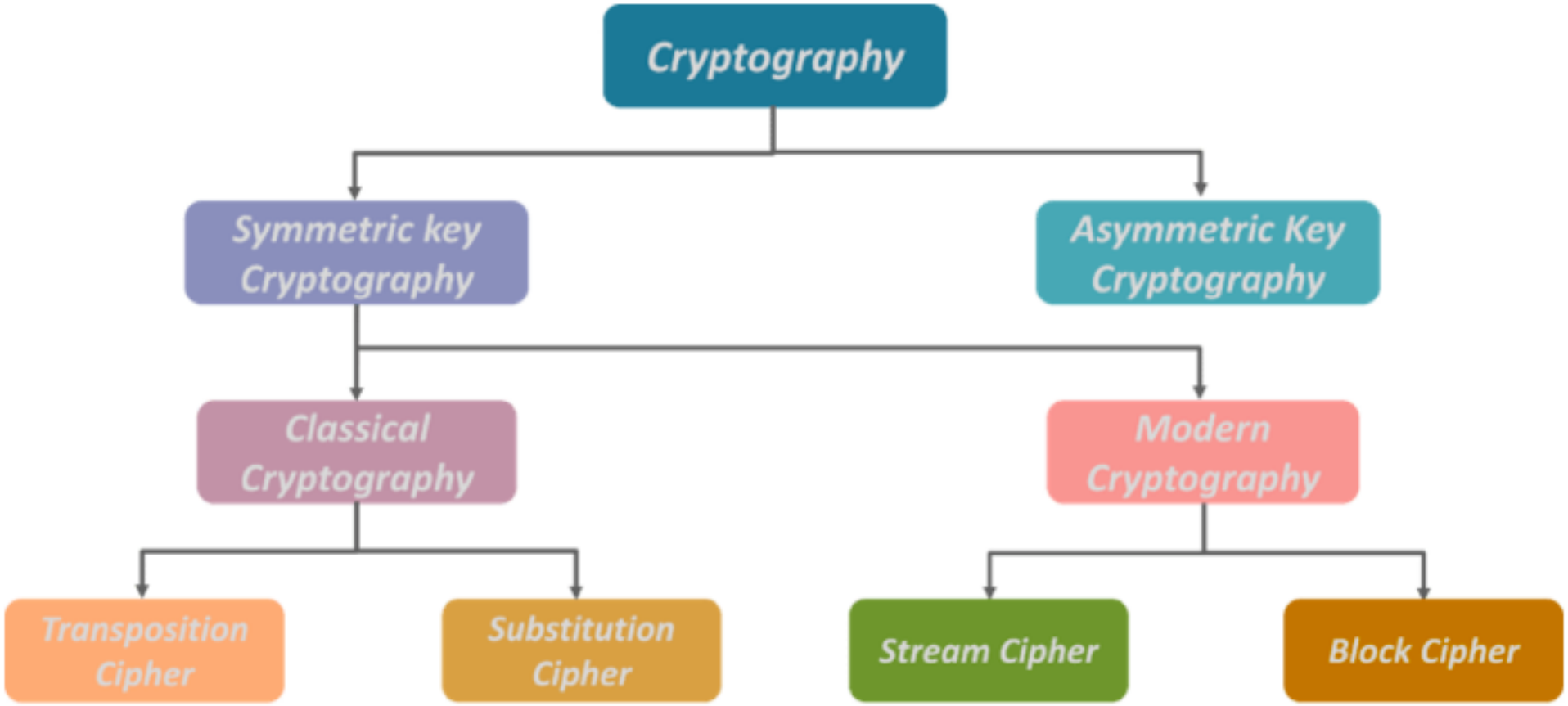
Practical Cryptography

Handout 6 – Public Key Cryptography

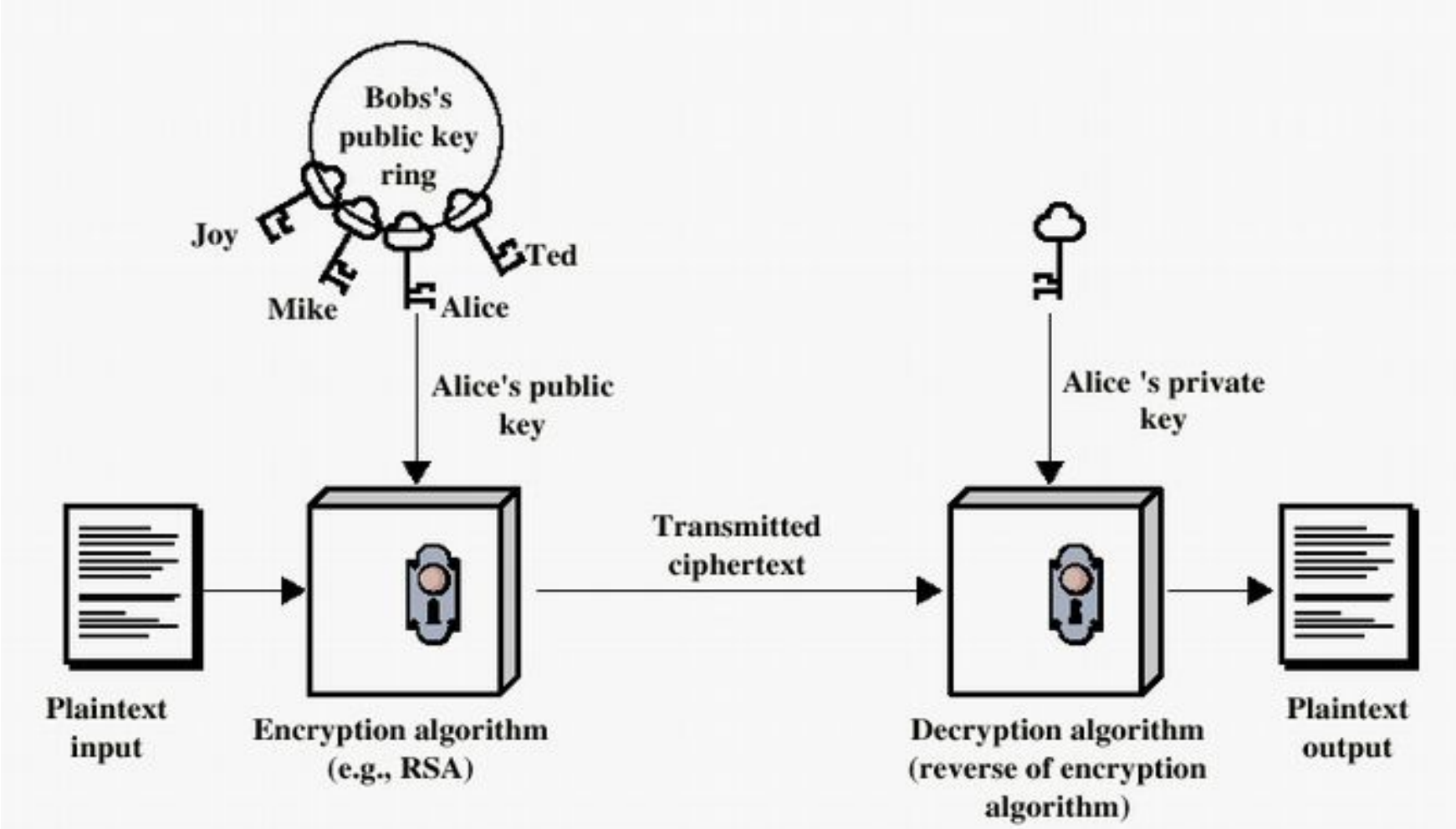
Kasun de Zoysa
kasun@ucsc.cmb.ac.lk

UNIVERSITY OF COLOMBO SCHOOL OF COMPUTING

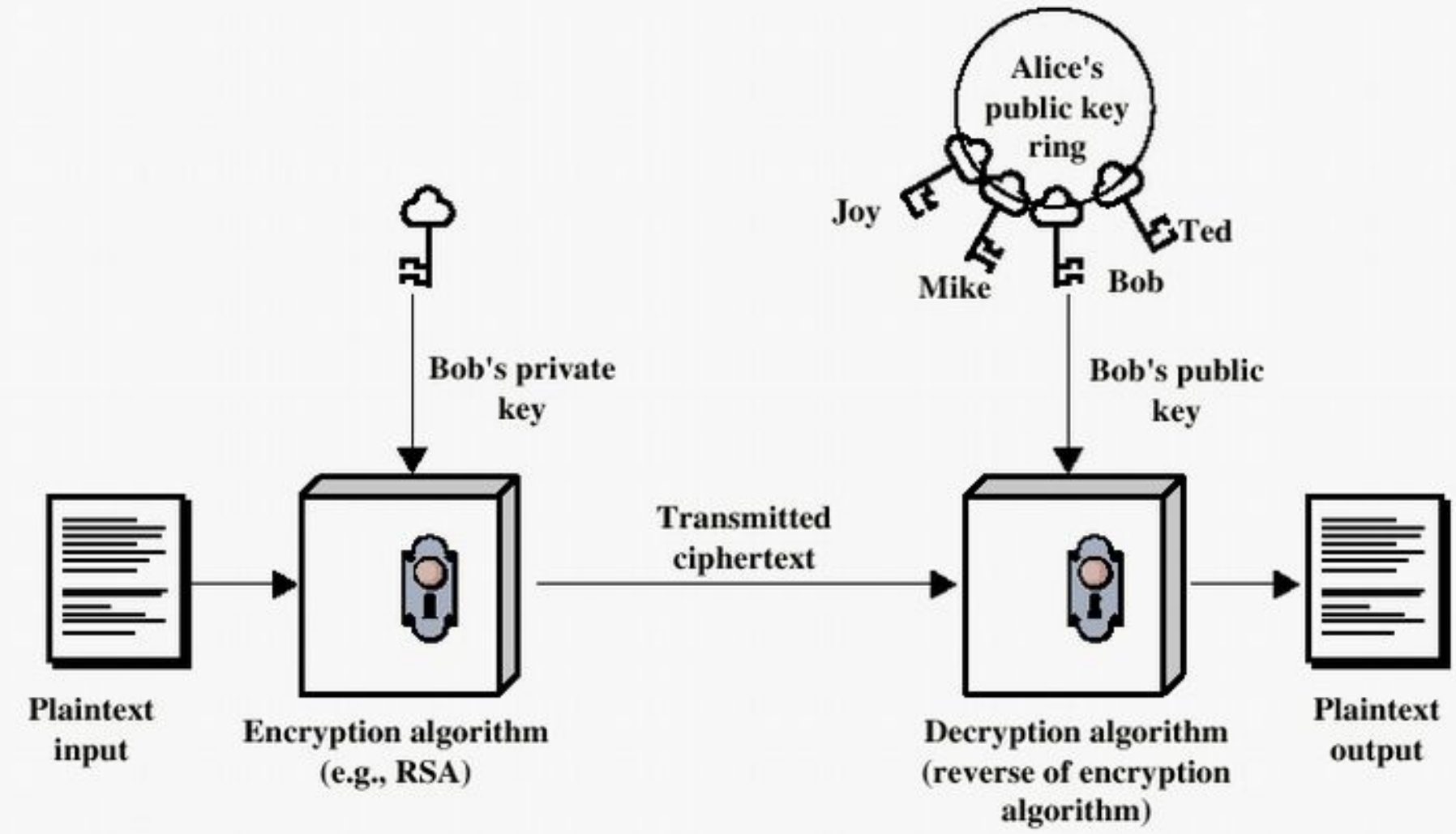
Cryptography



Encryption using Public-Key system



Authentication using Public-Key System



Applications for Public-Key Cryptosystems

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

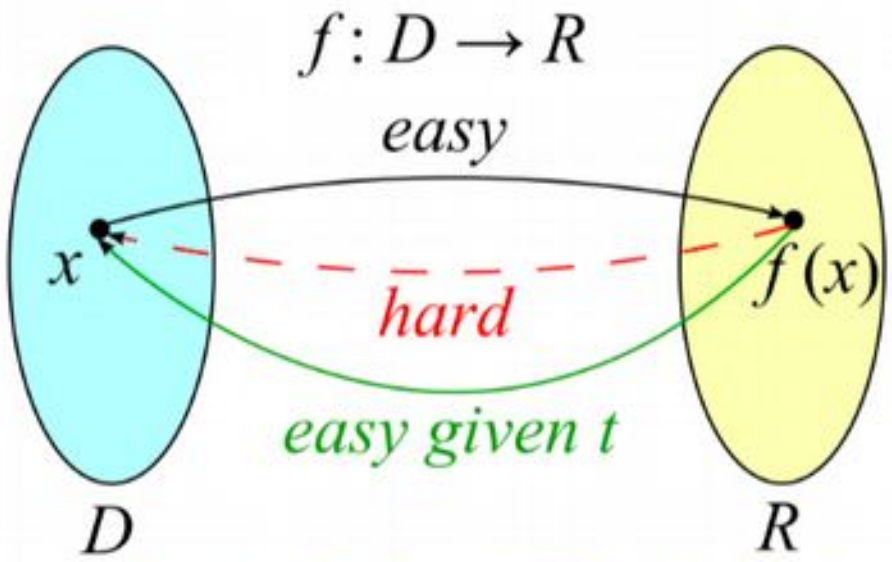
