Cybersecurity

Public-Key Cryptography

Mauro Barni
University of Siena

Private-Key Cryptography

- Traditional **secret key** cryptography uses **one** key
  - shared by both sender and receiver
  - if this key is disclosed communication secrecy is compromised
- Traditional crypto is **symmetric**, parties are equal
  - hence does not protect sender from receiver forging a message & claiming it was sent by the sender

Public-Key Cryptography

- Probably most significant advance in the 3000 years history of cryptography
- Based on number theoretic concepts rather than on substitutions and permutations
- Uses **two** keys – a public & a private key
- It is **asymmetric** since parties are **not** equal

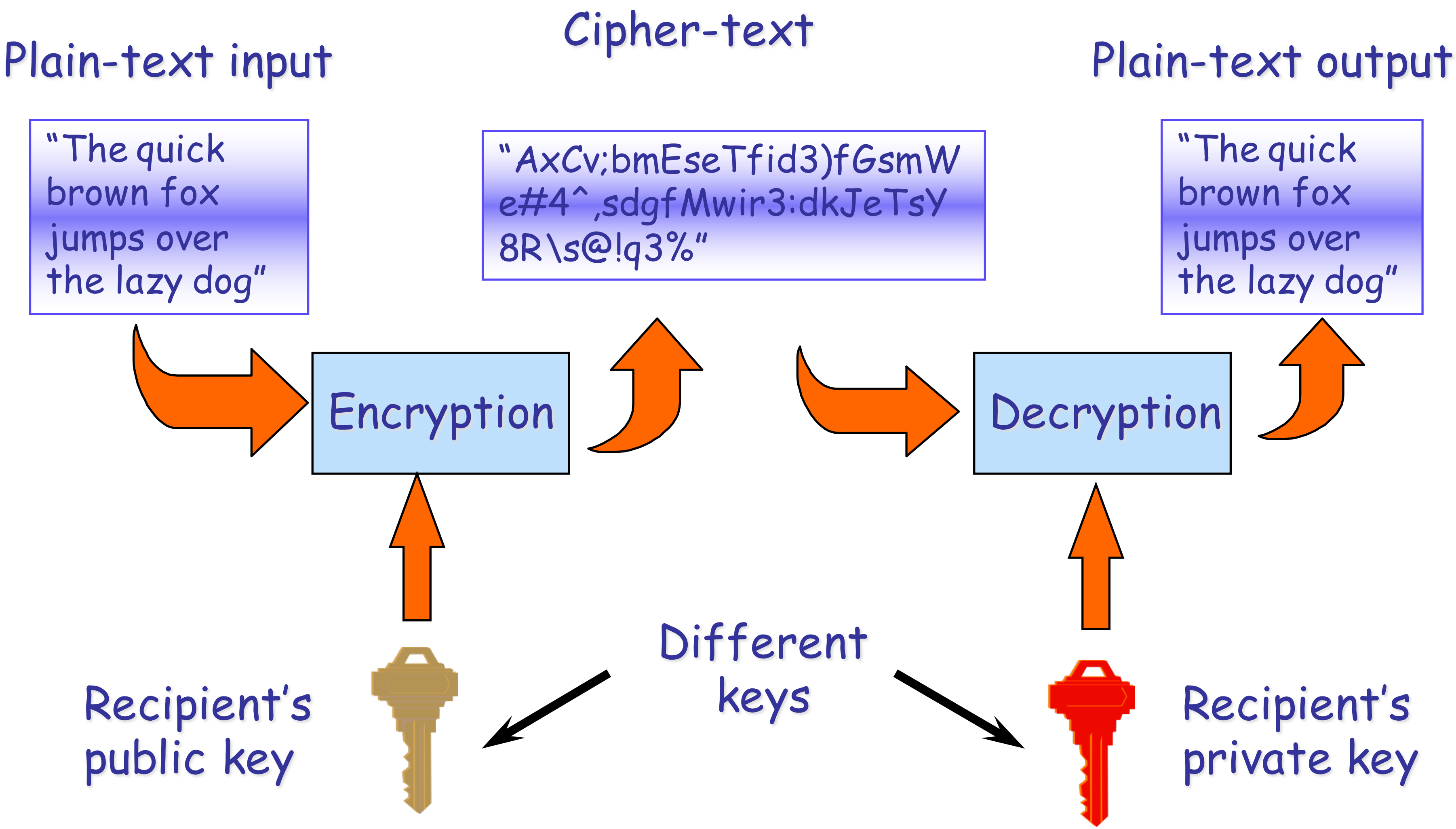
Public-Key Cryptography

- Public-key schemes are typically slower than symmetric-key algorithms
  - most commonly used in practice for the transport of keys used for data encryption by symmetric algorithms
  - for encrypting small data items such as credit card numbers and PINs.
