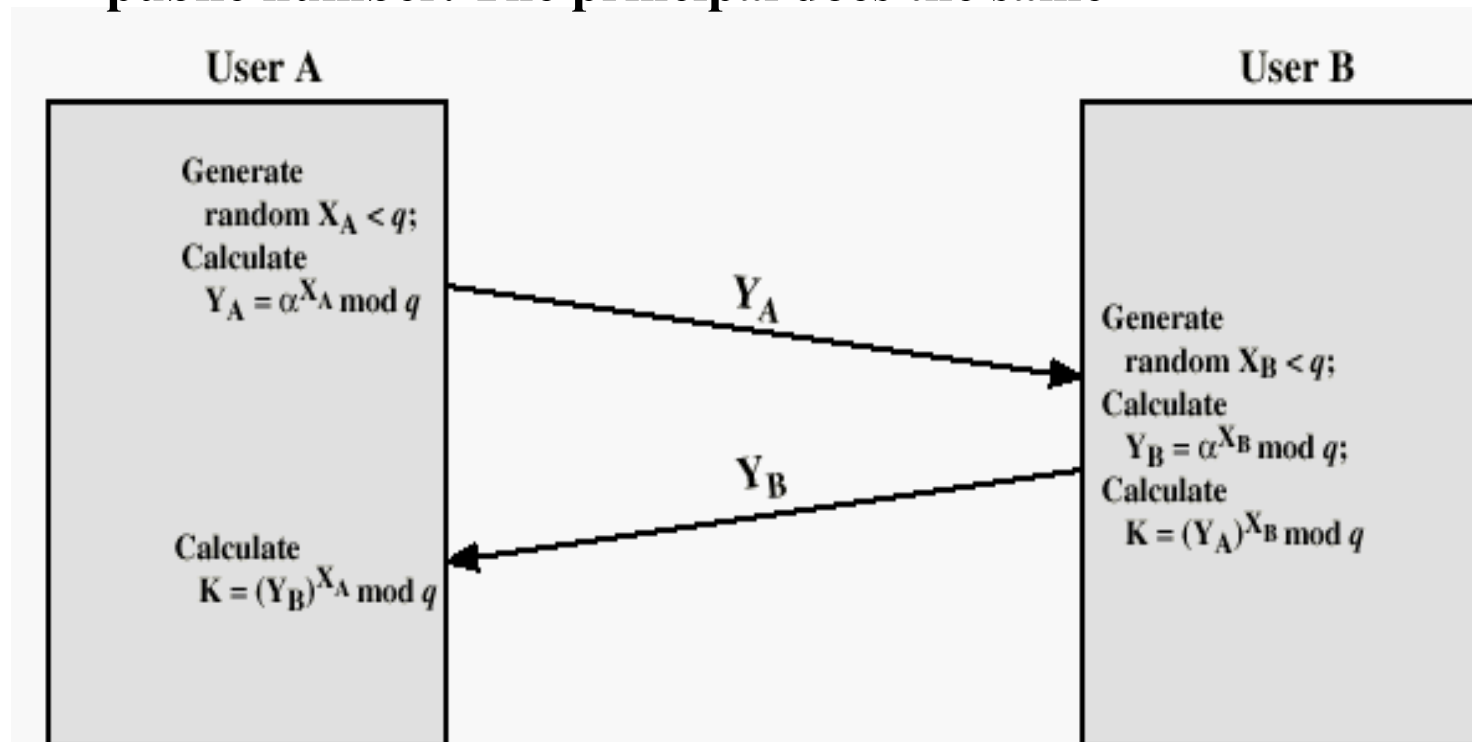
- Considering i the discrete logarithm for which:
 $a^i \bmod q = b$, taking a and q (as initial known parameters)
 - It must be simple to calculate b , knowing i
 - It must be very hard to calculate i only knowing b , a and q
 - This implies the computation of the discrete logarithm: no efficient solution (computational impossibility)
 - Hard as polynomial complexity
 - Linear to a , computational complexity equivalent to a^i

- From modular arithmetic properties for a , p and any value $i=R$:

$$\begin{aligned} a^R \bmod q &= a^{R1 \cdot R2} \bmod q \\ &= (a^{R1} \bmod q) (a^{R2} \bmod q) \\ &= (a^{R1} \bmod q)^{R2} \bmod q \end{aligned}$$

Diffie-Hellman Setup and agreement

- If A and B share the global parameters a and q , being a a primitive root modulo q
- Each user (eg. A, B) generates (private,public) pair
 - selects a random **private secret number**: $x < q$
 - Principal A computes: $y_A = a^{x_A} \bmod q$ and makes public y_A as a **public number**. The principal does the same



