DI-FCT-UNL
Segurança de Redes e Sistemas de Computadores
Network and Computer Systems Security

Mestrado Integrado em Engenharia Informática
MSc Course: Informatics Engineering
2020-2021, 1st Sem.

Public-Key Cryptography
(Asymmetric Cryptography)

In last lecture

- Foundations and details for the use of Symmetric Cryptographic Methods and Algorithms
  - Security concerns, applicability, padding and modes of operation
  - Important issues

Symmetric Cryptography: Issues (1)

- **Shared Keys** and/or Related Shared Secrecy Parameters
  - If a shared key is disclosed communications will be compromised (NDA of keys between principals involved).
    - Particularly delicate aspect of group-shared keys or long-term key reuse in multiple contexts (the same for secret association parameters or passwords, for ex.)
    - Dangers of key-exposure in large-scale sharing context
- **No base assumptions for peer-authentication and non-repudiation principles**
  - Does not protect sender from receiver forging a message & claiming is sent by sender (or vice versa)
    - Ex., No Peer-Authentication arguments when using Message Authentication Codes (CMACs and also HMACs)

Symmetric Cryptography: Issues (2)

- **Limitations for Perfect Secrecy Guarantees**
  - PFS (Perfect Forward Secrecy) or PBS (Perfect Backward Secrecy) conditions
- **Danger of compromising permanent (or long-term) shared keys (sometimes refereed as Master Keys)**
  - Long-term keys (as Master Keys) protecting short-term keys (ex., Session Keys)
    - Key Distribution/Rekeying Processes (for short-term or session keys distributed under the protection of master keys as long duration shared keys)

Symmetric Cryptography: Issues (3)

- **Other issues: quality of** key generation
  - Secure key maintenance control and non-disclosure conditions under the responsibility of KDCs (KEY DISTRIBUTION CENTERS) acting as central trusted parties
    - No control by principals (trustees)
    - No "Verifiable Contributive Key-Generation and Establishment Processes"
    - Furthermore, KDCs can be central points of failure or central targets for attacks

In this lecture ...

- Asymmetric Cryptography
  - Also known as Public-Key Cryptography

- **Outline:**
  - Public-Key cryptography principles
  - Public-Key algorithms
  - RSA algorithm
    - Key-Pair Generation and Encryption/Decryption
  - Diffie-Hellman key exchange
  - DSA
  - ECC

Outline

- **Asymmetric cryptography**
  - Public-Key cryptography principles
  - Public-Key algorithms
  - RSA algorithm
    - Key-Pair Generation and Encryption/Decryption
  - Diffie-Hellman key exchange

  \- - - - - - - - - - - - - - -

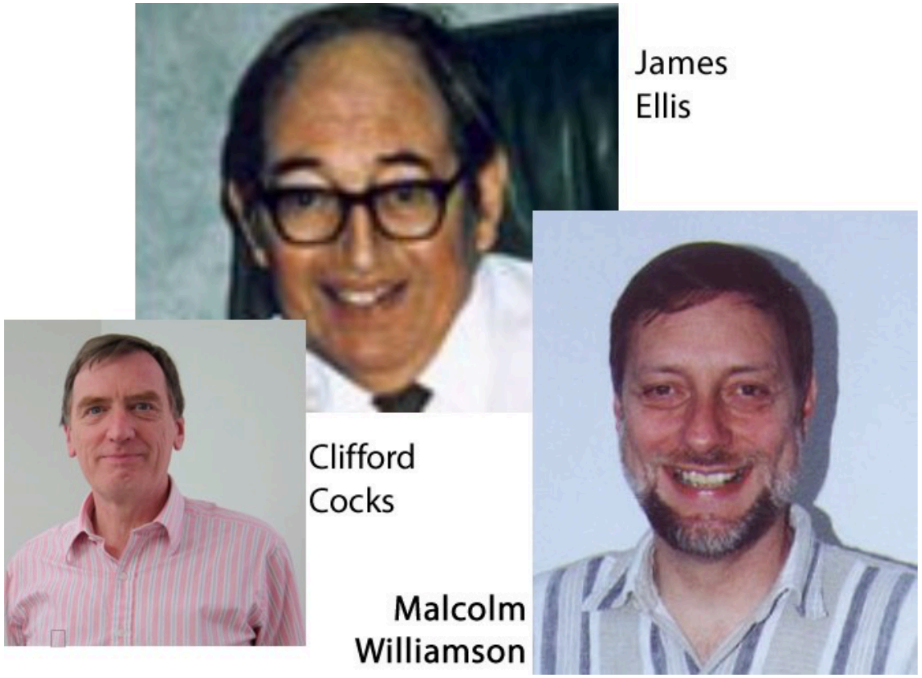
  - DSA
  - ECC

Public-Key Cryptography

- Probably most significant advance in 3000 years of history of cryptography …
  - https://en.wikipedia.org/wiki/RSA_(cryptosystem)
  - https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange
  - https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

- J.Ellis, M. Williamson, Clifford Cocks (British Intelligence/GCHQ first in 1973, declassif. In 1997)
- Whitfield Diffie & Martin Hellman, Stanford University (1976)
- Ron Rivest, Adi Shamir, Leonard Adleman (1978) (RSA)
- Neal Koblitz (1985) and Victor Miller (1985) (ECC)
- Emergent Public-Key Crrypto: Homomorphic and Quantum Crypto

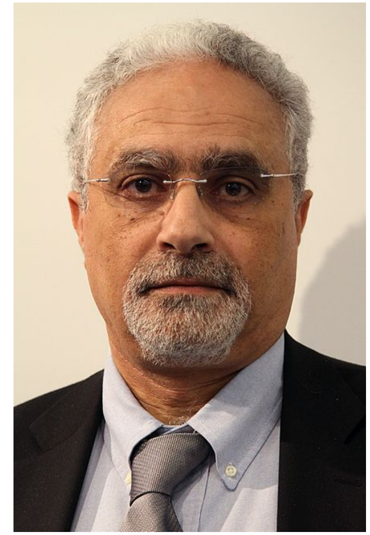
Some Public-Key Cryptography Pioneers



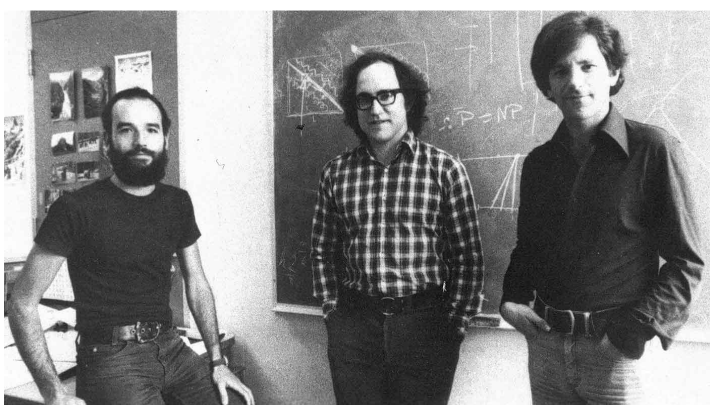
James Ellis

Clifford Cocks

Malcolm Williamson

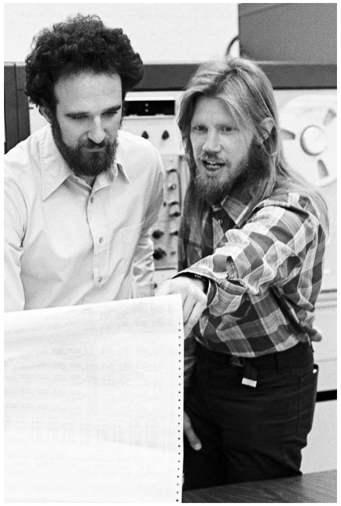


Taher A. Elgamal



Ron Rivest,
Adi Shamir and
Leonard Adleman

Some Public-Key Cryptography Pioneers



Whitfield Diffie and Martin Helman (Touring Award 2015)

Diffie-Hellman

Neal Koblitz

Victor Miller

Elliptic-Curves Cryptography

Homomorphic Cryptography

Craig Gentry

Pascal Paillier

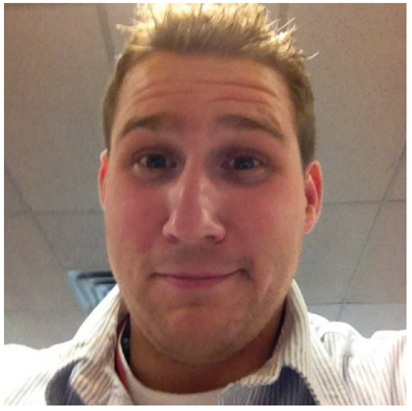
Shafi Goldwasser and Silvio Micalli

Fully Homomorphic Cryptography

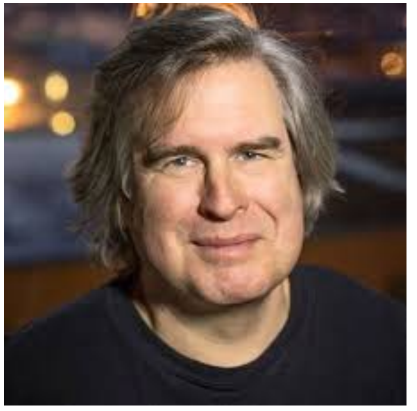
Partial Homomorphic Cryptography

Josh Benaloh

Some Quantum Crypto Pioneers



Stephen Wiesner



Gilles Brassard

Charles Bennet

David Deutsh

Quantum Cryptography

Public-Key Cryptography

- **Foundations**:
  - Number theory concepts and functions (D-H, RSA, DSA, ElGamal), Factorization and Prime Number Properties, Modular Arithmetic
  - Algebraic structures of elliptic curves over finite fields (ECC)

Note: Asymmetric Crypto computations more complex (slow) than symmetric encryption and hash processing

- **See, ex (benchmarks):**

```
$ openssl speed rsa dsa ecdsa ecdh des-ede3  blowfish aes sha1 sha256
```

Comparative Performance of crypto methods

```
The 'numbers' are in 1000s of bytes per second processed.
type              16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
sha1             15948.36k    34486.04k    59030.78k    72485.16k    75988.33k
des ede3         10664.47k    10887.42k    11000.83k    10635.26k    10807.98k
blowfish cbc     52188.69k    56682.20k    58439.83k    57829.38k    57455.96k
aes-128 cbc     124214.88k   130045.31k   129590.87k   130960.38k   129077.43k
aes-192 cbc     107241.83k   110602.70k   111848.28k   114678.41k   111719.77k
aes-256 cbc      93615.96k   101543.21k   103047.68k   102965.93k   100832.83k
sha256            7578.61k    15069.78k    25287.58k    30401.54k    32098.99k
                  sign    verify    sign/s verify/s
rsa  512 bits 0.000836s 0.000084s   1196.4  11868.2
rsa 1024 bits 0.004869s 0.000421s    205.4   2378.0
rsa 2048 bits 0.033557s 0.001575s     29.8    635.1
rsa 4096 bits 0.217391s 0.005722s      4.6    174.8
```

RSA Encryption >>>>>> SHA256 > SHA1 : $\sim 10^6 – 10^7$

RSA Sig Verif. > Sig : ~10

RSA >>>> >> 3DES > DES > BF > AES : $\sim 10^6 - 10^7$

3DES > DES > SHA256 > BF > SHA1 > AES

Comparative Performance of crypto methods

	sign	verify	sign/s	verify/s
160 bit ecdsa (secp160r1)	0.0004s	0.0016s	2768.6	643.0
192 bit ecdsa (nistp192)	0.0004s	0.0015s	2805.4	682.3
224 bit ecdsa (nistp224)	0.0005s	0.0022s	1951.5	452.4
256 bit ecdsa (nistp256)	0.0006s	0.0028s	1614.3	354.7
384 bit ecdsa (nistp384)	0.0014s	0.0071s	720.2	141.2
521 bit ecdsa (nistp521)	0.0029s	0.0152s	346.7	66.0
163 bit ecdsa (nistk163)	0.0005s	0.0022s	2072.9	448.7
233 bit ecdsa (nistk233)	0.0009s	0.0031s	1064.9	323.6
283 bit ecdsa (nistk283)	0.0016s	0.0068s	632.8	146.7
409 bit ecdsa (nistk409)	0.0038s	0.0145s	265.7	68.9
571 bit ecdsa (nistk571)	0.0084s	0.0327s	119.4	30.6
163 bit ecdsa (nistb163)	0.0005s	0.0024s	2044.5	411.9
233 bit ecdsa (nistb233)	0.0010s	0.0033s	1038.6	301.0
283 bit ecdsa (nistb283)	0.0015s	0.0075s	650.6	133.1
409 bit ecdsa (nistb409)	0.0037s	0.0162s	270.5	61.9
571 bit ecdsa (nistb571)	0.0082s	0.0357s	122.2	28.0

RSA Sig >> ECDSA Sig : ~10^1 to 10^3

RSA Sig Verif. < > ECDSA Sig

but ECC keysizes < RSA keysizes for sama level of security (afawk)

Comparative Performance of crypto methods

	op	op/s
160 bit ecdh (secp160r1)	0.0013s	772.7
192 bit ecdh (nistp192)	0.0012s	823.8
224 bit ecdh (nistp224)	0.0018s	541.7
256 bit ecdh (nistp256)	0.0023s	431.4
384 bit ecdh (nistp384)	0.0057s	176.4
521 bit ecdh (nistp521)	0.0124s	80.6
163 bit ecdh (nistk163)	0.0011s	932.3
233 bit ecdh (nistk233)	0.0015s	663.7
283 bit ecdh (nistk283)	0.0034s	298.5
409 bit ecdh (nistk409)	0.0071s	141.5
571 bit ecdh (nistk571)	0.0161s	62.3
163 bit ecdh (nistb163)	0.0012s	839.7
233 bit ecdh (nistb233)	0.0016s	609.7
283 bit ecdh (nistb283)	0.0037s	268.9
409 bit ecdh (nistb409)	0.0081s	124.1
571 bit ecdh (nistb571)	0.0180s	55.5

Signed DH >> DH >> ECDH
ECDH comparable woth ECDSA (Sig and Sig Verif)

Hybrid Constructions

- **For practical purposes (security vs. usability vs. performance)** we use hybrid constructions

Ex. of Typical Constructions for Secure Communication:

$$\underbrace{\{Ks, \ldots Km, \ldots\}_{Kpub} \;||\; \{M\}_{KS} \;||\; \text{Digital Sig }(M)}_{} \;||\; MAC_{Km}(C)$$

(C = $\{M\}_{KS} \;||\; \text{Digital Sig }(M)$)

$$\{Ks, \ldots Km, \ldots\}_{Kpub} \;||\; \{M \;||\; MAC_{KM}(M)\}_{KS} \;||\; \text{Digital Sig }(M)$$

Etc…
Constructions can optimize for specific uses the tradeoff:
<Security vs. Usability vs. Performance>

Ex., Can you understand TLS Ciphersuitres ?