Public-Key Cryptographic Algorithms

- **Diffie-Hellman**
  - Exchange a secret key securely
  - Compute discrete logarithms
- **RSA** - Ron Rives, Adi Shamir and Len Adleman at MIT, in 1977.
  - The most widely implemented
- **Elliptic Curve Cryptography (ECC)**

Trapdoor Function

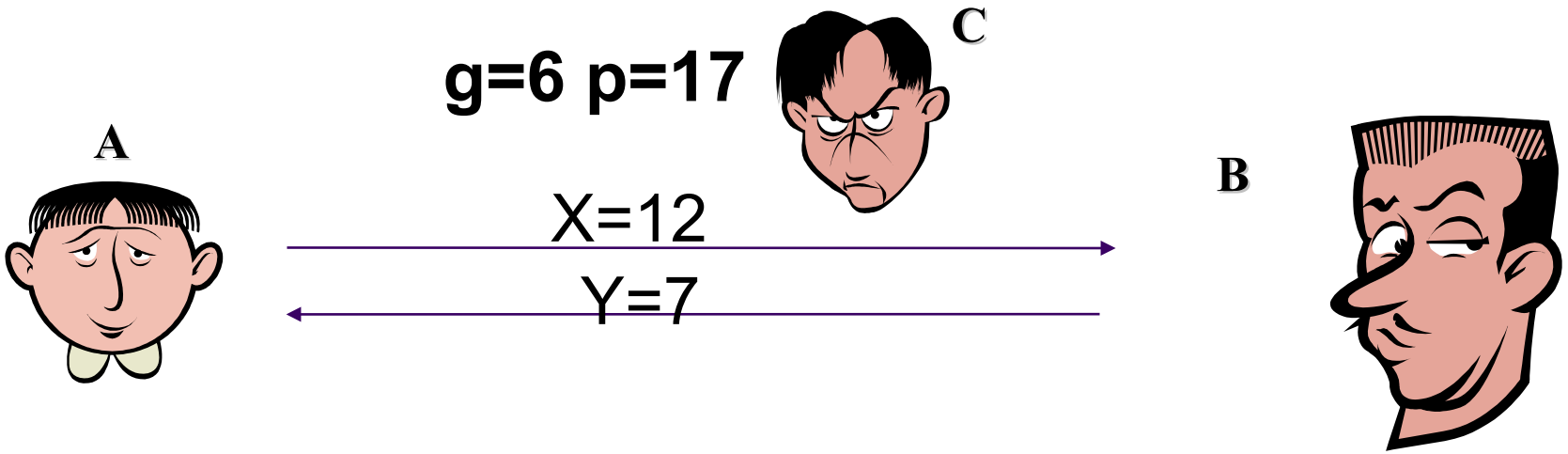
Trapdoor functions

- Easy to compute in one direction
- Difficult to compute in other direction (finding the inverse)
  but easy to compute, with some special information (trapdoor)