- Complements **rather than** replaces private key crypto
- It is not intrinsically more secure than private key crypto

Public-Key Cryptography

- **Public-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public-key**, distributed by the owner to anybody,
  - a **private-key**, known only to the owner.
- Each user will thus have a collection of public keys of all the other users.
- It is **asymmetric** because
  - keys used to encrypt messages **cannot** be used to decrypt them

Asymmetric Cryptography



Why Public-Key Crypto?

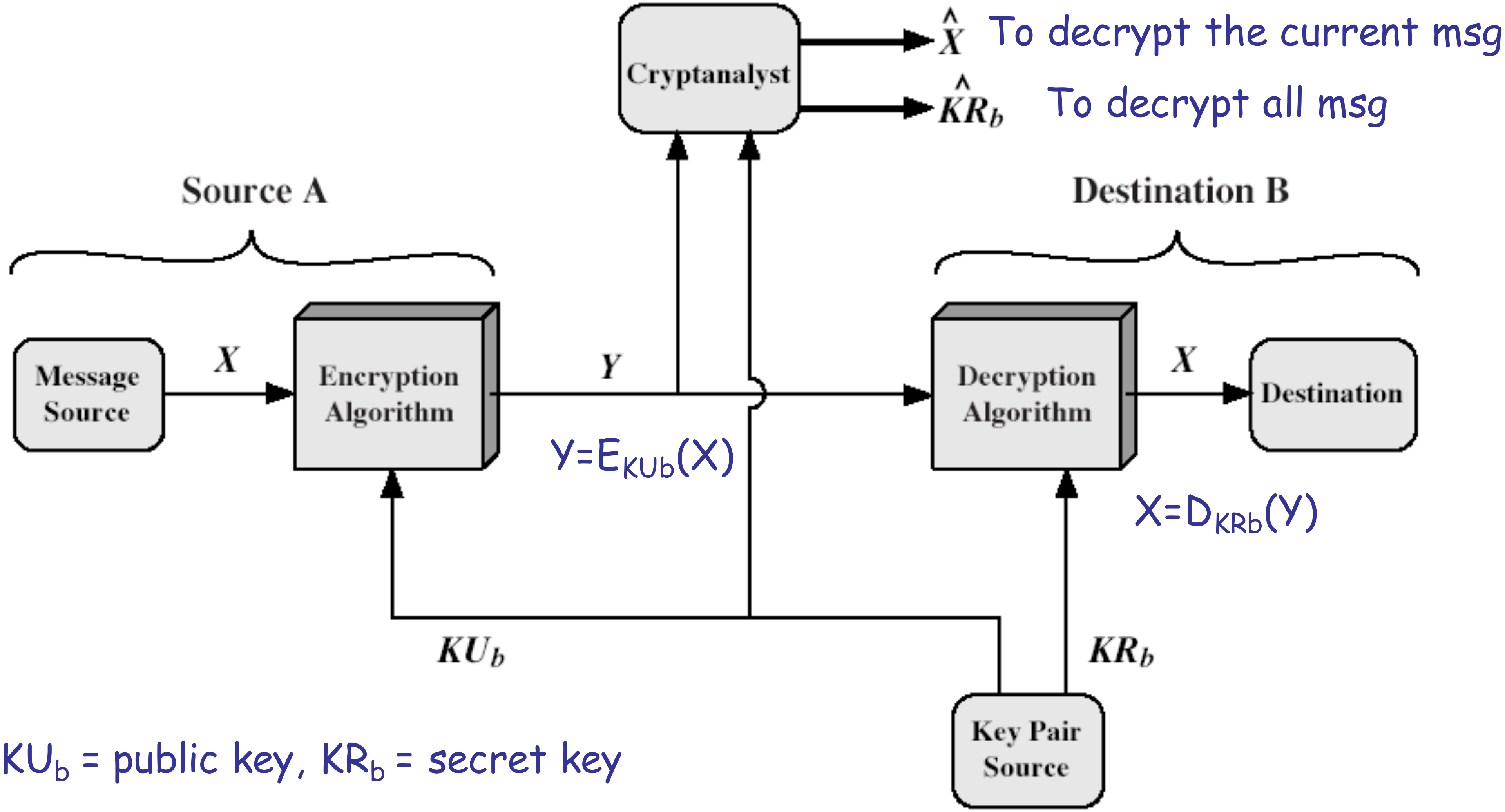
- It was developed to address two key issues:
  - **key distribution**
    - how to communicate securely without trusting a KDC
  - **digital signatures**
    - verify that a message is intact and comes from the claimed sender
- *Public* invention due to Diffie & Hellman at Stanford University in 1976
  - The concept had been previously described in a classified report in 1970 by James Ellis (UK CESG) - and subsequently declassified in 1987