- Can you understand the TLS standardized Ciphersuites as the Hybridization of Different Cryptographic Methods ?

- Ex., Labels for Ciphersuites for JSSE in Java:

  - https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html#ciphersuites

- Ex., Labels for Ciphersuites in OPENSSL

  - https://www.openssl.org/docs/man1.0.2/man1/ciphers.html

Use of Asymmetric Cryptography

Use of Public-Key Cryptography

Public-key/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 *

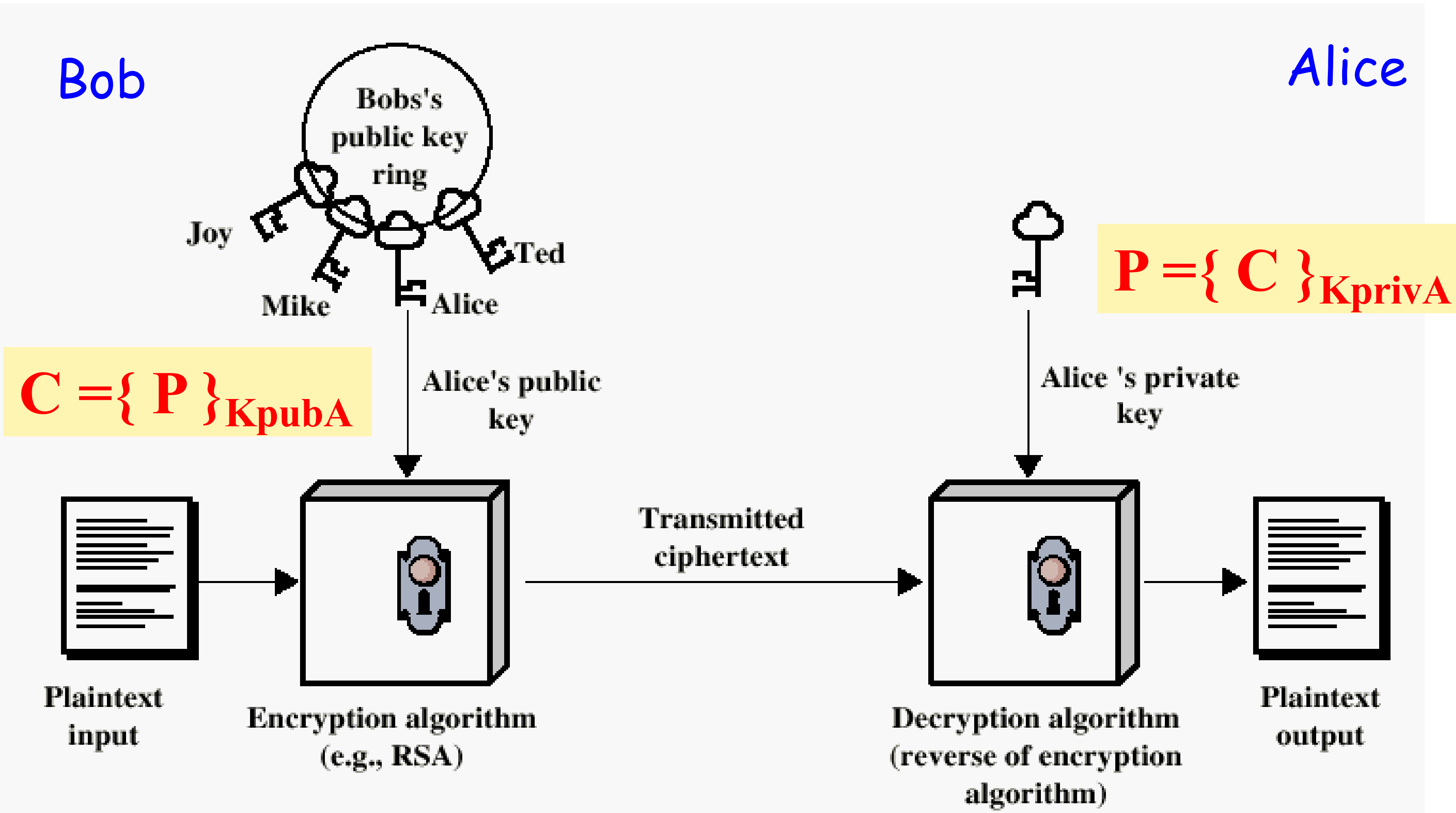
Same function (same computation) for encryption and for decryption

(*) Note the difference w/ Symmetric Encryption: use the **same (shared) key for Encryption and Decryption** with **different Encryption and Decryption computations**

Encryption using a Public-Key System

For confidentiality principles:

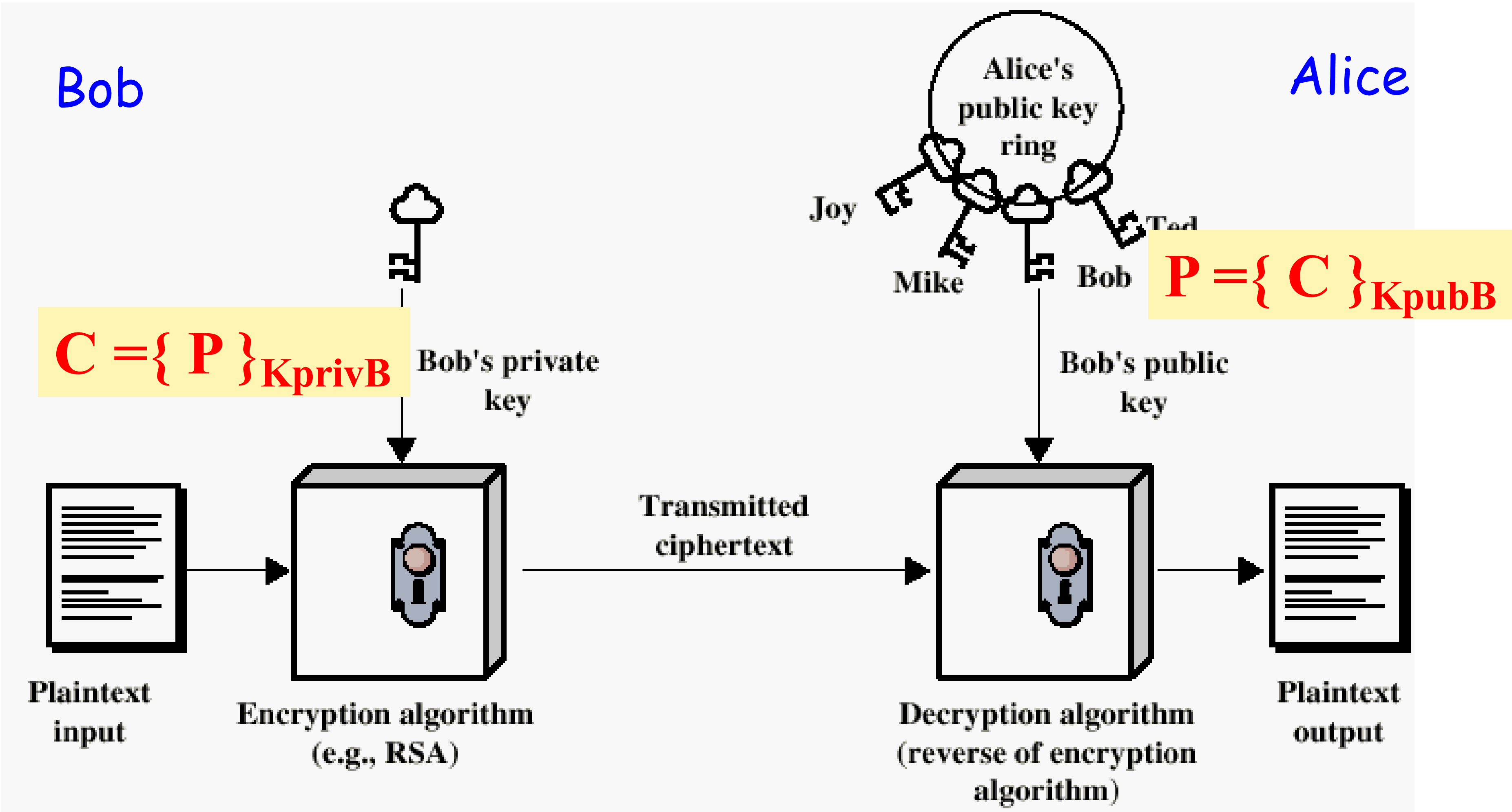
Encryption with the destination Public key



Authentication using Public-Key System

For authentication principles:

Encryption with the sender Private Key



Public-Key Cryptography Assumptions

- In **asymmetric** methods:
  - Those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures
  - Considering the key pair, **what is encrypted with one key pair, is decrypted by the other key of that pair** **(for well-known algorithms)**
  - Encryption and Decryption functions implemented by the same computation

For ex: in RSA (Integer Modular Arithmetic)

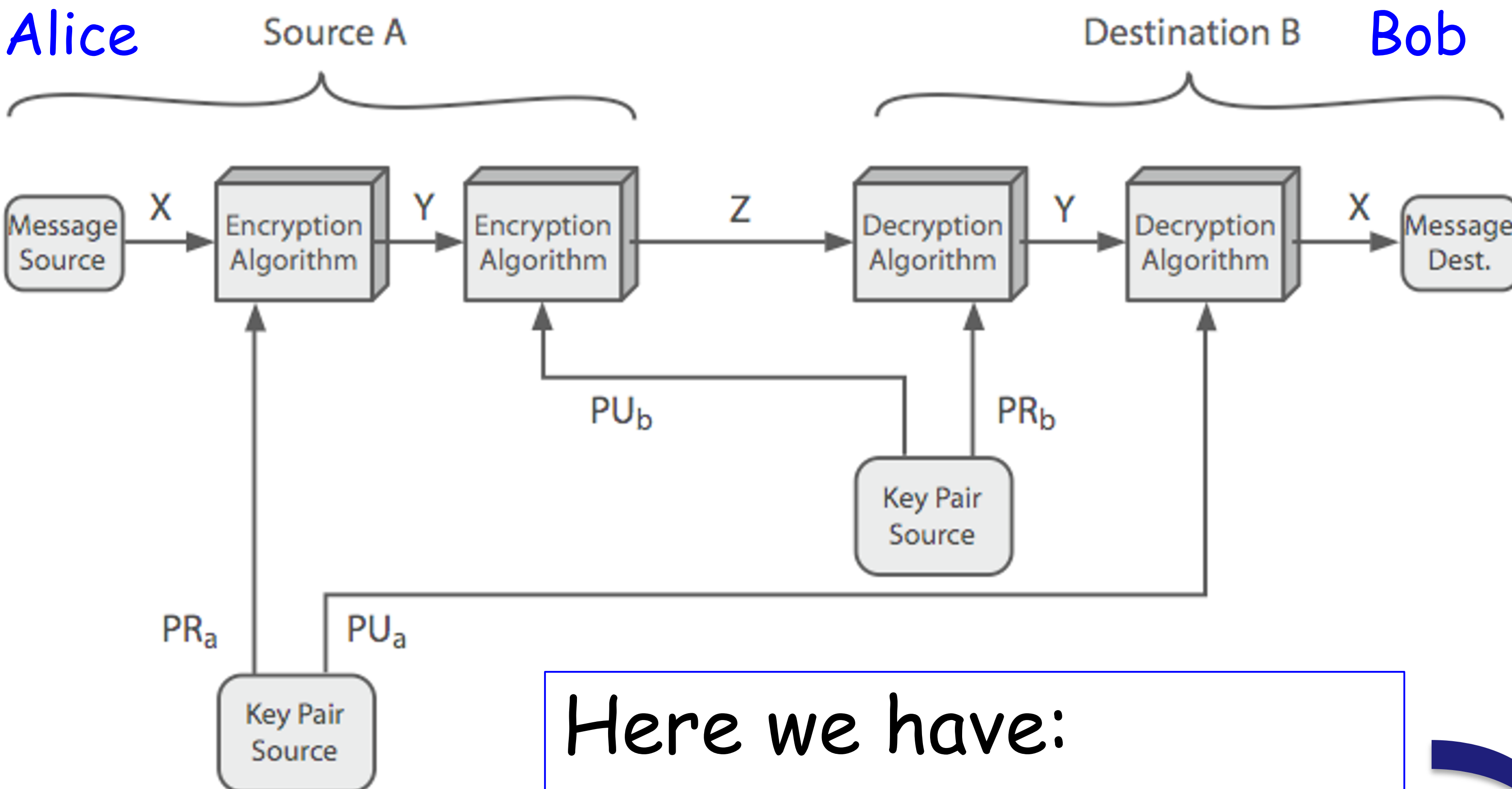
$C = P^{Kpub} \bmod N$ for Encryption

$P = C^{Kpriv} \bmod N$ for Decryption

Keypair: [Kpriv, Kpub]

Exactly the same computation with different operators

Confidentiality + Authentication



Here we have:

$\{\ \{\ M\ \}_{KprivAlive}\ \}_{KpubBob}$

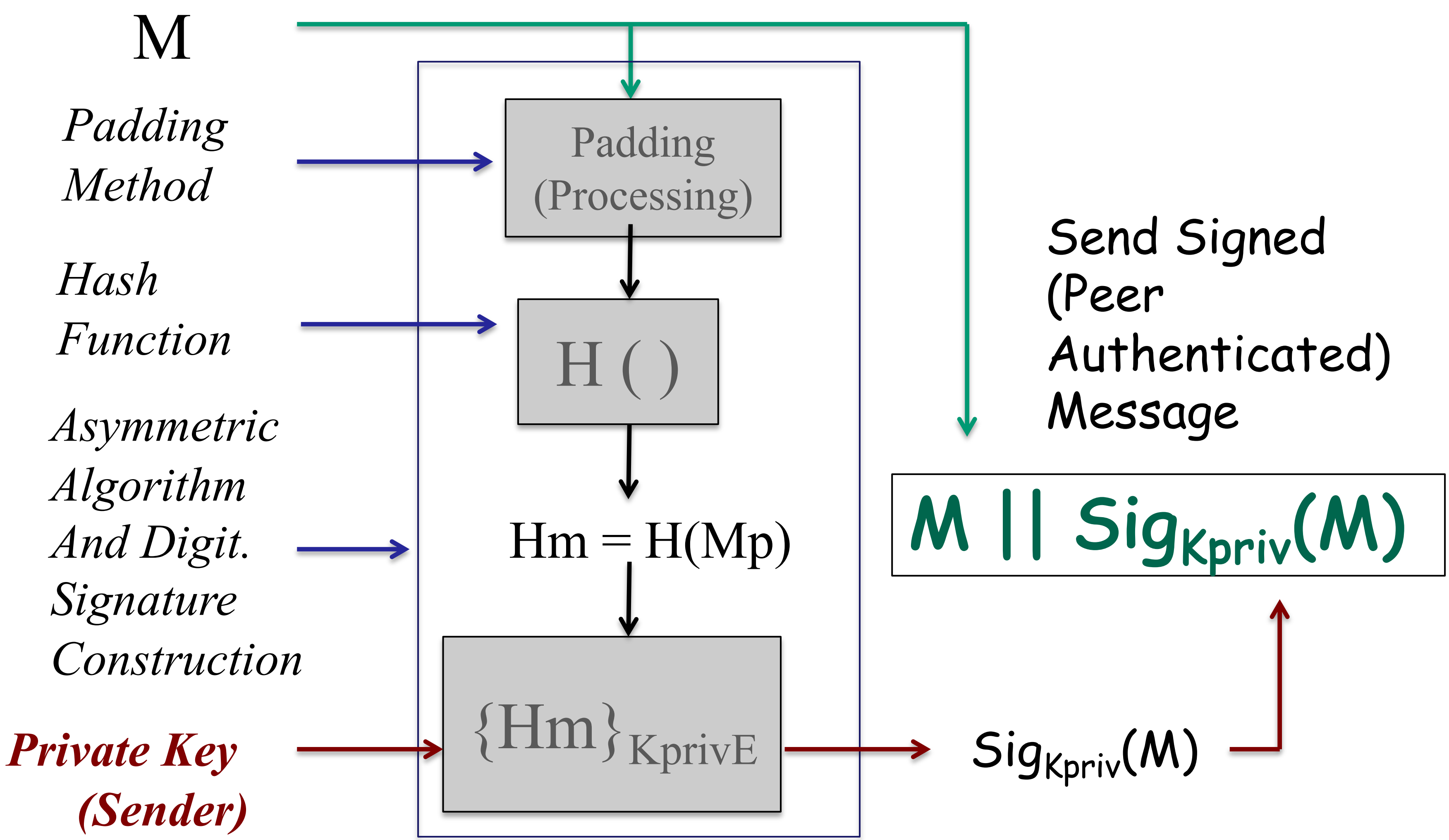
Can we do better for practical use ? How ?

$\{M\}KprivA\ ||\ \{M\}_{KpubBob}$ is wrong !!! Why ?

Design Principle for Digital Signatures (for Peer-Authentication)

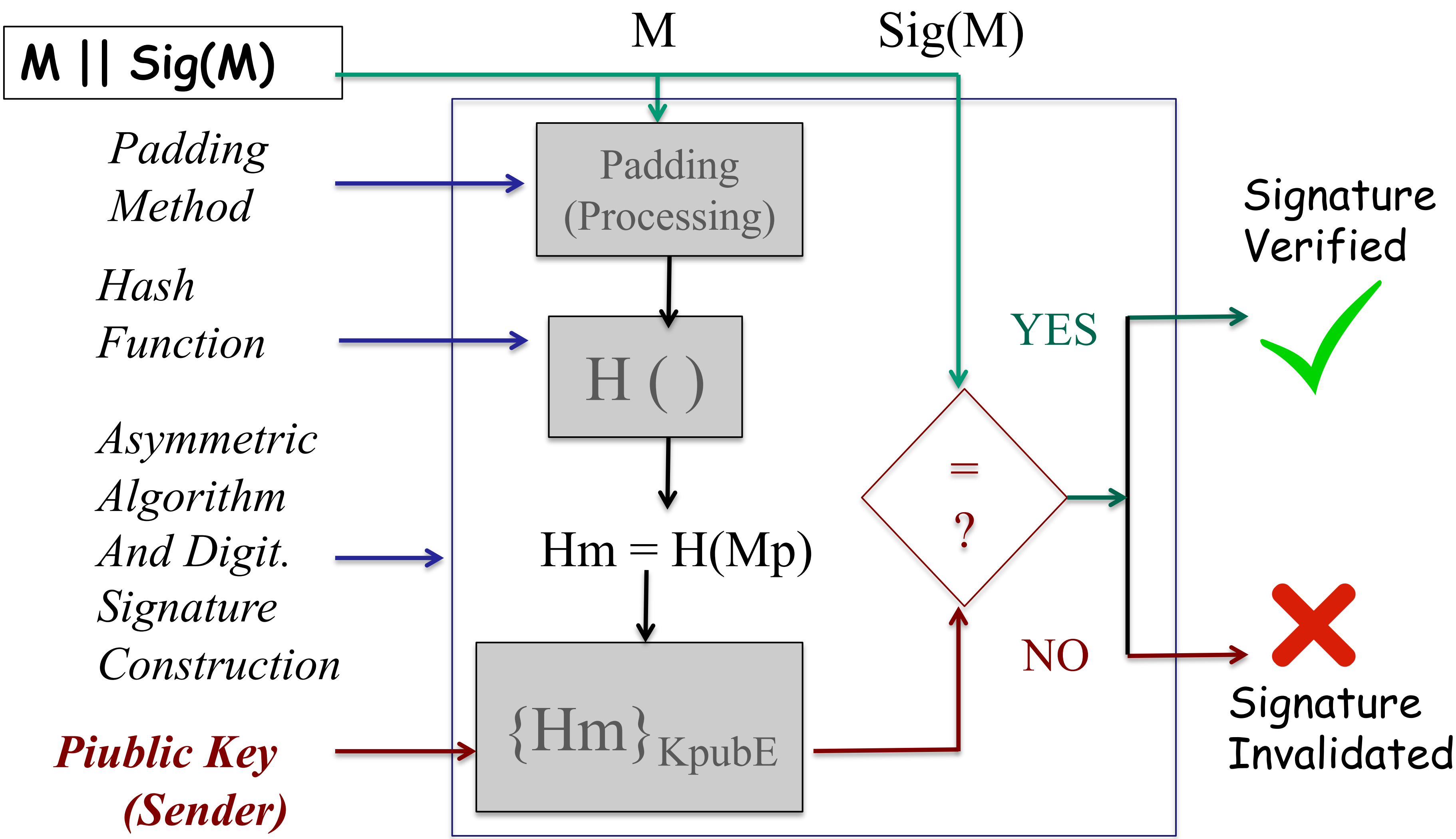
Base Scheme for Authentication

- **Principle of construction of Digital Signatures Schemes: Sender**



Verification (recognition) of Digital Signatures

- **Principle of Verification of Digital Signatures Schemes: Receiver Verification**



Design Principle for Confidentiality

Public Key Envelopes w/ Symmetric "Session" Keys
+
Encryption with Symmetric Encryption

Use for Confidentiality

Alice: send **M** to Bob with confidentiality

- Generates a Session Key Ks for Symmetric Crypto Algorithm
- Decide the required security associations (ex., IVs and other considered security association parameters SAPs)

Alice send to Bob:

$$\{Ks, \langle SAPs \rangle\}_{KpubDest} \;||\; \{M\}_{Ks}$$

Public Key Envelope

Much better ! Why ?
Think on
"Security vs. Performance" vs. Flexibility

Confidentiality + Integrity + Message Authenticity

Alice: send M to Bob with confidentiality

- Generates a Session Key Ks for Symmetric Crypto Algorithm
- Decide all required security associations (ex., IVs and other considered security association parameters SAPs)
- Decide on the use of Hash Functions or MAC construction
- Generates a MAC key

Alice send to Bob: Confidentiliaty + Integrity

$\{Ks, \langle SAPs \rangle\}_{KpubDest} \; || \; \{M \; || \; H(M)\}_{Ks}$

Or Confidentiliaty + Integrity + Message Authentication

$\{Ks, Km, \langle SAPs \rangle\}_{KpubDest} \; || \; \{M\}_{Ks} \; || \; MAC_{Km}(M)$

$\{Ks, Km, \langle SAPs \rangle\}_{KpubDest} \; || \; \{M\}_{Ks} \; || \; MAC_{Km}(\{M\}_{Ks})$

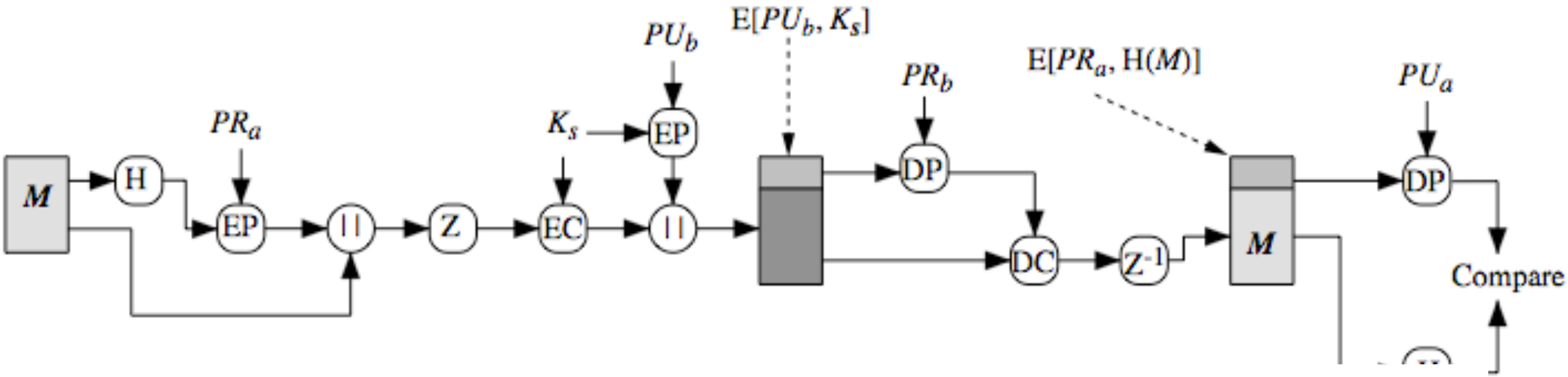
Use of public-key cryptography in general

- **Confidentiality and Authentication**
  - **Verification by each principal, based on correct and trusted 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., Keys (or other secrecy parameters) generated by *Senders* and distributed to *Receivers* in confidential envelopes protected by the destination Public Key:
  - Some Other Assym. Crypto Methods are specifically targeted for Key-Exchange: ex., DH - Diffie Hellman, or GDH)

Ex. Hybrid use with different Crypto. Methods

- Example (in PGP - Preety Good Privacy)



Confidentiality + Authentication
Public-Key Method + Symmetric Encryption + Cryptographic hash

Note in this case:
Compression always before encryption !
Compression always after signature !

Why ?