Source: https://en.wikipedia.org/wiki/Trapdoor_function

Diffie-Hellman Key Agreement



$X=g^x \bmod p$
$k=Y^x \bmod p = g^{xy} \bmod p$

Alice picks x=3
Alice's X = 6^3 mod 17
= 216 mod 17 = 12

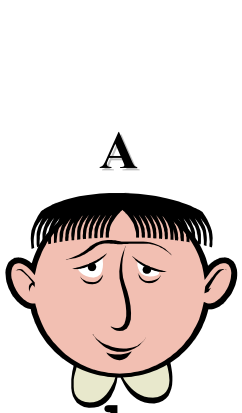
Alice's k = 7^3 mod 17
= 243 mod 17 = 3

$Y=g^y \bmod p$
$k=X^y \bmod p = g^{xy} \bmod p$

Bob picks y=5
Bob's Y = 6^5 mod 17
= 7776 mod 17 = 7

Bob's k = 12^5 mod 17
= 248832 mod 17 = 3

Attacks on Diffie-Hellman Key Agreement



X

Y

$X=g^x \bmod n$

$k=Y^x \bmod n = g^{xy} \bmod n$

$Y=g^y \bmod n$

$k=X^y \bmod n = g^{xy} \bmod n$

Possible to do man in the middle attack

- You really don't know anything about who you have exchanged keys with
- Alice and Bob think they are talking directly to each other, but Caldera is actually performing two separate exchanges
- **You need to have an authenticated DH exchange**

RSA

The Association for Computing Machinery (ACM) has named RONALD L. RIVEST, ADI SHAMIR, and LEONARD M. ADLEMAN as winners of the 2002 A. M. Turing Award, considered the "Nobel Prize of Computing", for their contributions to public key cryptography. The Turing Award carries a $100,000 prize, with funding provided by Intel Corporation.

As researchers at the Massachusetts Institute of Technology in 1977, the team developed the RSA code, which has become the foundation for an entire generation of technology security products. It has also inspired important work in both theoretical computer science and mathematics. RSA is an algorithm—named for Rivest, Shamir, and Adleman—that uses number theory to provide a pragmatic approach to secure transactions. It is today's most widely used encryption method, with applications in Internet browsers and servers, electronic transactions in the credit card industry, and products providing email services.

- Excerpt from ACM news release on

2002 Turing award

Ron Rivest
born in 1947

Adi Shamir
born in 1952

Leonard M. Adleman
born in 1945

Revest-Shamir-Adelman (RSA)

By Rivest, Shamir and Adelman in 1978

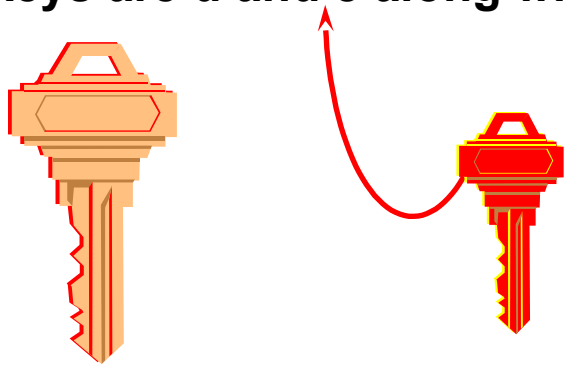
1. Find 2 large prime numbers p and q (100 digits=512bits)

2. Calculate the product n=p*q (n is around 200 digits)

3. Select large integer e relatively prime to (p-1)(q-1)
Relatively prime means e has no factors in common with (p-1)(q-1).
Easy way is select another prime that is larger than both(p-1) and (q-1).

4. Select d such that e*d mod (p-1)*(q-1)=1

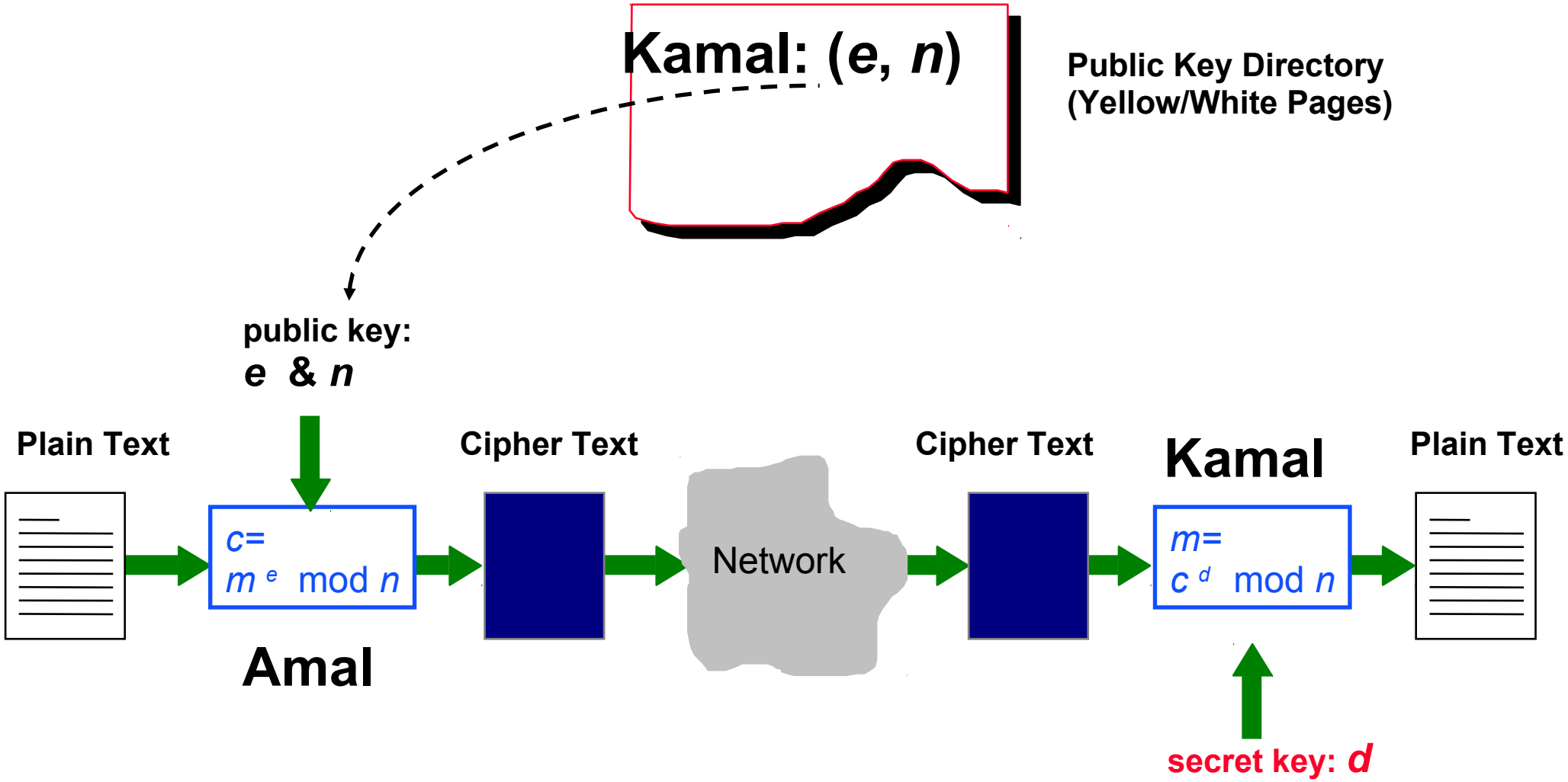
Two keys are d and e along with n



<u>Encryption</u>
$C=P^e \bmod n$

<u>Decryption</u>
$P=C^d \bmod n$

RSA Public Key Cryptosystem



RSA - Simple Example

1. **Find 2 prime numbers p and q**

   Let p=11 and q=17

2. **Calculate the product n=p*q**

   *n = 11*17=187*

3. **Select large integer e relatively prime to (p-1)(q-1)**

   *E=7 ; 7 IS Relatively prime to (p-1)(q-1) =10*16=160*

4. **Select d such that e*d mod (p-1)*(q-1)=1**

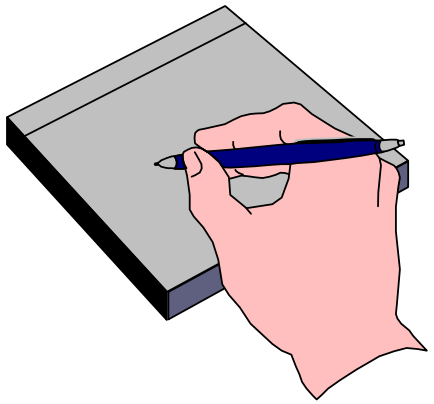
   *d=23 because, 23*7 mod 10*16=161 mod 160 =1*