Diffie-Hellman Key Exchange

- Shared session key for users A & B is K_{AB} :
$$K_{AB} = a^{x_A \cdot x_B} \bmod q$$
$$= y_A^{x_B} \bmod q \quad (\text{which } \mathbf{B} \text{ can compute})$$
$$= y_B^{x_A} \bmod q \quad (\text{which } \mathbf{A} \text{ can compute})$$
- K_{AB} is used as session key in secret-key sharing 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-numbers for new D-H agreement
 - Successive D-H agreements for rekeying of K_{AB}
 - PFS and PBS conditions warranted
- Note) It is possible to make generalized D-H agreements, extended to a group of N

Diffie-Hellman Example

- Users Alice & Bob who wish to swap keys:
- Ex., agree on prime $q=353$ and $a=3$
- Select random secret numbers:
 - A chooses $x_A=97$, B chooses $x_B=233$
- Compute respective the public numbers:
 - $y_A = 3^{97} \bmod 353 = 40$ (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)
- PFS and PBS, without knowing the private numbers (never exposed) and without any previous shared secret or long-time duration secrets

Diffie-Hellman Key Exchange (example)

$q = 353, a=3$

Alice

Generate
random

$X_a=97 < 353$

$$Y_A = 3^{97} \bmod 353 = 40$$

$$\begin{aligned} Y_B^{X_A} \bmod 353 \\ &= 248^{97} \bmod 353 \\ &= 160 \\ K_{ab} &= 160 \end{aligned}$$