Security Properties in Asymmetric Cryptography

Properties of Public-Key Cryptography (1)

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

   BOB: public key: *KpubB*; private key: *KprivB*

   ALICE: public key: *KpubA*; private key: *KprivA*

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

$$C = \{\mathrm{M}\}_{KpubB}$$

3. Easy for the receiver (B) to decrypt *ciphertext* using the correct private key

$$M = \{C\}_{KprivB} = \{ \{\mathrm{M}\}_{KpubB} \}_{KprivB}$$

Properties of Public-Key Cryptography (2)

4. Computationally **infeasible** to determine private key (*Kpriv*) knowing the related public key (*Kpub*)

5. Computationally **infeasible** to recover message M, knowing *Kpub* and ciphertext *C*

6.* Either of the two keys can be used for encryption, with the other used for decryption (**depending on the algorithms and purpose**):

$$M = \{ \{M\}_{Kpub} \}_{Kpriv} = \{ \{M\}_{Kpriv} \}_{Kpub}$$

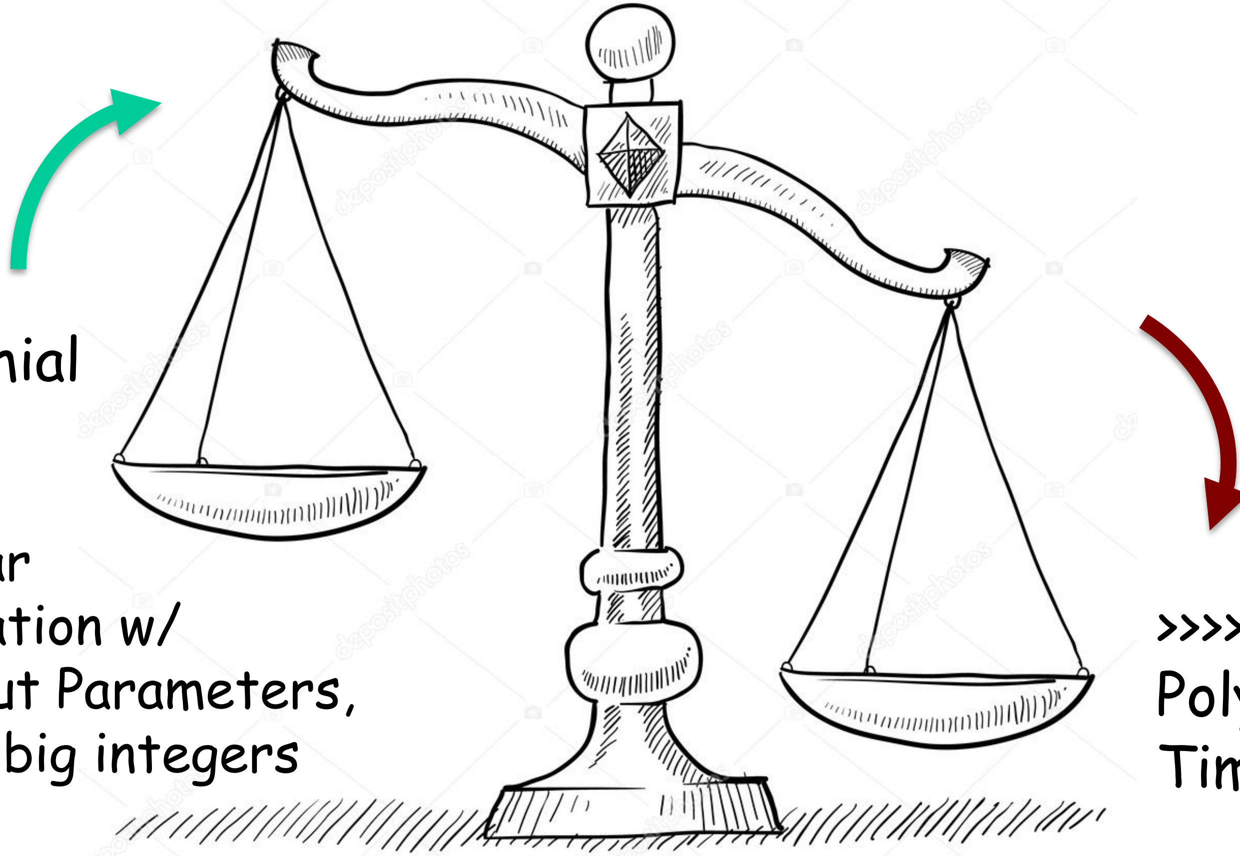
---

*) In practice, some Asymmetric Algorithms used for different purposes

What means "*easy*" or "*unfeasible*" ?

- **Easy, Feasible**: something computationally solved (bound) in polynomial time, as a function of the input length
  - Input: *n* bits => function proportional to n^a, with pre-known **a = fixed constant**
  - Ex., RSA, DH. DSA: Modular exponentiations with Functions of class P (Prime Numbers and Properties of Functions w/ Prime Numbers)
- **Unfeasible**: if the effort to compute grows faster (much high complexity) than polynomial time
  - Ex., RSA, DH, DSA: Prime Factorization of Big Numbers (Big Integers) + Computation of Discrete Logarithm Problem with very large exponents

Computational Cost
"Easy" (feasible) vs. "Unfeasible"



<= Polynomial Time

Ex: Modular Exponentiation w/ Known Input Parameters, even using big integers

>>>>> than Polynomial Time

Ex: Solve the discrete Logarithm Problem given a a very big number

Use of Padding Processing for Asymmetric Cryptography

Encryption/Decryption using Public Key Algorithms

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

Key pair: <Kpub, Kpriv>

$$C = \{ M \}_{Kpub}$$

$$P = \{ C \}_{Kpriv}$$

With no Padidng

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

Key pair: <Kpub, Kpriv>

$$C = \{ Padding \,||\, M \}_{Kpub}$$

$$P = \{ C \}_{Kpriv}$$

With Padidng

Ex., Padding for RSA: PKCS#1

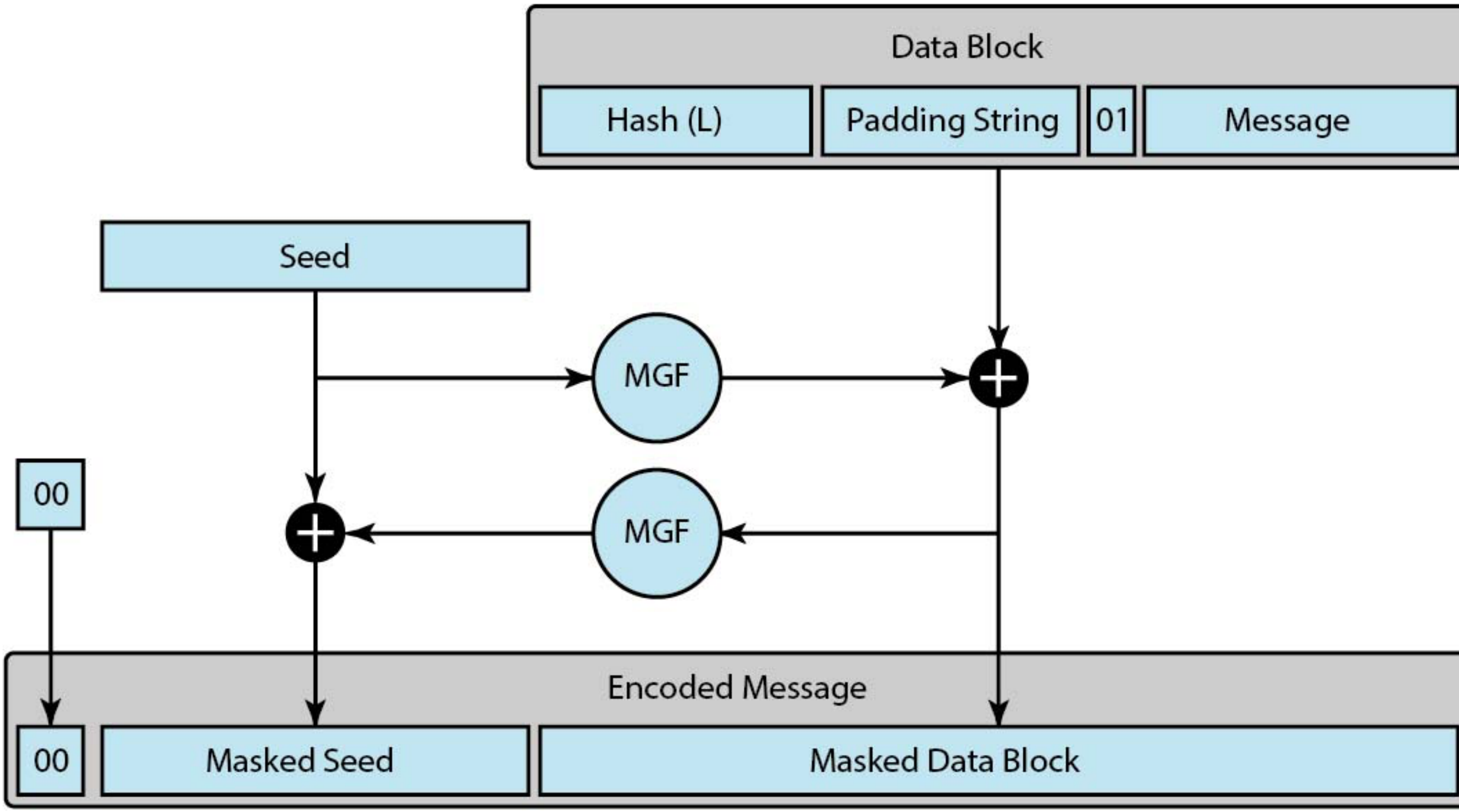
- Form of structured, randomized values, added to plaintext M (on the left) before encryption assuring that:
  - The M value (as an integer) does not fall into the range of insecure plaintexts
  - M, once padded, will encrypt to one of a larger number of different possible *ciphertext* numbers !

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 available PSSs based schemes w/ patents expired in the period 2009 and 2019

Example: RSA-OAEP

- Optimal Asymmetric Encryption Padding
- Published at Eurocrypt 2000 (Coron et al.,) Crypto 1998



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: <Kpub, Kpriv>

$$Sig(M) = \{ H (Padding \,//\, M) \}_{Kpriv}$$

Ex: RSA-PKCS#1, RSA-PSS

RSA PKCS#1 (v1.5)

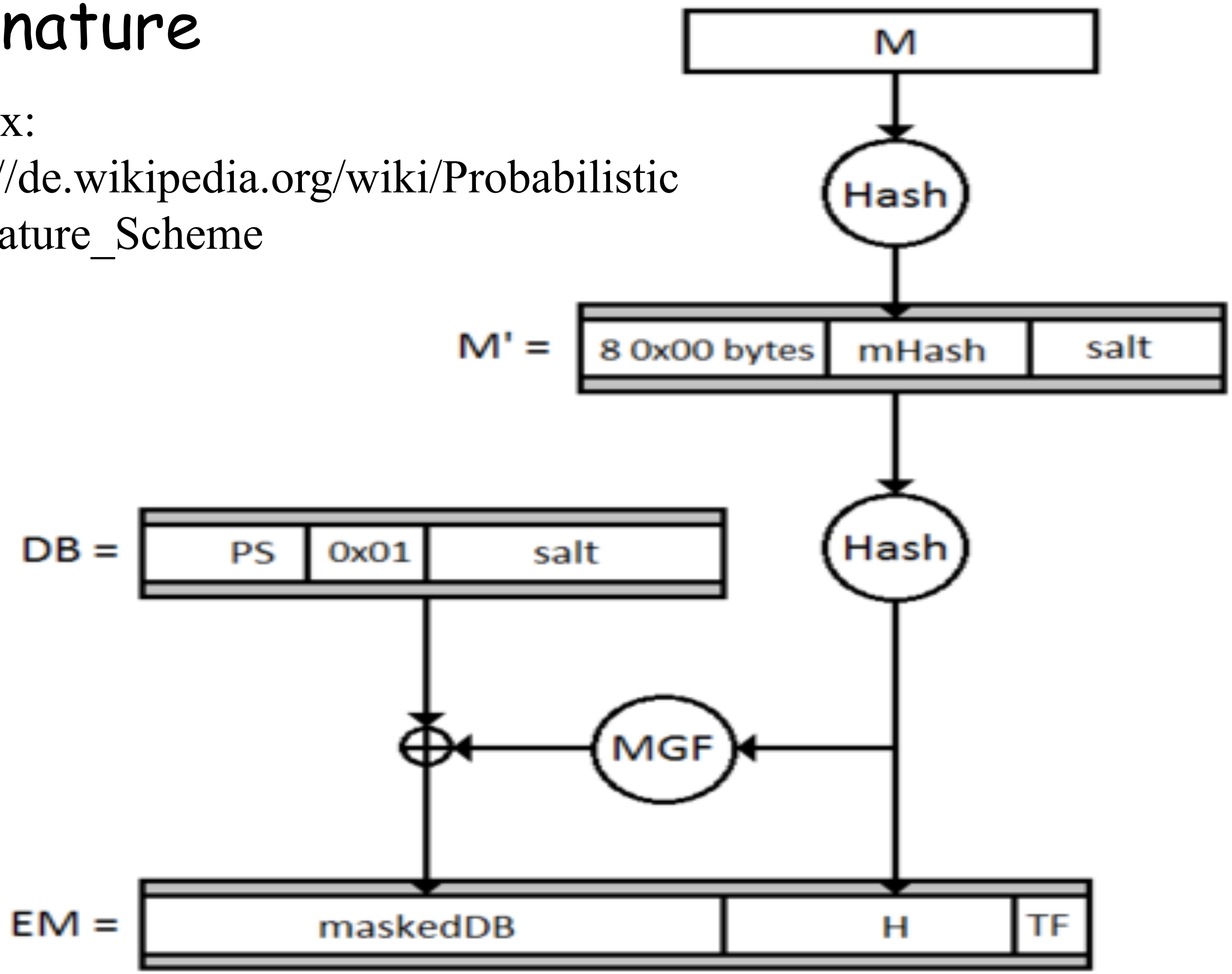
PKCS #1 (v1.5): Padding Formats and Usage

- Sign: 01 || ff … ff || 00 || DER(HashAlgID,Hash(*M*))
- Encrypt: 02 || pseudorandom PS || 00 || M
- Ad hoc design
- Widely deployed, incorporated in many Internet standards, such as:
  - PKIX profile
  - TLS
  - IPSEC
  - S/MIME