Encryption

$C=P^e \bmod n$

Decryption

$P=C^d \bmod n$



RSA - Simple Example

recipient knows:

- PR={23,187} // d=23, n=187
- 187=17×11 // p=17, q=11
- ϕ(n)=(p-1)(q-1)=160 // check: 7×23 mod 160=1

sender knows:

- PU={7,187} // e=7, n=187
- plaintext to encrypt: M=88 // 88 < 187

RSA - Simple Example

sender knows:

- PU={7,187}
- plaintext to encrypt: M=88 // 88 < 187

ciphertext

Encryption

$$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187$$

$$88^1 \bmod 187 = 88$$

$$88^2 \bmod 187 = 7744 \bmod 187 = 77$$

$$88^4 \bmod 187 = 59{,}969{,}536 \bmod 187 = 132$$

$$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894{,}432 \bmod 187 = 11$$

RSA - Simple Example

recipient knows:

- PR={23,187}
- 187=17×11 // p=17, q=11
- $\phi(n)=(p-1)(q-1)=160$ // check: 7×23 mod 160=1
- receives cipher text: 11

Decryption

$$11^{23} \bmod 187 = [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$$

$$11^1 \bmod 187 = 11$$

$$11^2 \bmod 187 = 121$$

$$11^4 \bmod 187 = 14{,}641 \bmod 187 = 55$$

$$11^8 \bmod 187 = 214{,}358{,}881 \bmod 187 = 33$$

$$11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187$$
$$= 79{,}720{,}245 \bmod 187 = 88$$ ← plaintext

RSA --- 2nd small example (1)

- Kamal:
  - chooses 2 primes: *p=5, q=11*
    multiplies p and q: *n = p*q = 55*
  - finds out two numbers *e=3* & *d=27* which satisfy
    (*3 * 27*) *mod 40 = 1*
  - Kamal's public key
    - 2 numbers: (*3, 55*)
    - encryption alg: modular exponentiation
  - secret key: (*27,55*)

RSA --- 2nd small example (2)

- Amal has a message $m=13$ to be sent to Kamal:
  - finds out Kamal's public encryption key $(3, 55)$
  - calculates c:

$$c = m^e \pmod n$$
$$= 13^3 \pmod{55}$$
$$= 2197 \pmod{55}$$
$$= 52$$

  - sends the ciphertext $c=52$ to Kamal

RSA --- 2nd small example (3)

- Kamal:
  - receives the ciphertext *c=52* from Amal
  - uses his matching secret decryption key *27* to calculate m:

$$m = 52^{27} \pmod{55}$$
$$= 13 \text{ (Amal's message)}$$

RSA --- 3rd small example

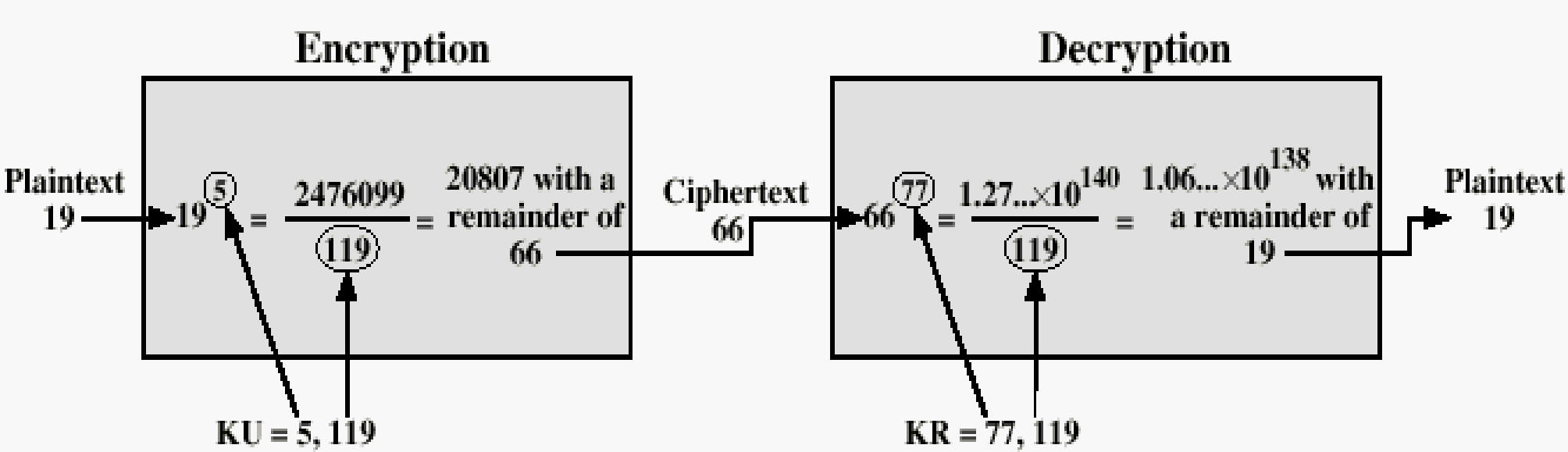


Figure 3.9 Example of RSA Algorithm

RSA Signature --- an eg (1)

- Kamal:
  - chooses 2 primes: *p=5, q=11*
    multiplies p and q: *n = p*q = 55*
  - finds out two numbers *e=3* & *d=27* which satisfy
    (*3 * 27*) *mod 40 = 1*
  - Kamal's public key
    - 2 numbers: (*3, 55*)
    - encryption algo: modular exponentiation
  - secret key: (*27,55)*

RSA Signature --- an eg (2)

- Kamal has a document *m=19* to sign:
  - uses his secret key *d=27* to calculate the digital signature of *m=19*:

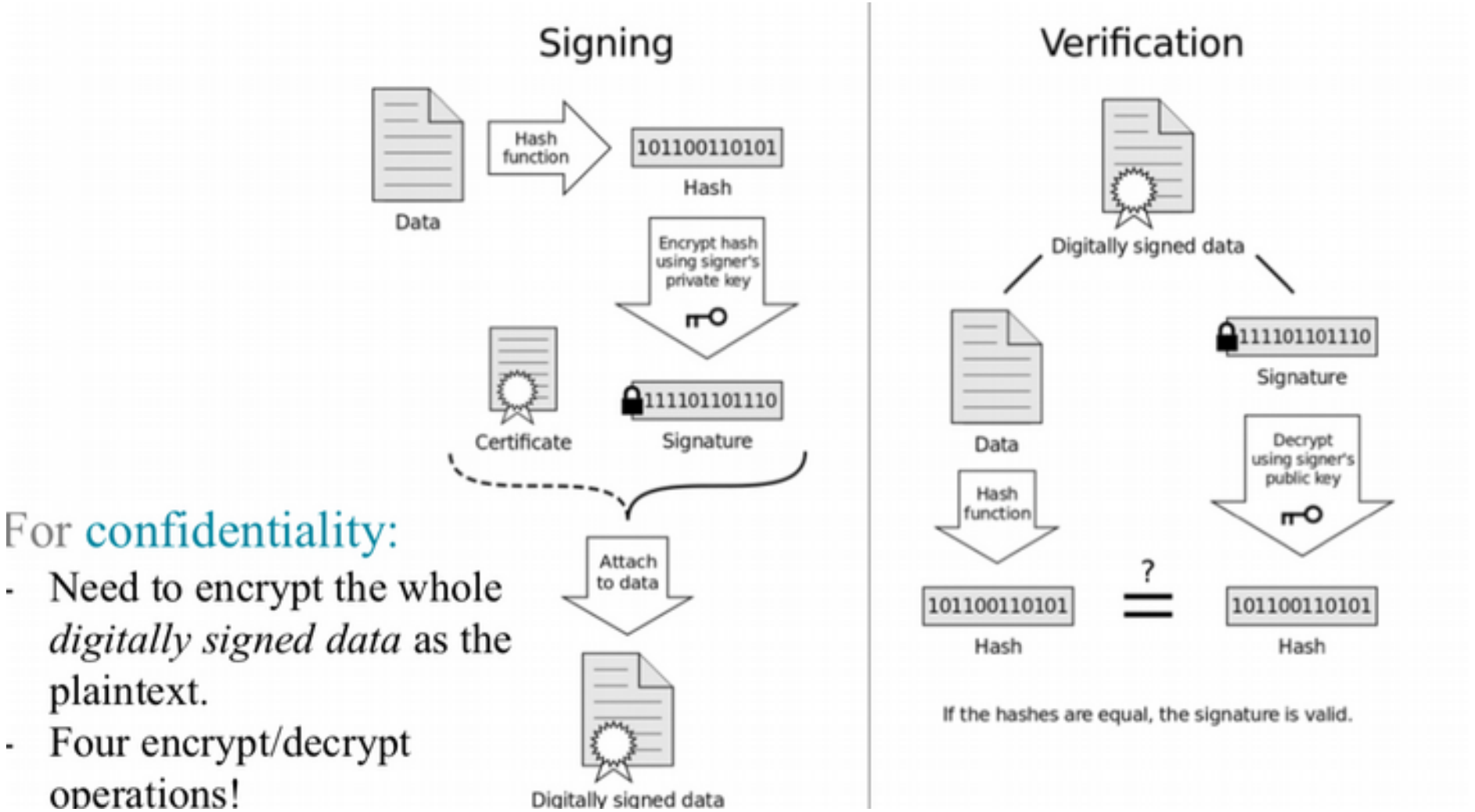
$$s = m^d \pmod{n}$$
$$= 19^{27} \pmod{55}$$
$$= 24$$

  - appends 24 to 19. Now *(m, s) = (19, 24)* indicates that the doc is 19, and Kamal's signature on the doc is 24.

RSA Signature --- an eg. (3)

- Nimal, a verifier:
  - receives a pair *(m,s)=(19, 24)*
  - looks up the phone book and finds out Kamal's public key *(e, n)=(3, 55)*
  - calculates $t = s^e \pmod n$
    $= 24^3 \pmod{55}$
    $= 19$
  - checks whether *t=m*
  - confirms that (*19,24*) is a genuinely signed document of Kamal if *t=m*.

Typical Digital Signature



For confidentiality:

- Need to encrypt the whole *digitally signed data* as the plaintext.
- Four encrypt/decrypt operations!

Factoring a product of two large primes

- **The best known conventional algorithm requires the solution time proportional to:**

$$T(n) = \exp\left[c(\ln n)^{1/3}(\ln\ln n)^{2/3}\right]$$

For p & q 65 digits long T(n) is approximately one month using cluster of workstations.

For p&q 200 digits long T(n) is astronomical.

Quantum Computing algorithm for factoring.

- **In 1994 Peter Shor from the AT&T Bell Laboratory showed that in principle a quantum computer could factor a very long product of primes in seconds.**
- **Shor's algorithm time computational complexity is**

$$T(n) = O\left[(\ln n)^3\right]$$

Once a quantum computer is built the RSA method would not be safe.

Signature Creation

- **Generate Public/Private key pair**

```
openssl genrsa -out mykey.pem
openssl rsa -in mykey.pem -pubout >mypub.pem
```

- **Create the signature**

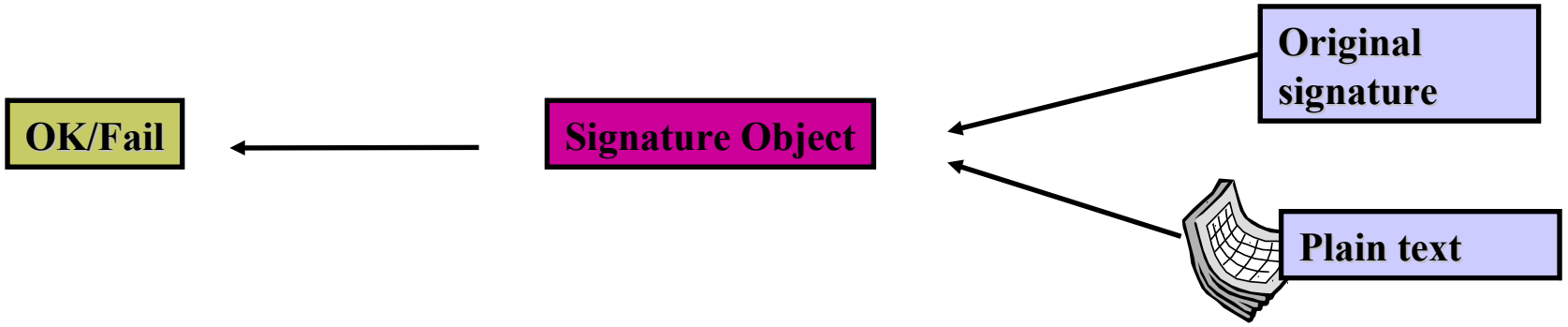
```
openssl dgst -sha1 -sign mykey.pem
-out mysign.sha1 jethavanaya.jpg
```

Signature Verification

- **Retrieves the Public key**

- **Verify the signature**

```
openssl dgst -sha1 -verify mypub.pem
-signature mysign.sha1 jethavanaya.jpg
```



Signature Creation

- **Generate Public/Private key pair**

```
KeyPairGenerator keyGen=
KeyPairGenerator.getInstance("DSA");
keyGen.initialize(1024,new SecureRandom());
KeyPair keyPair = keyGen.generateKeyPair();
```

- **Initialize the Signature object**

```
Signature signature= Signature.getInstance("SHA1withDSA");
signature.initSign(keyPair.getPrivate(),new SecureRandom());
```

- **Create the signature**

```
signature.update(msg.getBytes());
byte[]  sigBytes = signature.sign();
```

Signature Verification

- **Retrieves the Public key**

```
Let's say KeyPair object is keyPair
```

- **Initialize the Signature object**

```
Signature signature= Signature.getInstance("SHA1withDSA");
signature.initVerify(keyPair.getPublic());
```

- **Verify the signature**

```
Let's say sigBytes contains the original signature

signature.update(msg.getBytes());
signature.verify(sigBytes)
```

Elliptic Curve Cryptography (ECC)

ECC invented (independently):

- 1985
- wide-scale adoption circa 2005
  barrier to adoption: patent/license protections

Neal Koblitz
born in 1948

Victor S. Miller
born in 1947

Elliptic Curve

- An elliptic curve is the set of solutions to the equation

$$y^2 = x^3 + ax^2 + bx + c$$

- These solutions are not ellipses, the name elliptic is used for historical reasons and has do to with the integrals used when calculating arc length in ellipses:

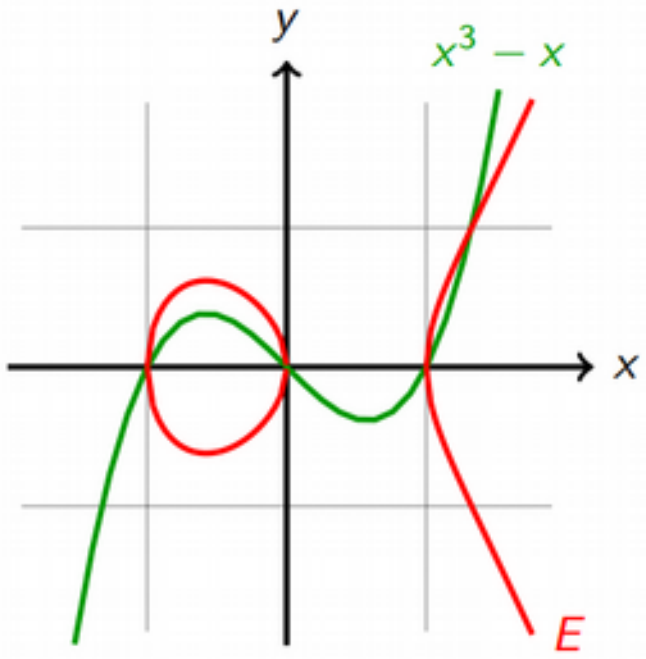
$$\int_a^b \frac{dx}{\sqrt{x^3 + ax^2 + bx + c}}$$