Shared Values

$q = 353, a=3$

$$Y_A = 40$$

$$Y_B = 248$$

Mallory

$q = 353, a=3$

Bob

Generate
random

$X_b=233 < 353$

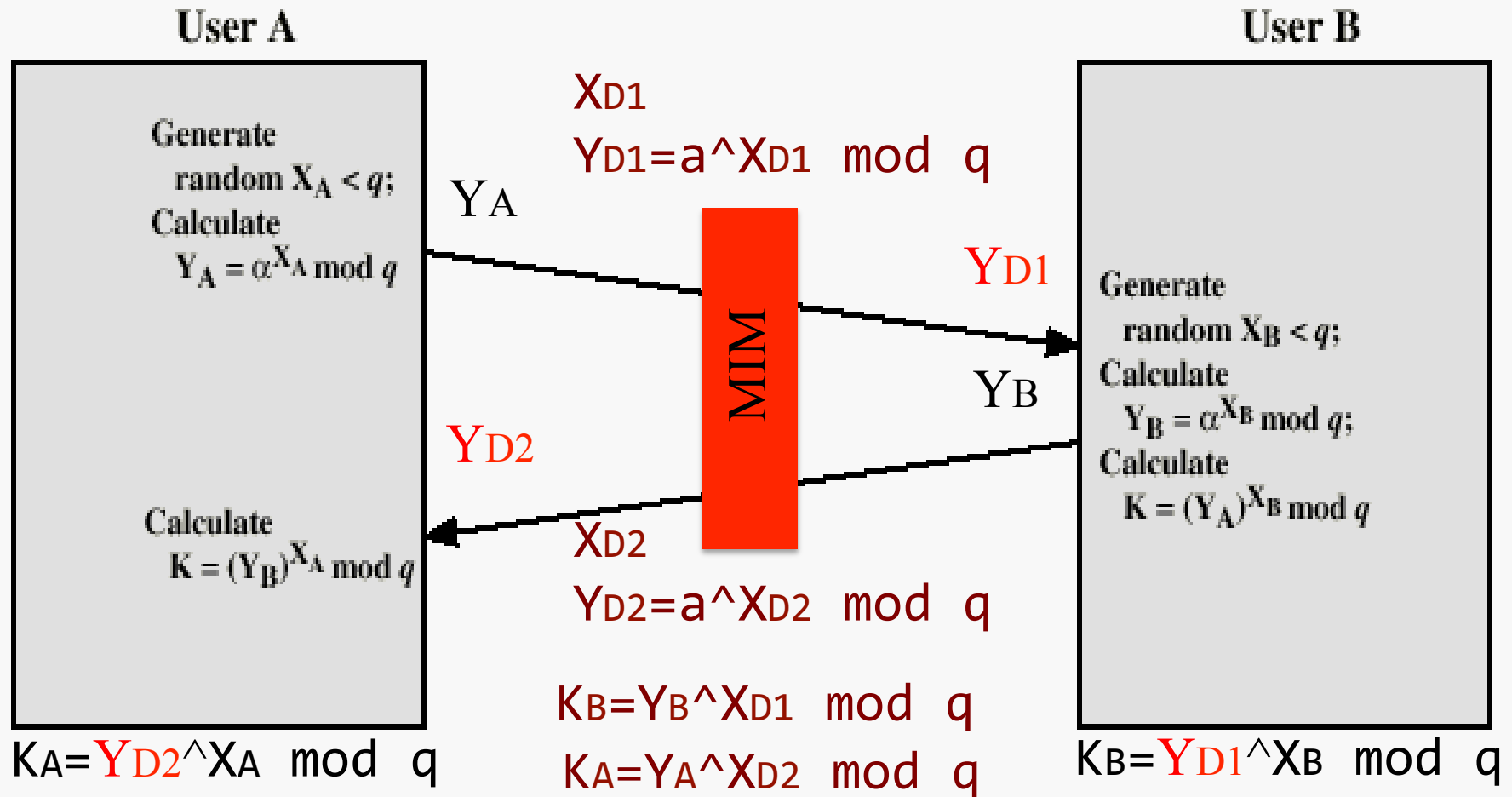
$$Y_B = 3^{233} \bmod 353 = 248$$

$$\begin{aligned} Y_A^{X_B} \bmod 353 \\ &= 40^{233} \bmod 353 \\ &= 160 \\ K_{ba} &= 160 \end{aligned}$$

Mallory can obtain K_{ab} ?

Mallory can attack as a "man in the middle" ?

D-H with a MIM Attack



$$C = \{M\}_{K_A} \quad \text{---} \quad M = \{C\}_{K_A} \quad \text{---} \quad C = \{M\}_{K_B}$$

D-H Key Exchange: Authentication Problem

- 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
 - Ephemeral D-H Agreement (EDH)
 - Fixed D-H Agreement (FDH)
- Both of these are vulnerable to a possible Meet-in-the-Middle (MIM) Attack
 - Why ?
 - Anonymous D-H agreement (ADH)
- Authentication of the exchanged values is needed
 - So, you will need Authenticated D-H agreements
 - How ?

Possible solution: Authenticated Key-Agreement

- Combination of D-H with another Public-Method allowing Digital Signatures covering the public D-H numbers exchanged by the principals involved

Alice sends to Bob $\text{Sign}_{K_{\text{priv}A}}(Y_a)$

Bob recognizes the signature and believes that Y_a is an authentic DH public number generated by Alice

Bob sends to Alice $\text{Sign}_{K_{\text{priv}B}}(Y_b)$

Alice recognizes the signature and believes that Y_b is an authentic DH public number generated by Bob

Authenticated Key-Exchange using Public-Key Methods for Digital Signatures and D-H

- We can use a public-key (asymmetric) method to support digital signatures to authenticate public Diffie-Hellman public numbers
 - Exemplified: RSA Signatures or DSA Signatures
- After the authenticated D-H exchange, the session key must be established independently by the principals involved
 - No problem with seed materials sent in the channel (because public D-H numbers are public !!!)
 - Contributive key generation (or contributive rekeying), with PFC and BFS guarantees
 - Perfect security with key generation control and key (or rekeying) independence

Multiparty DH Agreements

- DH Agreement is easily extensible for key-establishment protocols for multi-party environments
- Why ? How ?
 - We will see this in action later in practical classes (See Practical Labs)

Security of D-H

- The choice of G (*cyclic group generator*) and the generated element g
 - The order of G must be large enough ! Particularly in the case that the same group used for large amounts of traffic
 - G should have a large prime factor
 - Prevents optimized forms of solving the discrete logarithm problem (ex., Pohlig-Hellman Algorithm)
 - Vulnerability if generation of private numbers with no secure random generators
 - **Avoidance of using repeated DH numbers:** tradeoff between security and usability (performance) must be carefully addressed => source of some problems

Security of D-H

- State-of art (The best Discrete Logarithm Algorithms, ex., Number Field Sieve)
- Today: DH numbers of 2048, 3072 bits !
- Recommendation: signatures w/ ECCDH, using a *group generator* for P at least w/ 2048 bits

The case for ECDH

- ECC applied to DH Agreement (See the ECC as explained before):
 - Considering that X_a and X_b are the private numbers computed by A and B, using an Elliptic Curve, the computation of the correspondent public numbers will be:

$$Y_a = X_a G; \quad Y_b = X_b G;$$

With G the generation or base point chosen for the used curve

So, a shared (agreed) key K can be obtained, with the computation:

$$\begin{aligned} k &= X_a Y_b && // \text{ as computed by A} \\ &= X_a X_b G \\ &= X_b X_a G \\ &= X_b Y_a && // \text{ as computed by B} \end{aligned}$$

Revision: Suggested Readings



Suggested Readings:

W. Stallings, Network Security Essentials - Applications and Standards, Chap 3., sections 3.1, 3.2

Optional Complementary Readings



Optional / Complementary Readings:
(Internals of Secure Hash Algorithms), more detail for
those who are interested :

W. Stallings, Cryptography and Network Security - Principles
and Practices, Pearson - Prentice Hall, Chap. 8, 9, 10