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

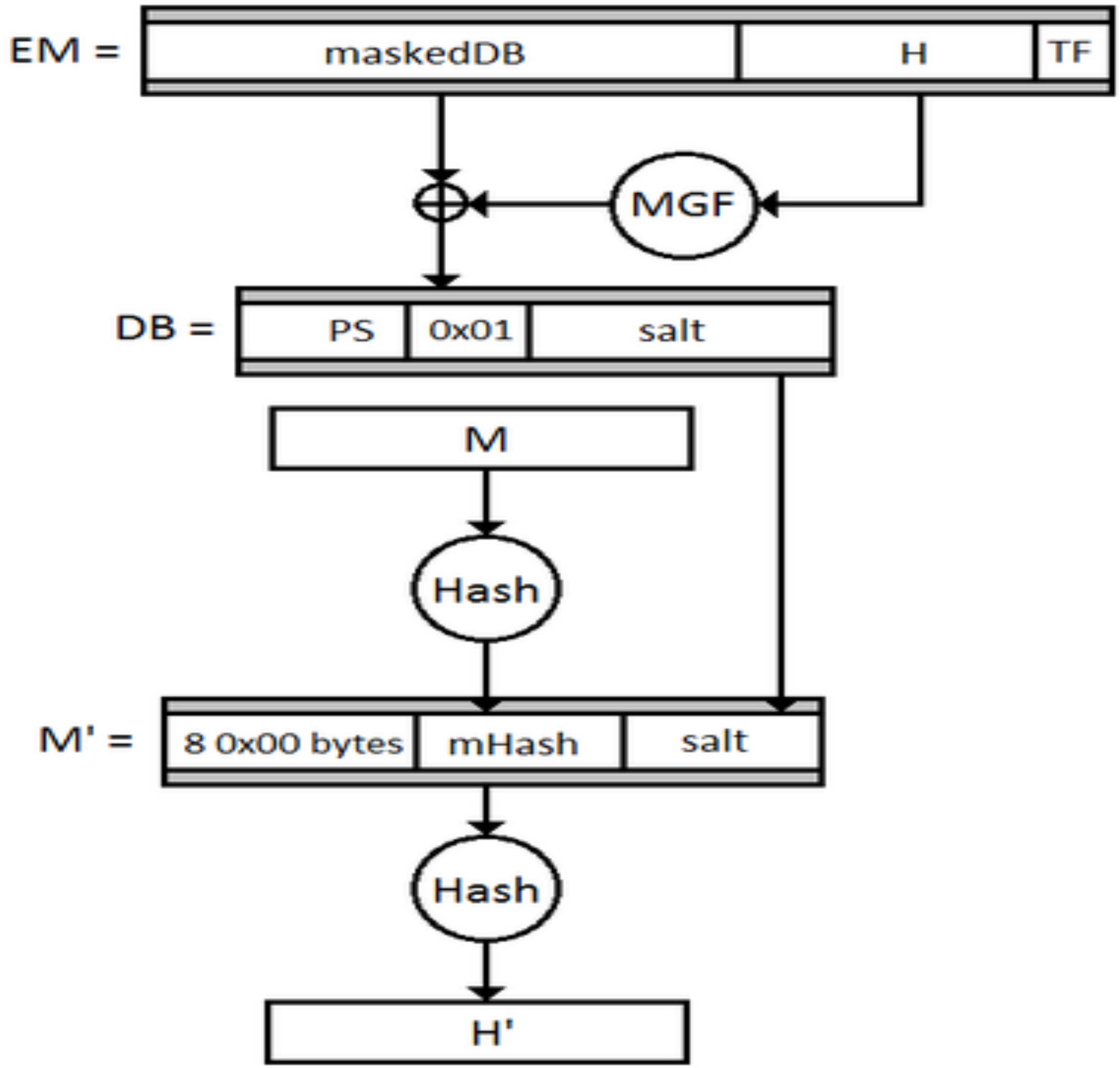
Signature

See, ex:
https://de.wikipedia.org/wiki/Probabilistic_Signature_Scheme



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

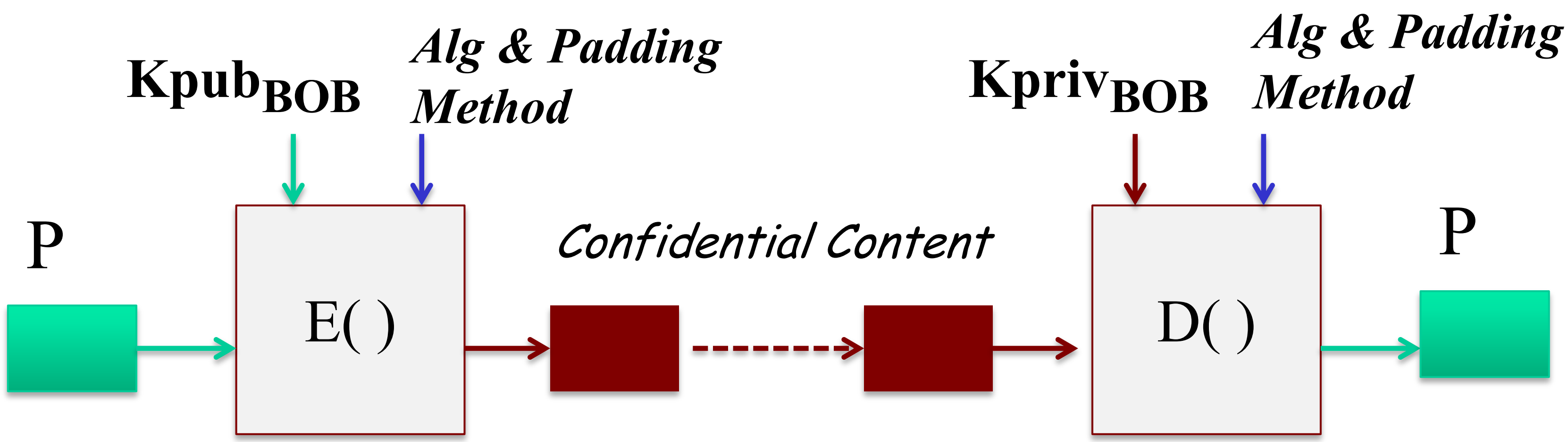
Signature Verification



Practical use in Summary
(See Padding Exercises in next LABs)

Practical use in summary (Alice > Bob):

For Encryption (Confidentiality, Secure Envelopes):

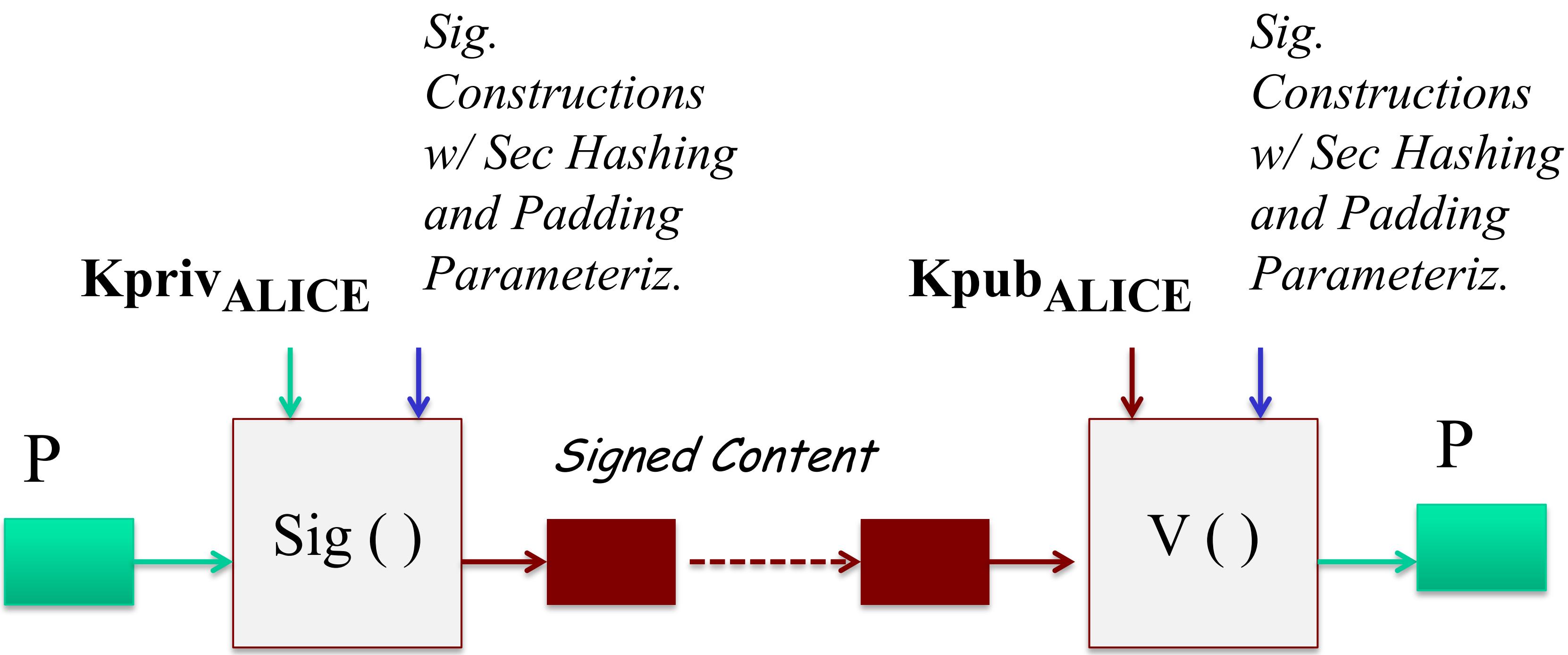


We must use Secure and Standardized Constructions (provided in available crypto libraries or crypto-provider implementations)

=> TRUSTED COMPUTING BASES !

Practical use in summary (Alice > Bob):

For Authenticity
(Signed Content w/ Sender Peer-Authenticity Guarantees):



Must use secure and classified patterns (standards) for Digital Signatures and Verifications, involving the combination of Asymmetric Crypto Alg., Secure Hash Functions and Secure Padding Processing

Outline

- **Asymmetric cryptography**
  - Public-Key cryptography principles
  - Public-Key algorithms
  - RSA algorithm
    - Key-Pair Generation and Encryption/Decryption
  - Diffie-Hellman key exchange

  - DSA
  - ECC

Public Key Algorithms

Different algorithms …

Pay attention:

Asymmetric Algorithms are used for their specific purposes (and purposes are combined for different secure protocols and services), ex:

Encryption/ Decryption	Digital Signatures	Key (or Secrets) Exchange
RSA, ElGamal *ECC-Curves* *Paillier, Cramer-Shoup* *Knapsack, …*	*DSA,* *ECDSA* …	*DH,* *ECDH* …

RSA: Rivest, Shamir & Adleman, MIT, 1977

- 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
- 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
  - nb. exponentiation takes $O((\log n)^3)$ operations (feasible)
- Uses large integers (eg. 1024, 2048, 4096 bits)
- Security due to cost of factoring large numbers
  - nb. factorization takes $O(e^{\log n \log \log n})$ operations (hard)

RSA and Math involved

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

DH, El Gammal, 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 Gammal**
  - Block Cipher
  - Unfeasible solution of discrete logarithms (computational time and complexity)
- **Digital Signature Standard (DSS) or DSA**
  - Initially Make use of the SHA-1 (recent standardization can use other Hash funcitons (SHA-2 and SHA-3)
  - For digital signatures (only), not for encryption or key echange
  - 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 growing

- Majority of public-key crypto (RSA, D-H) use either integer or polynomial arithmetic with very large numbers/polynomials
- Imposes a significant load in storing and processing keys and messages
- ECC appears as an alternative for offering same security with smaller bit sizes
- Newer, but not as well (crypt)analyzed // Ongoing Research
- Standardization problem: different ECC curves and characteristics

Comparable Key Sizes for Equivalent Security

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

Other Public-Key Algorithms …

Other public-key algorithms:

- Knapsack, Pohlig-Hellman, Rabin, McEliece, LUC, Finite Automaton, Paillier, etc.

Public-Key signature algorithms:

- DSA variants, GOST, Discrete Logarithm Variants,
- Ong-Schnorr-Shamir, ESIGN, etc.

See also:

Bruce Schneier, Applied Cryptography, Wiley, 2006

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

Practical Use

- RSA Enc/Dec w/ Padding (PKCS#1 and OAEP)
- PKCS#1, PSS Padded Digital Signatures w/ RSA
- ElGamal Enc/Dec w/ Padding
- Use od DSA and ECDSA (Elliptic Curve) Digital Signatures
- Construction of Secure and Authenticated Envelopes
  - Public Key Envelopes for Distribution of Symmetric Keys and Security Association Parameters
  - Key-Wrapping (Protection) Techniques
  - Protection of Private Keys wrapped w/ Symmetric Encryption Keys

Outline

- **Asymmetric cryptography**
  - Public-Key cryptography principles
  - Public-Key algorithms
  - RSA algorithm
    - Key-Pair Generation and Encryption/Decryption
  - Diffie-Hellman key exchange

  \- - - - - - - - - - - - - - -

  - DSA
  - ECC

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	Kpub = {e,n}
7.	Private key	Kpriv = {d,n}

1) Ex., 7, 17 2) n = 7 x 17 = 119 3) ø(n) = 6 x 16 = 96

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

5) 5xd = 1 x mod 96, com d< 96 d=77
5 x 77 = 385 , notar que 4x96+1 =385

Kpub = (5, 119)

Kpriv = (77, 119)