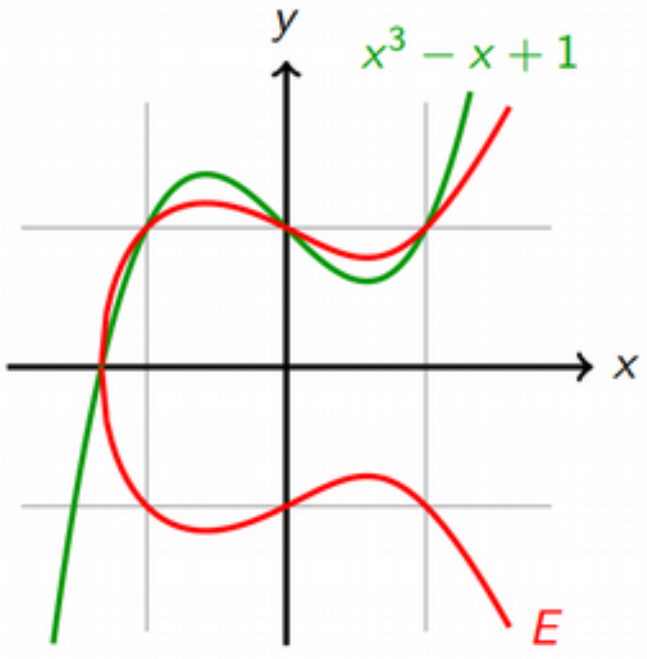
Elliptic Curve

- An elliptic curve is the set

$$E = \{(x, y) : y^2 = x^3 + ax^2 + bx + c\}$$

- Examples:





Elliptic curve cryptography (ECC)

Elliptic curves have been studied by mathematicians for over a hundred years. They have been deployed in diverse areas

- Number theory: proving Fermat`s Last Theorem in 1995 [4]
  - The equation $x^n + y^n = z^n$ has no nonzero integer solutions for x,y,z when the integer n is grater than 2.
- Modern physics: String theory
  - The notion of a point-like particle is replaced by a curve-like string.
- Elliptic Curve Cryptography
  - An efficient public key cryptographic system.

Elliptic curve cryptography (ECC)

Elliptic curves over real numbers

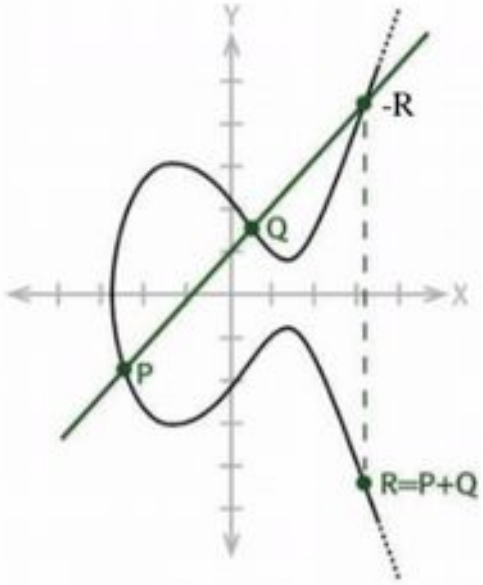
- Calculations prove to be slow
- Inaccurate due to rounding error
- Infinite field

Cryptographic schemes need fast and accurate arithmetic

- In the cryptographic schemes, elliptic curves over two finite fields are mostly used.
  - Prime field $\mathbb{F}_p$, where p is a prime.
  - Binary field $\mathbb{F}_{2^m}$, where m is a positive integer.

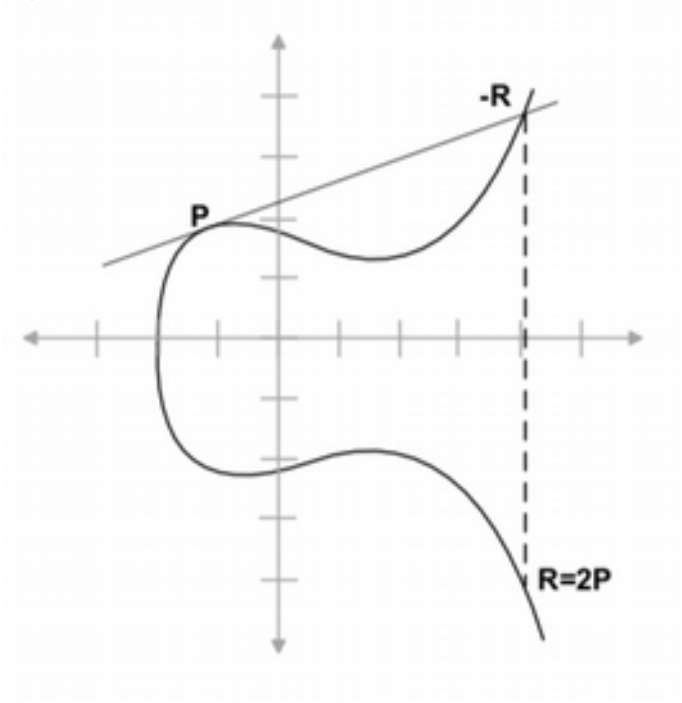
Point Addition

To add two distinct points P and Q on an elliptic curve, draw a straight line between them. The line will intersect the elliptic cure at exactly one more point –R. The reflection of the point –R with respect to x-axis gives the point R, which is the results of addition of points P and Q.



Point Doubling

To the point P on elliptic curve, draw the tangent line to the elliptic curve at P. The line intersects the elliptic cure at the point –R. The reflection of the point –R with respect to x-axis gives the point R, which is the results of doubling of point P.



Scalar Multiplication

Intuitive approach:

$$dP = \underbrace{P+P+\ldots+P}_{d \text{ times}}$$

It requires *d*-1 times point addition over the elliptic curve.

Observation: To compute 17 *P*, we could start with 2*P*, double that, and that two more times, finally add *P*, i.e. 17*P*=2(2(2(2*P*)))+*P*. This needs only 4 point doublings and one point addition instead of 16 point additions in the intuitive approach. This is called Double-and-Add algorithm.

Elliptic Curve Cryptography

- Key exchange
  - ECDH -Elliptic Curve Diffie-Hellman
- Digital Signatures
  - ECDSA -Elliptic Curve Digital Signature Algorithm
- ECDH and ECDSA are standard methods
- Encryption/Decryption with ECC is possible, but not common

ECC Cryptography

- Remember the discrete logarithm problem: given x and a primitive root g, find k so that

$$x = g^k \bmod p$$

- There is an analog on elliptic curves: given two points A and B on an elliptic curve, find k so that

$$B = kA = A + A + \ldots + A$$

- This might seem different, but is the equivalent problem. The only difference is the group operation name ("multiplication or "addition")