Public-Key Applications

We can classify its uses into 3 categories:

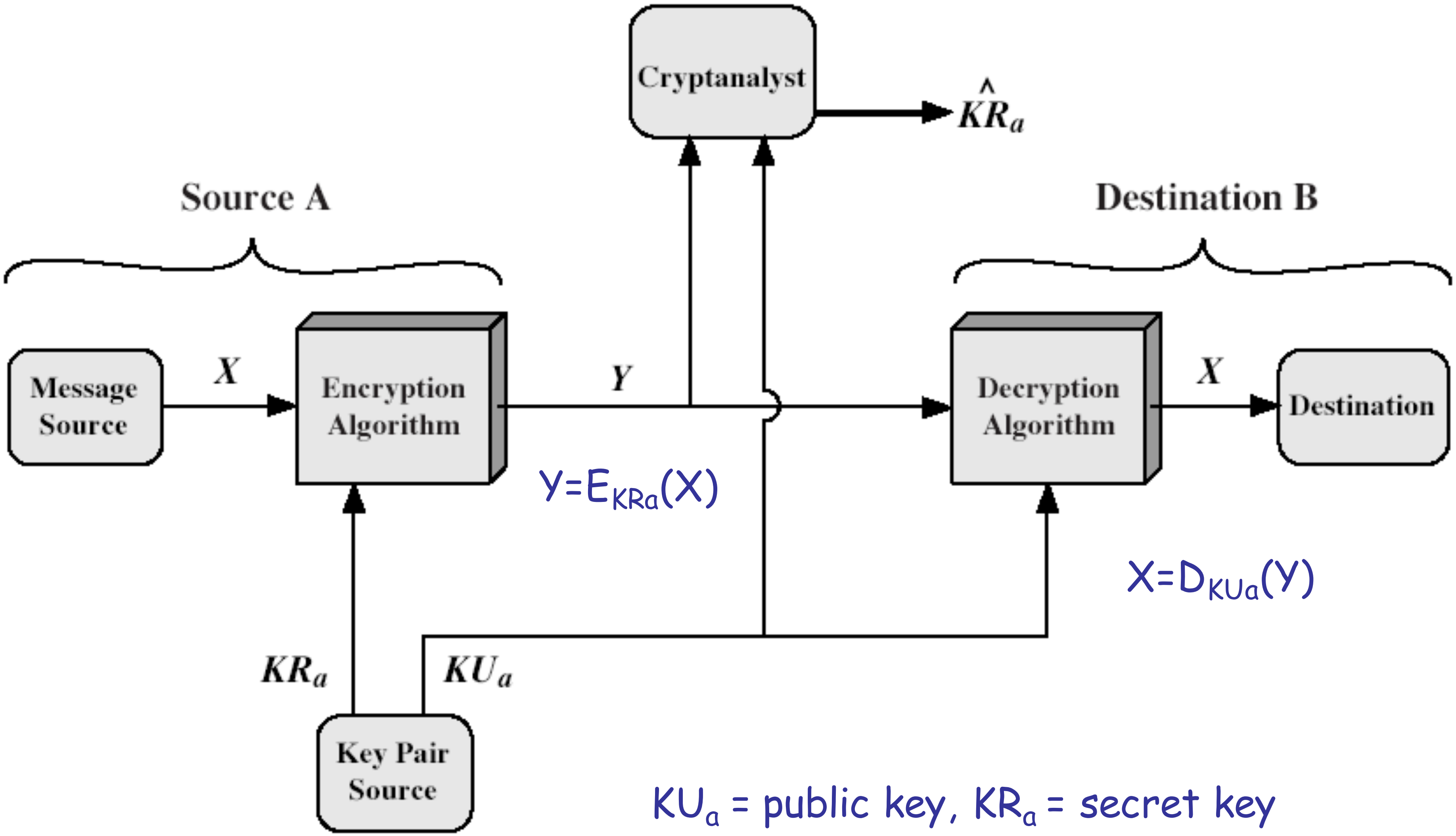
- **encryption/decryption** (secrecy)
  - sender encrypts the msg with recipient's public key
- **digital signatures** (authentication & data integrity)
  - sender encrypts msg with his/her private key
- **key exchange** (of session keys)
  - several approaches, using one or two private keys .