The RSA Algorithm: Encryption/Decryption

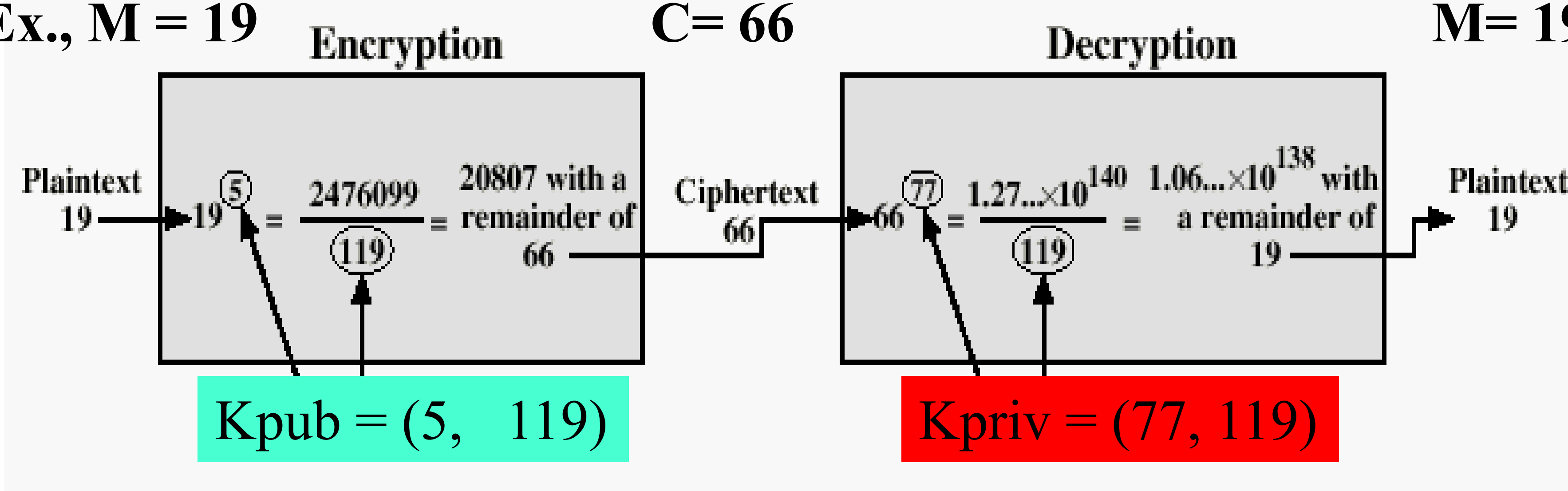
Encryption: $C = \{P\}_{Kpub}$

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

Decryption: $P = \{C\}_{Kpriv}$

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

Ex., M = 19 **C= 66** **M= 19**



Another RSA Example - Key Setup

1. Select primes: p=17 & q=11 (secrets)
2. Compute $n = pq$ =17x11=187
3. Compute $\varnothing(n)=(p-1)(q-1)$=16x10=160
4. Select e: gcd(e,160)=1; choose e=7
5. Determine d: de=1 mod 160
   and d < 160 Value is d=23 since
   23x7=161= 10x160+1
1. Publish public key Kpub={7,187}
2. Keep secret private key Kpriv={23,187}

Another RSA Example - Encrypt/Decrypt

- given message $M = 88$ (nb. $88<187$)
- encryption:

  $C = 88^7 \bmod 187 = 11$
- decryption:

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

More about RSA

- See W. Stallings, Network Security Essentials
  - Chap. 3 – Public Key Cryptography and Message Authentication
    - See, Sections 3.4 to 3.6

Security vs Practical Use (ex. RSA)

Security considerations

- **Math Attacks:**
  - Evolving Methods for Optimization in Factoring the product of two big primes and relatively primes

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

EU Agency for Network and Information Security : Algorithms, Key Size and Parameters Report), Nov 2014): 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 processing-delay

- **Chosen Ciphertext Attacks (or Oracle Attacks)**
  - Selection of Data Blocks to be processed by the Private Key for the purpose of cryptanalysis
  - These attacks must be avoided using Strong Padding Schemes
  - Also relevant to avoid the "low exponentiation problems": large blocks and large keys

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), 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

Outline

- **Asymmetric cryptography**
  - Public-Key cryptography principles
  - Public-Key algorithms
  - RSA algorithm
    - Key-Pair Generation and Encryption/Decryption
  - Diffie-Hellman key exchange

  - DSA
  - ECC

Diffie-Hellman Key Exchange

- First public-key type scheme:
  - Diffie & Hellman in 1976 along with the exposition of public key concepts

  New Directions in Cryptography, IEEE TRANSACTIONS ON INFORMATION THEORY, Vol IT 22, N. 6. Nov 1976, https://ee.stanford.edu/~hellman/publications/24.pdf

- Note: now know that Williamson (UK CESG) secretly proposed the concept in 1970
- Practical method for public exchange of a secret key *k* between two principals: *A* and *B*
  - Never exposing *k:* key generated independently by A and B
    - *Key-Agreement without key exposition*
  - No pre-shared secrets between A and B
  - PFS and PBS warranties
  - Can be extended to groups of proncipals (A,B,C.D, …. etc)
- Used in many security standard protocols and today in several commercial products

Diffie-Hellman Key Exchange

- It 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 for the generation of key session
- Value of key depends on (and only on) the participants (and their private and public DH parameters)
  - D-H Private and Public Numbers + Initial (non-secret) setup parameters

Diffie-Hellman Security and Math Behind

- Based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial)

  Easy to compute ! (computationally feasible)

- Security relies on the difficulty of computing discrete logarithms (similar to factoring)

  Hard to compute (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 **mod q** i.e., $b_q \equiv a^i \bmod q$
  - **a** is called the generator of a multiplicative group of integers modulo **q**
  - $a^i \bmod q$, where $0 <= i <= (q-1)$ **generates all the integers between** 1 **and** q-1, in some permutation order
  - For any integer **b** < **q** 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 (mod q)

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

Diffie-Hellman: foundations (2)

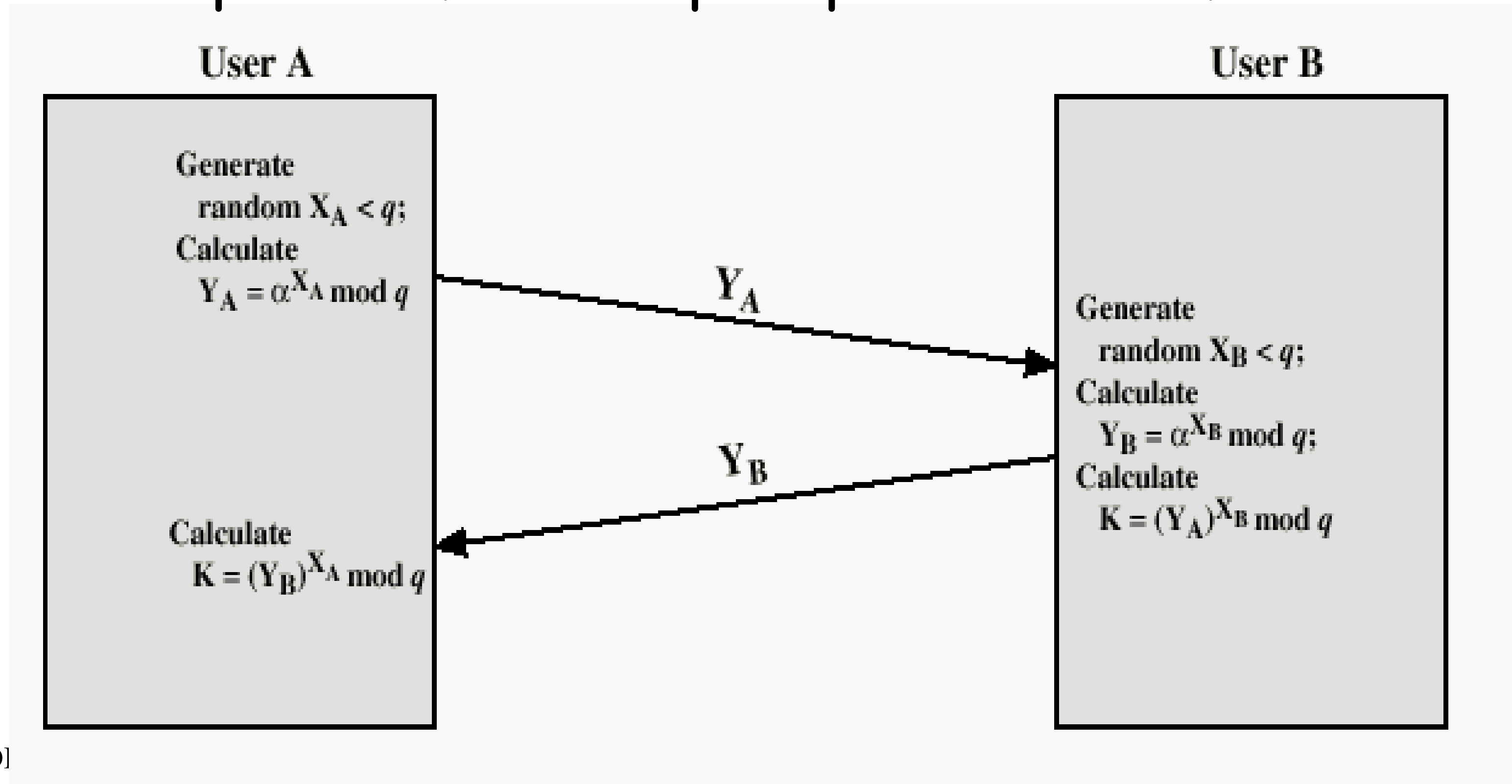
- Considering i the discrete logarithm for which:
  $a^i \bmod q = b$, taking **a** and **q** (as known parameters)
  - It is simple to calculate **b**, knowing **i**
  - It is very hard to calculate **i** only knowing **b**, **a** and **p**
  - This implies the computation of the discrete logarithm: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

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

$$a^{R} \bmod q = a^{R1.R2} \bmod q$$
$$= (a^{R1} \bmod q) \; (a^{R2} \bmod q)$$
$$= (a^{R1} \bmod q)^{R2} \bmod q$$

Diffie-Hellman Setup and Agreement

- If A and B share the global parameters **a** and **q**, being **a** a primitive root modulo q
- A and B generate their (private, public) pairs:
  - 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}:

$$K_{AB} = a^{x_A . x_B} \bmod q$$
$$= {y_A}^{x_B} \bmod q \quad \text{(which } \mathbf{B} \text{ can compute)}$$
$$= {y_B}^{x_A} \bmod q \quad \text{(which } \mathbf{A} \text{ can compute)}$$

- 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

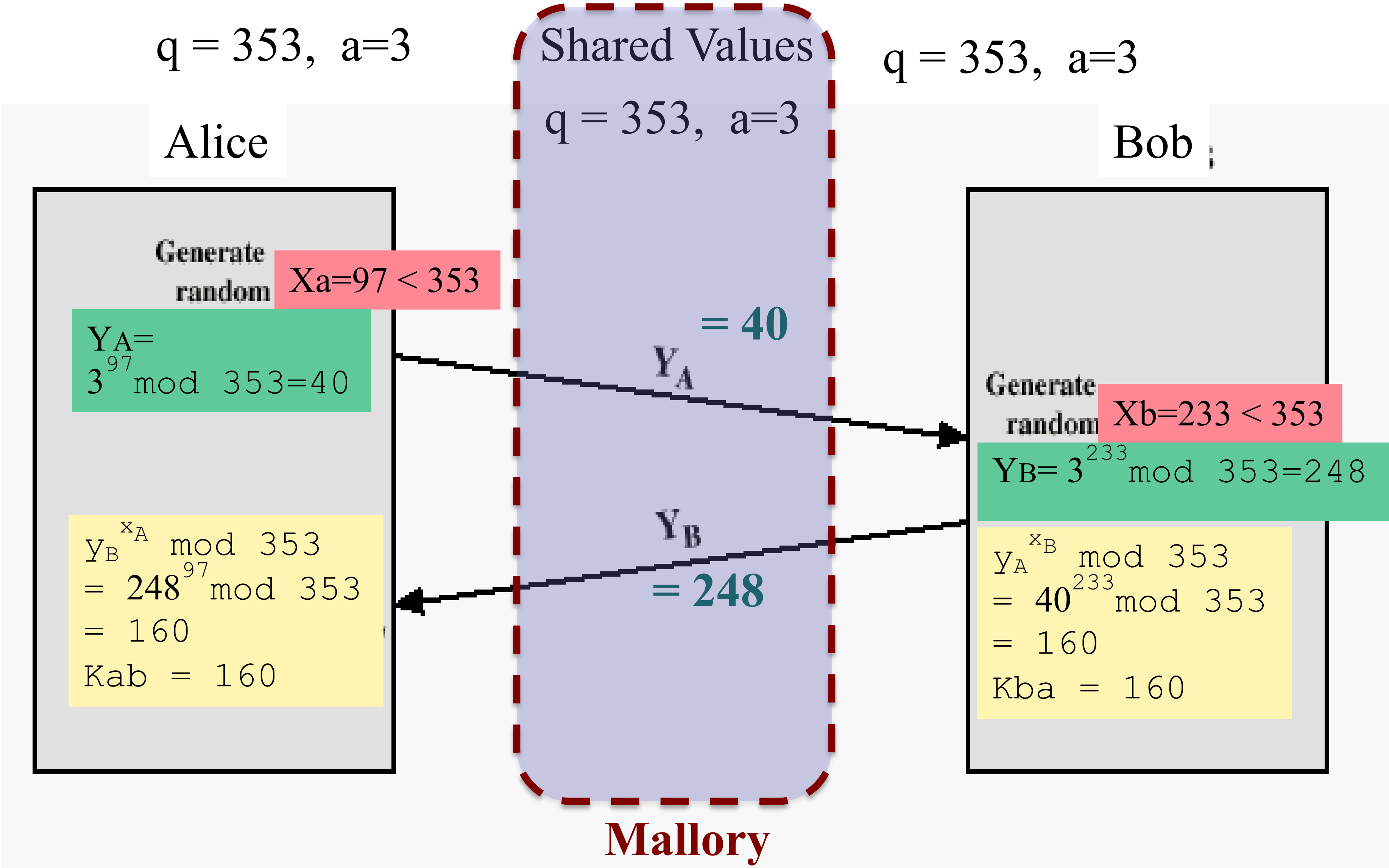
- Note) It is possible to make generalized D-H agreements, extended to a group of N