ECC Cryptography

Elliptic curves are used to construct the public key cryptography system

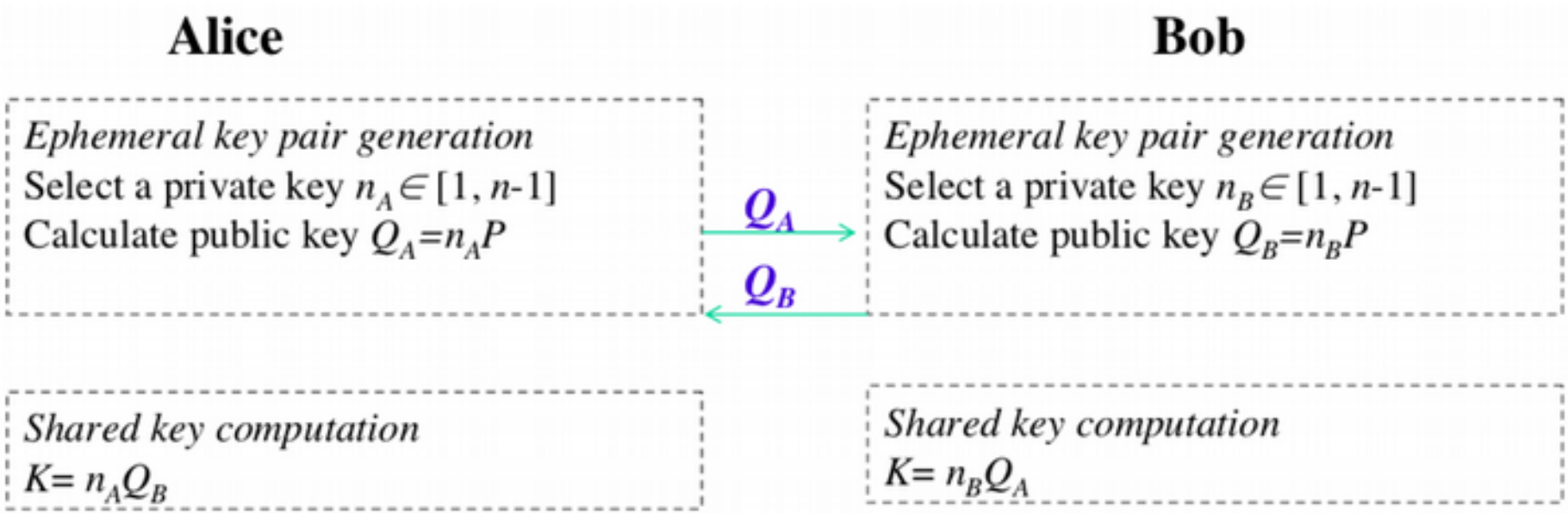
The private key d is randomly selected from [1,n-1], where n is integer.

Then the public key Q is computed by dP, where P,Q are points on the elliptic curve.

Like the conventional cryptosystems, once the key pair (d, Q) is generated, a variety of cryptosystems such as signature, encryption/decryption, key management system can be set up.

Computing dP is denoted as **scalar** multiplication. It is not only used for the computation of the public key but also for the signature, encryption, and key agreement in the ECC system.

Elliptic Curve Deffie-Hellmen (ECDH)



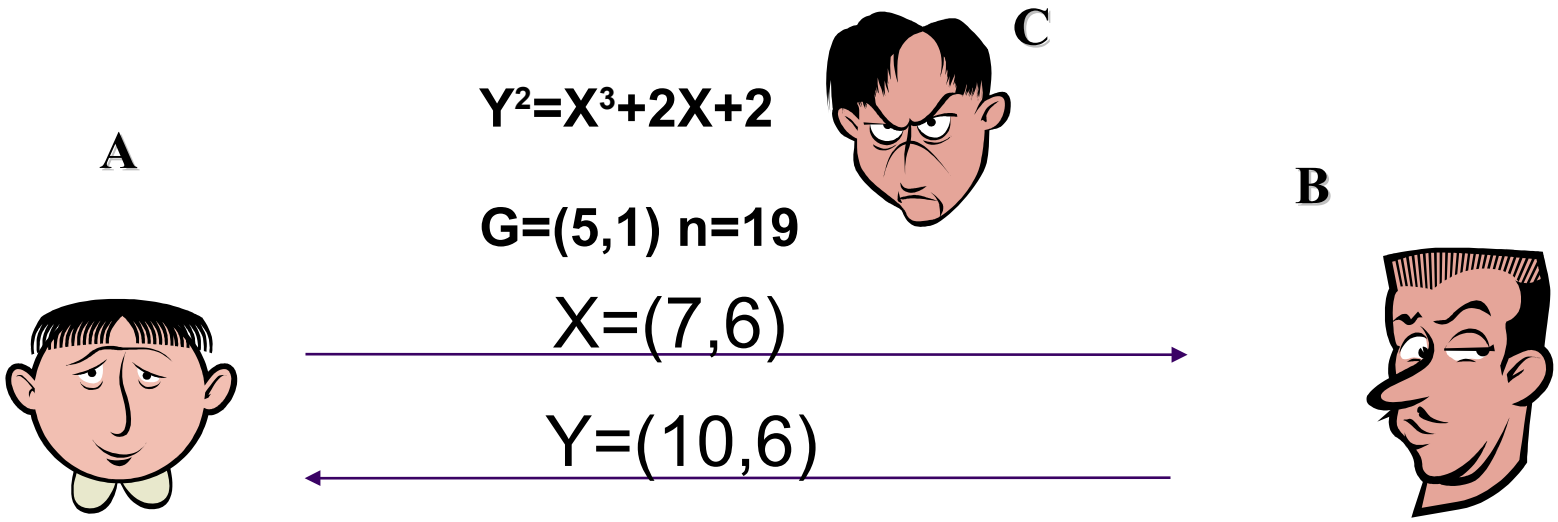
- **Consistency: $K = n_A Q_B = n_A n_B P = n_B Q_A$**
- **ECDH is vulnerable to the man-in-the-middle attack**

http://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/

Example Curve $Y^2=X^3+2X+2$

- **G = (5,1)**
- 2G=(6,3)
- 3G=2G+G=(10,6)
- 4G=2(2G)=(3,1)
- 5G=2(2G)+G=(9,16)
- 6G=2(2G)+2G=(16,13)
- 7G=2(2G)+2G+G=(0,6)
- 8G=2(2(2G))=(13,7)
- 9G=2(2(2G)))+G=(7,6)
- 10G=2(2(2G))+2G=(7,11)
- 11G=2(2(2G))+2G=G=(13,10)
- 12G=2(2(2G))+2(2G)=(0,11)
- 13G=2(2(2G))+2(2G)+G=(16,4)
- 14G=2(2(2G))+2(2G)+2G=(9,1)
- 15G=2(2(2G))+2(2G)+2G+G=(3,16)
- 16G=2(2(2(2G)))=(10,11)
- 17G=2(2(2(2G)))+G=(6,14)
- 18G=2(2(2(2G)))+2G=(5,16)
- **19G=2(2(2(2G)))+2G+G=0 (infinite)**