Diffie-Hellman Example

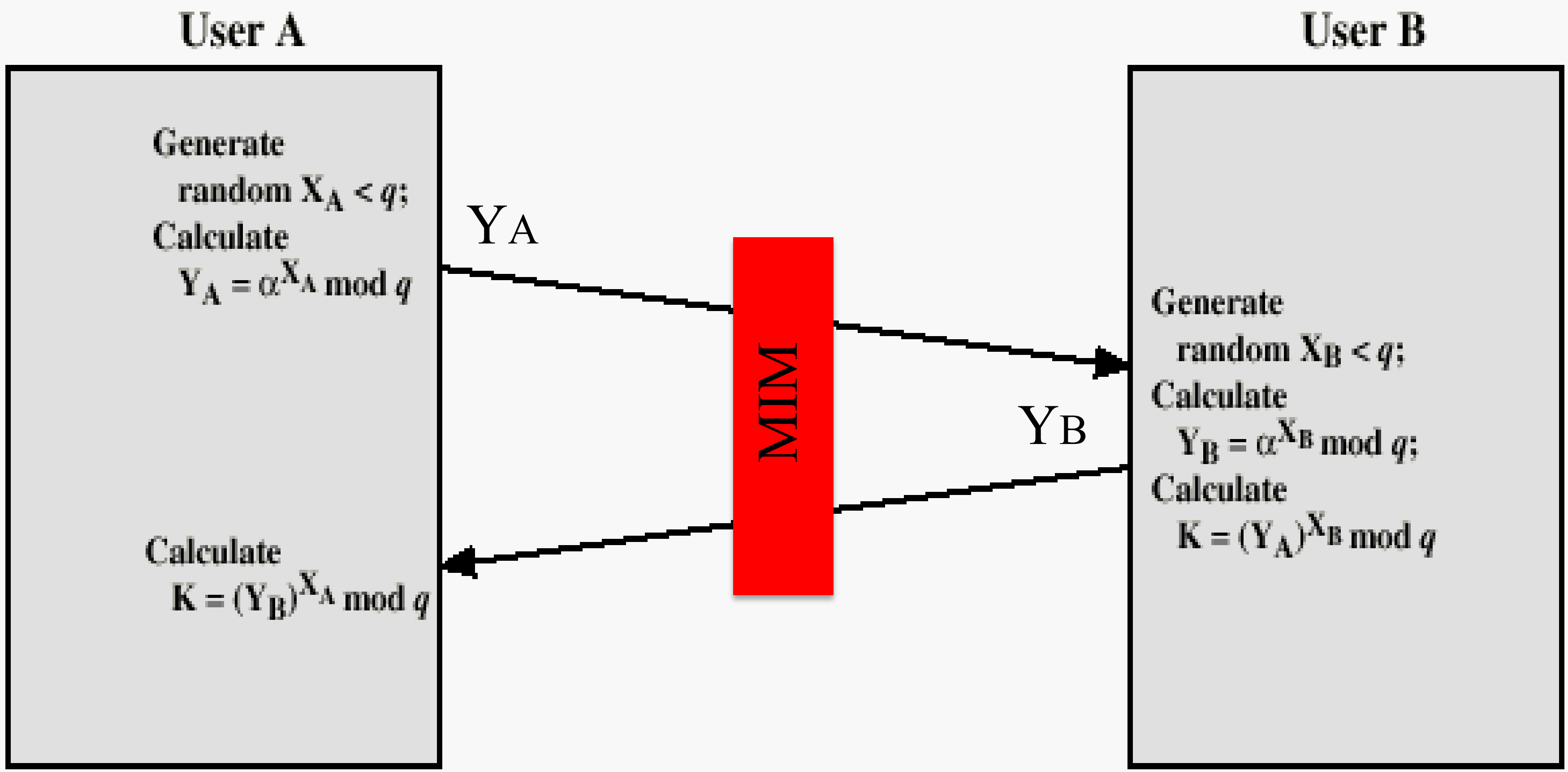
- Users Alice & Bob who wish to swap keys:
- Ex., agree on prime $q=353$ and $a=3$
- Select random secret numbers:
  - A chooses $x_A=97$, B chooses $x_B=233$
- Compute respective the public numbers:
  - $y_A=3^{97} \bmod 353 = 40$ (Alice)
  - $y_B=3^{233} \bmod 353 = 248$ (Bob)
- Compute shared session 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 Echange (example)



So Far so good ! But what if there is a MiM Attack?



$K_A = Y_B \wedge X_A \bmod q$

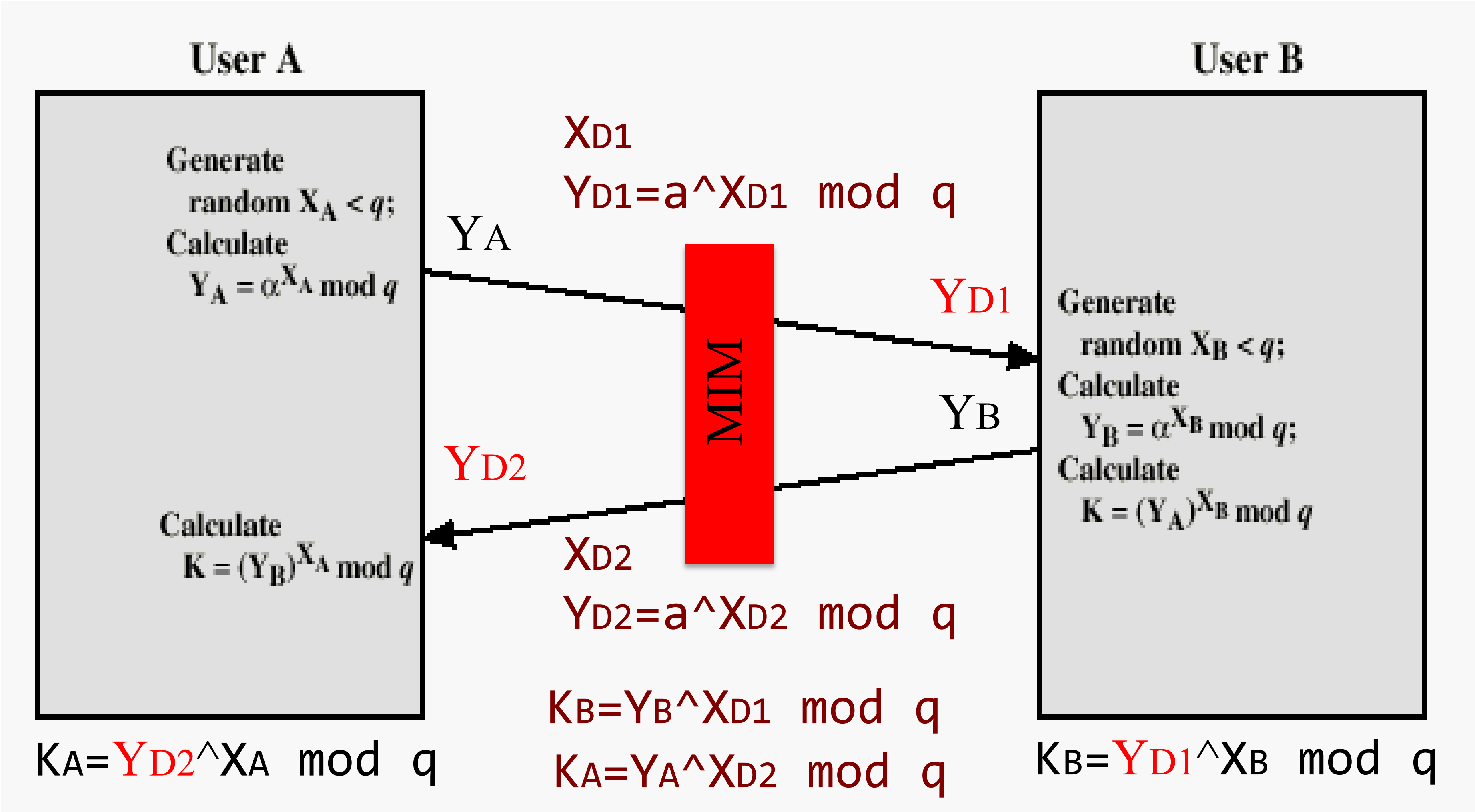
$K_B = Y_A \wedge X_B \bmod q$

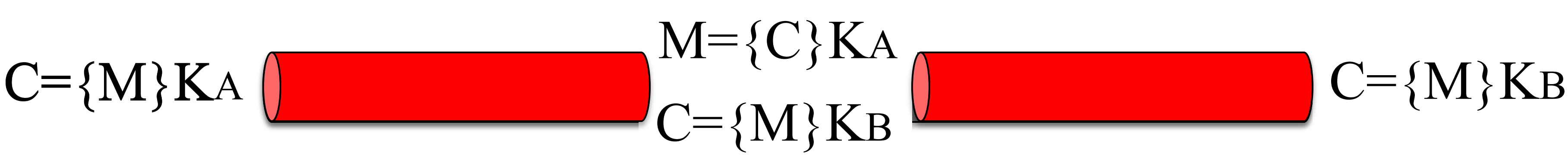
$K_A = K_B = K$

$C=\{M\}_K$

$M=\{C\}'_K$

D-H with a MIM Attack





The DH 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

- Principle (in the DH-agreement):

Alice sends to Bob $Sign_{KprivA}(Ya)$

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

Bob sends to Alice $Sign_{KprivB}(Ya)$

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

Authenticated Key-Exchange using Digital Signatures

- We can use a public-key (asymmetric) method to support digital signatures to authenticate public Diffie-Hellman public numbers
  - Exampled: RSA Signatures, DSA Signatures, ECC-DSA Signatures etc…

- After the authenticated D-H exchange, the session key must be established independently by the principals involved
  - No problem with seed materials passing in the channel (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 ?
- Group-Diffie Hellman Schemes
  - We will see this in action later, in a demo implementation 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)
  - Key point is the generation of private numbers with no secure random generators
  - Avoidance of using repeated DH numbers: trade-off between security, performance and usability

Security of D-H

State-of art (The best Discrete Logarithm Algorithms, ex., Number Field Sieve): Complexity => computational impossibility

- Today: DH numbers of 2048, …. 3072 bits !
- Recommendation: signatures w/ ECDH, using a group generator for P at least w/ 2048 bits

- Generation process for <public, private> DH numbers can be hard (harder for big modulus and big prime number generation/verification)
  - In practice, cam use pre-generated parameters ! (Pre-selected parameters in standard protocols)

Diffie-Hellman Agreements: Practical verifications

Example: DH using openssl (1)

Need Global Parameters (G, Prime)

openssl genpkey -genparam -algorithm DH -out dhp.pem

Remember, these PUBLIC Global Parameters (no problem to be known by anybody), that Alice and Bob will be shared for the DH Agreement

Now Alice and Bob will generate theitr own pairs <private, public>

Alice:

openssl genpkey -paramfile dhp.pem -out dhkey2.pem

Bob:

openssl pkey -in dhkey2.pem -text -noout

Example: DH using openssl (2)

Now will extract the public numbers

Alice:

openssl pkey -in dhkey1.pem -pubout -out dhpub1.pem

Public Nr from Alice:

openssl pkey -pubin -in dhpub1.pem -text

Bob

openssl pkey -in dhkey2.pem -pubout -out dhpub2.pem

Public Nr from Alice:

openssl pkey -pubin -in dhpub2.pem -text

Example: DH Agreement

- Given the public numbers echanged ... Can compute the shared key:

Alice:

openssl pkeyutl -derive -inkey dhkey1.pem -peerkey dhpub2.pem -out secret1.bin

Bob:

openssl pkeyutl -derive -inkey dhkey2.pem -peerkey dhpub1.pem -out secret1.bin

See the both independent computations:

cmp secret1.bin secret2.bin   (or diff)

See what is inside with od (octal dump)
or xxd (hexadecimal dump)

Example: DH using openssl (Size Impact)

```
Generation of public parameters today
(In this case we generate a prime w/ different bit sizes)
```

Command	Time
openssl dhparam -out dhparams.pem 256	Tens of ms (*)
openssl dhparam -out dhparams.pem 512	hund. ms to some sec.
openssl dhparam -out dhparams.pem 1024	Tens of sec.
openssl dhparam -out dhparams.pem 2048	Some-Tens of Minutes
openssl dhparam -out dhparams.pem 4096	☹ (((

….

What is the lesson leaned here ?

(*) MAC Book Pro (Late 2013) Intel Core i7, 2,3GHz
Openssl running on Mac OS Mojave 10.4

In Labs

- **We will see also how to program w/ DH primitives (Java /JCE) in Lab:**
  - Two Way DH Agreement
  - How to generalize to 3, 4 … N particioants
- Will see also ECDH Agreements

Outline

- **Asymmetric cryptography**
  - Public-Key cryptography principles
  - Public-Key algorithms
  - RSA algorithm
    - Key-Pair Generation and Encryption/Decryption
  - Diffie-Hellman key exchange

  - DSA
  - ECC

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 Gammal Crypto. but specifcally targetd for digital signatures only : similar to El Gammal 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 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 $g^q = 1 \bmod p$

So we have these shared parameters: **p**, **q** and **g**

DSA Security Conditions

Decisions on the key length *L* and *N*. This is the primary measure of the criptographic 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 recommendation for lengths of 2,048 (or 3,072) for keys with security lifetimes extending beyond 2010 (or 2030), using correspondingly longer *N*

FIPS 186-3 specifies *L* and *N* length pairs of (1,024, 160), (2,048, 224), (2,048, 256), and (3,072, 256).

N must be less than or equal to the output length of the hash *H*.

DSA Keys

Key pair (Kpriv, Kpub)

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

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 $s = k^{-1} (H(M) + xr) \bmod 1$

   if s = 0, regenerate the random k

4. If s !=0 => the Sig(M) = (r, s)

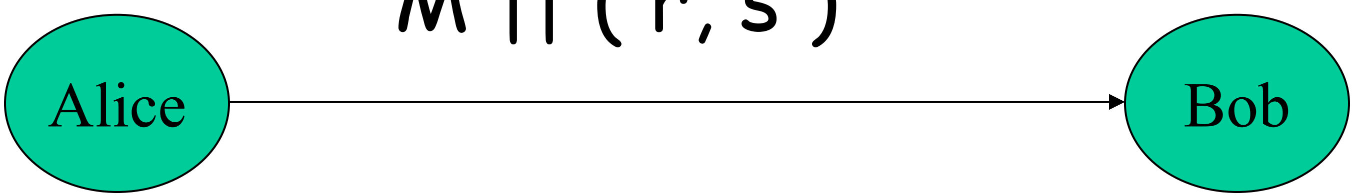
… So we need initial parameters: **p**, **q** and **g**

DSA Parameters involved

Initial
Shared Parameters: **p, q, g**

$K_{pubAlice}$

$M \,||\, (r, s)$



DSA Signature Construction

Validation

DSA Signature Verification

Received M, (r,s) … and knowing p, k, g and KpubAlice

1. We must reject a signature

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

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

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

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

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

6. If $v = r$ signature is valid ! Otherwise not valid !

DSA Practical Observations

- DSA Signature Verification tend to be slowly compared with RSA, Signatures tend to be faster
- Sizes of signatures are shorter (and may have variable sizes)
  - Can see this effect in LABs
  - In RSA, the signature size is 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 byres for bitstrng encoding. So you can expect: 47 to 51 bytes
- In general, the DSA "keypair" generation process is faster than RSA (keys w/ same size)

Ex: openssl benchmark (Sign vs. Verif)

```
                  sign    verify    sign/s verify/s
rsa  512 bits 0.000836s 0.000083s   1196.7  12104.4
rsa 1024 bits 0.004916s 0.000405s    203.4   2468.7
rsa 2048 bits 0.033003s 0.001584s     30.3    631.1
rsa 4096 bits 0.221087s 0.005828s      4.5    171.6
                  sign    verify    sign/s verify/s
dsa  512 bits 0.000790s 0.000878s   1265.5   1138.4
dsa 1024 bits 0.002693s 0.003040s    371.3    329.0
dsa 2048 bits 0.009653s 0.010966s    103.6     91.2
```

Outline

- **Asymmetric cryptography**
  - Public-Key cryptography principles
  - Public-Key algorithms
  - RSA algorithm
    - Key-Pair Generation and Encryption/Decryption
  - Diffie-Hellman key exchange

  - DSA
  - ECC

ECC: Elliptic Curve Cryptography

- Not one … But many Elliptic Curves !
- Different Curves => Different levels of security and => Different computation complexity

- Elliptic Curve (EC) systems as applied to cryptography: first proposed in 1985 independently by Neal Koblitz and Victor Miller.

- The discrete logarithm problem on elliptic curve groups is believed to be more difficult than the corresponding problem in (the multiplicative group of nonzero elements of) the underlying finite field.

Definition of Elliptic curves

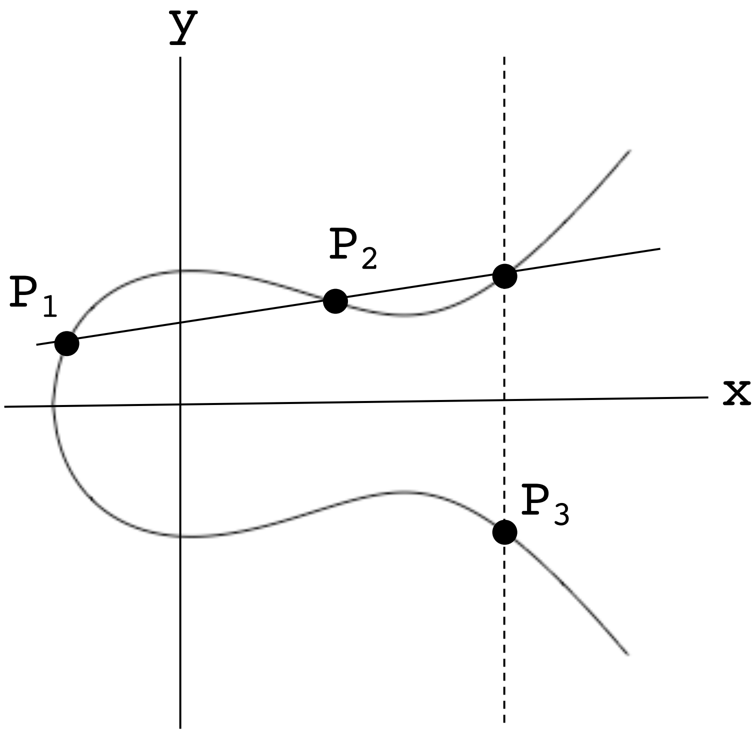
- An elliptic curve over a field *K* is a **nonsingular cubic curve in two variables, *f(x,y) =0* with a rational point (which may be a point at infinity).**
- The field *K* is usually taken to be the complex numbers, reals, rationals, algebraic extensions of rationals, p-adic numbers, or a **finite field**.
  - ABELIAN Groups
- Elliptic curves groups for cryptography are examined with the underlying fields of F_p (*where p>3 is a prime*) and F_{2^m} (*a binary representation with 2^m elements*).

Abelian Groups

Given two points P,Q in *E(Fp)*, there is a third point, denoted by $P+Q$ on *E(Fp)*, and the following relations hold for all P,Q,R in *E(Fp)*

- $P + Q = Q + P$ (*commutativity*)
- $(P + Q) + R = P + (Q + R)$ (*associativity*)
- $P + O = O + P = P$ (*existence of an identity element*)
- there exists $(-P)$ such that $-P + P = P + (-P) = O$ (*existence of inverses*)

Elliptic Curve Picture



- Consider elliptic curve

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

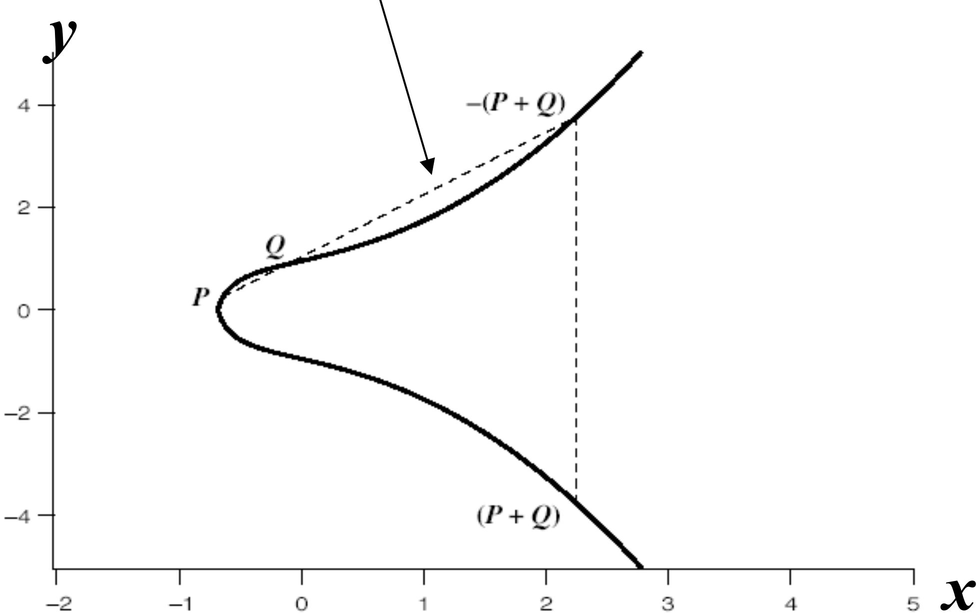
- If P_1 and P_2 are on E, we can define

  $$P_3 = P_1 + P_2$$

  as shown in picture
- Addition is all we need

Addition in Affine Co-ordinates

$y=m(x-x_1)+y_1$



$y^2=x^3+Ax+B$

$$P=(x_1,y_1),Q=(x_2,y_2)$$
$$R=(P+Q)=(x_3,y_3)$$

Let, $P \neq Q$,

Doubling of a point

- Let, P=Q

$$2y\frac{dy}{dx} = 3x^2 + A$$

$$\Rightarrow m = \frac{dy}{dx} = \frac{3x_1^2 + A}{2y_1}$$

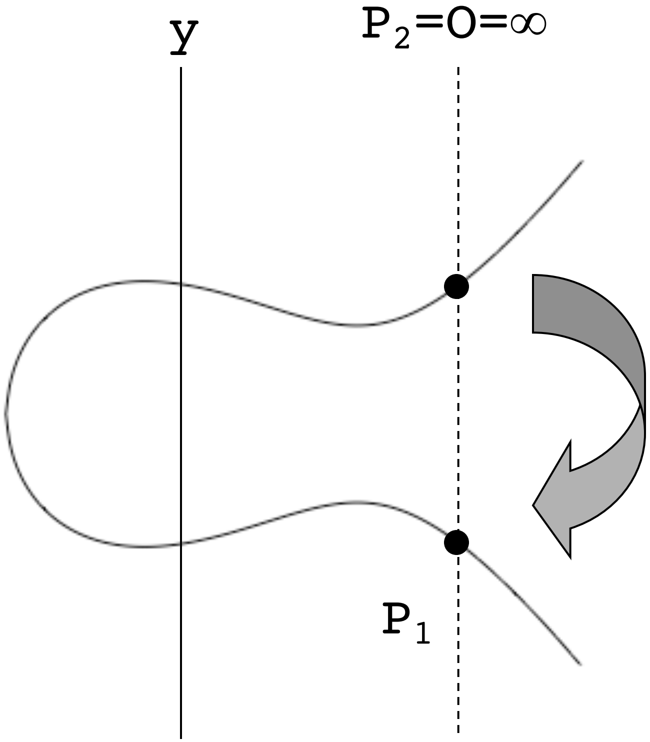
$$If,\ y_1 \neq 0 \text{ (since then } P_1+P_2=\infty\text{):}$$

$$\therefore 0 = x^3 - m^2x^2 + \ldots$$

$$\Rightarrow x_3 = m^2 - 2x_1,\ y_3 = m(x_1 - x_3) - y_1$$

- What happens when $P_2=\infty$?

Why do we need the reflection?



$P_1 = P_1 + O = P_1$

What Is Elliptic Curve Cryptography (ECC)?

- Elliptic curve cryptography [ECC] is a **public-key** cryptosystem just like RSA, Rabin, and El Gamal.
- Every user has a **public** and a **private** key.
  - Public key is used for encryption/signature verification.
  - Private key is used for decryption/signature generation.
- Elliptic curves are used as an extension to other current cryptosystems.
  - Elliptic Curve Diffie-Hellman Key Exchange
  - Elliptic Curve Digital Signature Algorithm

Using Elliptic Curves In Cryptography

- The central part of any cryptosystem involving elliptic curves is the **elliptic group**.
- All public-key cryptosystems have some underlying mathematical operation.
  - RSA has exponentiation (raising the message or ciphertext to the public or private values)
  - ECC has point multiplication (repeated addition of two points).

Generic Procedures of ECC

- Both parties agree to some publicly-known data items
  - The **elliptic curve equation**
    - values of *a* and *b*
    - prime, *p*
  - The **elliptic group** computed from the elliptic curve equation
  - A **base point**, B, taken from the elliptic group
    - Similar to the generator used in current cryptosystems
- Each user generates their public/private key pair
  - Private Key = an integer, x, selected from the interval [1, p-1]
  - Public Key = product, Q, of private key and base point
    - (Q = x*B)

Operations in ECCs

- After that we can model and implement any other conventional operation (as in DSA, DH or RSA) with additions and multiplicatioons and modular constructions

Why use ECC?

- How do we analyze Cryptosystems?
  - How difficult is the underlying problem that it is based upon
    - RSA – Integer Factorization
    - DH – Discrete Logarithms
    - ECC - Elliptic Curve Discrete Logarithm problem
  - How do we measure difficulty?
    - We examine the algorithms used to solve these problems

Security of ECC

- To **protect** a 128 bit AES key it would take a:
  - RSA Key Size: 3072 bits
  - ECC Key Size: 256 bits
- How do we strengthen RSA?
  - Increase the key length
- **Impractical?**

NIST guidelines for public key sizes for AES

ECC KEY SIZE (Bits)	RSA KEY SIZE (Bits)	KEY SIZE RATIO	AES KEY SIZE (Bits)
163	1024	1 : 6	
256	3072	1 : 12	128
384	7680	1 : 20	192
512	15 360	1 : 30	256

Supplied by NIST to ANSI X9F1

Applications of ECC

- Many devices are small, with limited resources (store, computational power and energy)
- Where can we apply ECC?
  - **Wireless communication devices**
  - **Edge computing devices**
  - **Smart cards, Smart tokens**
  - **Mobile phonee, avoiding energy, stiorage anc computatioal costs**
  - **Web servers that need to handle many session-contexts (very high scale-in vs high levels of concurrency)**

  - **Any application where security is needed but lacks the power, storage and computational power that is necessary for our current cryptosystems**

Benefits of ECC

- Same benefits of the other cryptosystems: confidentiality, integrity, authentication and non-repudiation but…
- Shorter key lengths
  - Encryption, Decryption and Signature Verification speed up
  - Storage and bandwidth savings

Summary of ECC

- "Hard problem" analogous to discrete log
  - `Q=kP`, where `Q,P` belong to a prime curve
    given `k,P` → "easy" to compute `Q`
    given `Q,P` → "hard" to find `k`
  - known as the elliptic curve logarithm problem
    - `k` must be large enough
- ECC security relies on elliptic curve logarithm problem
  - compared to factoring, can use much smaller key sizes than with RSA etc
    → for similar security ECC can offer significant computational advantages

Some ECC Concerns

- *Political concerns*: the trustworthiness of NIST - produced curves being questioned after revelations that the NSA willingly inserts backdoors into software, hardware components and published standards were made;
  - well-known respectable cryptographers have expressed doubts about how the NIST curves were designed, and voluntary tainting has already been proved in the past.

- *Technical concerns*: the difficulty to properly implement the standard and the slowness and design flaws which reduce security in insufficiently precautions implementations on random number generations

Readings

- William Stallings, Network Security Essentials, 4rd Edition, 2011, Part One – Cryptography, Chap.3

Fore more detail:

- William Stallings, W. *Cryptography and Network Security: Principles and Practice, Chap. 9,* Pearson - Prentice Hall, *7th Ed. , 2017*

- More (for complementary interests)

  Bruce Schneier, *Applied Cryptography*, New York: Wiley, 1996, Chap.