Elliptic Cure Diffie Hellmann - Example



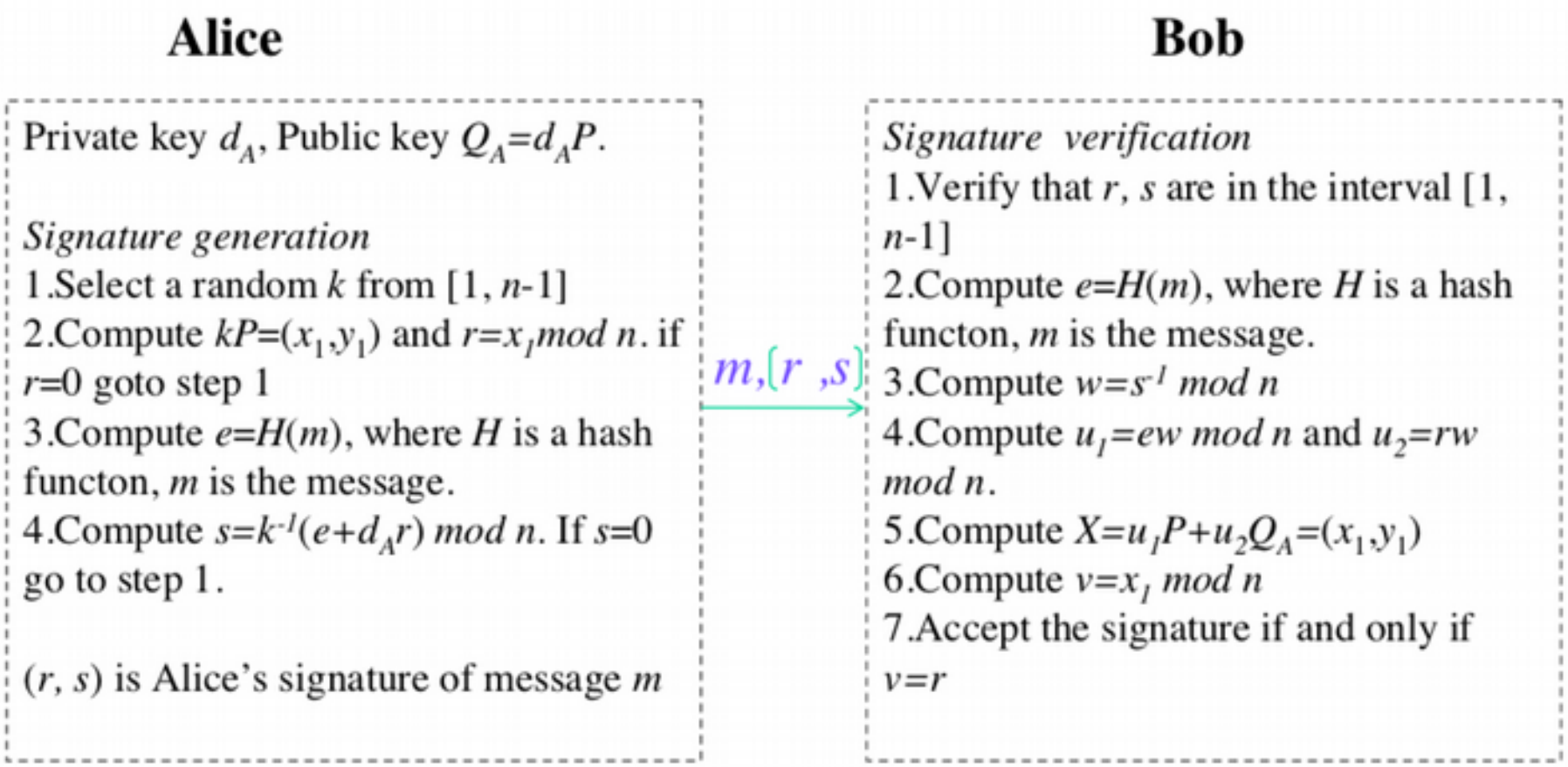
Alice picks **x=9**
Alice's **X=9G = (7,6)**

Alice's
k=9Y = 9(3G)=27G= 8G
= (13,7)

Bob picks **y=3**
Bob's Y = **Y=3G = (10,6)**

Bob's
k = 3X= 3(9G)=27G
=(13,7)

Elliptic Curve Digital Signature Algorithm (ECDSA)



Key measure: Encryption strength

The mathematic background of ECC is more complex than other cryptographic systems

•Geometry, abstract algebra, number theory

ECC provides greater security and more efficient performance than the first generation public key techniques (RSA and Diffie-Hellman)

•Mobile systems

•Systems required high security level (such as 256 bit AES)

Bits of Security	Symmetric Key Algorithm	Corresponding RSA Key Size	Corresponding ECC Key Size
80	Triple DES (2 keys)	1024	160
112	Triple DES (3 keys)	2048	224
128	AES-128	3072	256
192	AES-192	7680	384
256	AES-256	15360	512

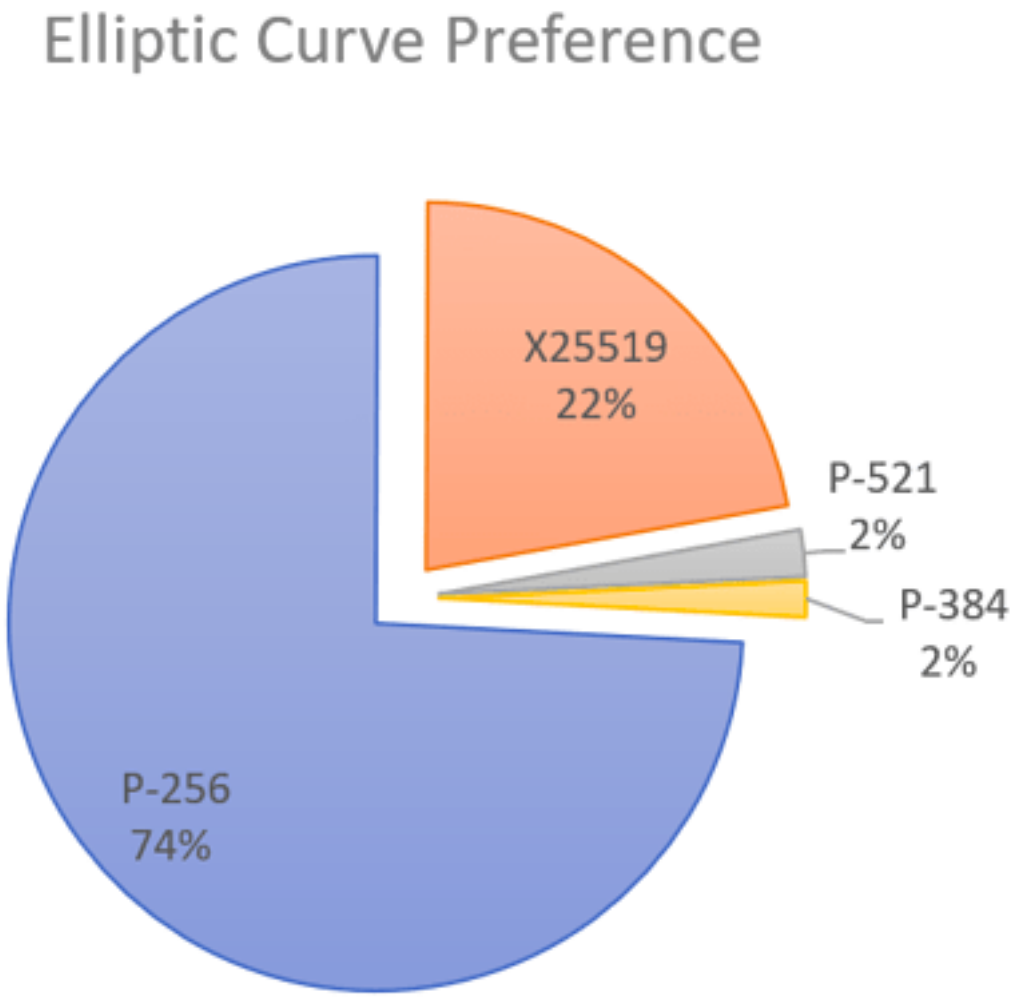
COMPUTATION TIMES OF CURVES WHEN USED FOR ECDH ALGORITHM

Type of curve	Time taken for point addition (ns)	Time to calculate Alice's public key $A(K_{Pb}) = a*P$ (ms)	Time to calculate Bob's public key $B(K_{Pb}) = b*P$ (ms)	Time to calculate secret on Alice side $a*B(K_{Pb})$ (ms)	Time to calculate secret on Bob side $b*A(K_{Pb})$ (ms)
M221	136	17.9	15.1	15.6	16.2
NIST P-224	73.2	14.3	13.9	13.8	14.4
Curve25519	144	20.2	15.5	14.9	14.1
BN(2,254)	73.2	14.6	13.9	16.4	14.4
Brainpool P256t1	70.2	16.2	16.5	14.8	15.1
NIST P-256	68.7	14.8	20	17	20.4
SECP 256k1	67.1	13.9	15.4	14.3	13.9
SECP 256r1	78.2	13.7	13.3	13.6	13.7
NIST P-384	70.2	15.4	15.1	18.7	17.3
M-511	72.1	22.2	16.2	16.4	16.1

COMPUTATION TIMES OF CURVES WHEN USED FOR ECDSA

Type of curve	Time taken for Signature Generation (*r*, *s*)		Time taken for Signature Verification (ms)
	Time to compute (*r*) (μs)	Time to compute (*s*) (μs)	
M221	1.98	3.38	26
NIST P-224	1.73	3.54	26.6
Curve25519	2.06	3.44	32.8
BN(2,254)	1.68	3.47	29.7
Brainpool P256t1	1.78	3.40	30
NIST P-256	3.37	6.66	29.9
SECP256k1	1.73	3.33	36
NIST P-384	1.68	4.09	44.7
M-511	1.91	5.29	72.7

TLS - Preference



Discussion