Confidentiality, key distribution



KU_b = public key, KR_b = secret key

Authentication, without confidentiality

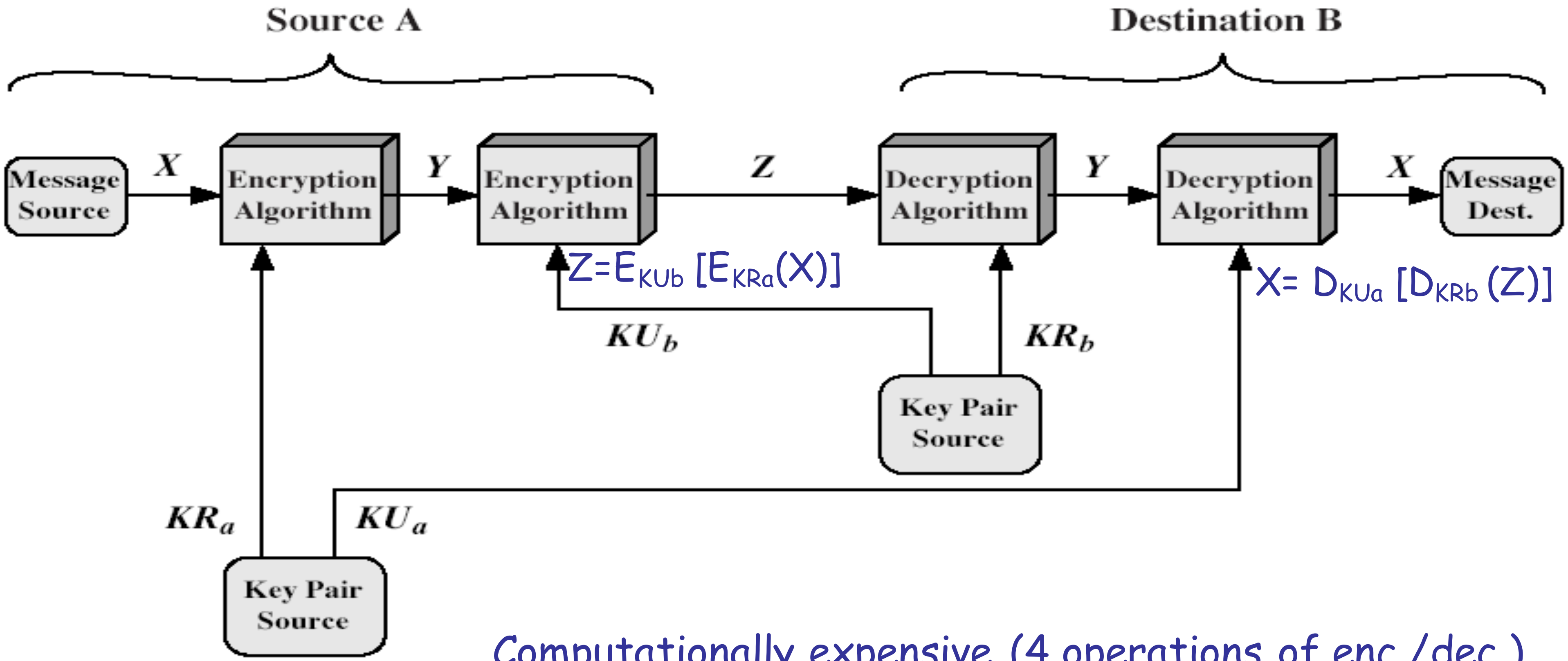


$Y=E_{KRa}(X)$

$X=D_{KUa}(Y)$

KU_a = public key, KR_a = secret key

Confidentiality and authentication



$Z=E_{KUb}[E_{KRa}(X)]$

$X= D_{KUa}[D_{KRb}(Z)]$

Computationally expensive (4 operations of enc./dec.)

Public-Key Applications

Some algorithms are suitable for all uses, others are specific to one

Table 9.2 Applications for Public-Key Cryptosystems

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

Requirements of Pub. Key Algorithms (DH)

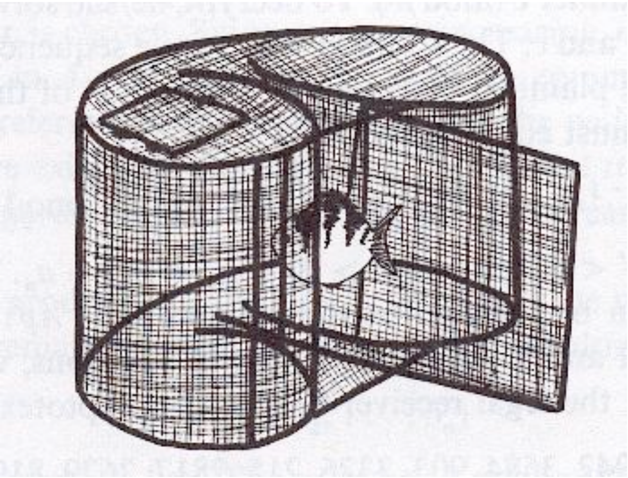
1. Computationally easy to generate a key pair
2. Computationally easy for sender A to generate the encrypted msg $Y=E_{KUb}(X)$
3. Computationally easy for recipient B to decrypt the encrypted msg $X=D_{KRb}(Y)$
4. Computationally impossible for an intruder, by knowing KUb, to determine the key KRb
5. Computationally impossible for an intruder, by knowing KUb and Y, to determine msg X
6. It should be possible to apply encryption/decryption in whatever order

Requirements of Pub. Key Algorithms (DH)

- These requirements are very difficult to be satisfied: only elliptic curves and RSA have been accepted !
- These reqs can be satisfied if we can find a monodirectional “trapdoor function” f.

Trapdoor function

- A **trapdoor function** is a function easy to compute in one direction, yet believed to be difficult to compute in the opposite direction (finding its inverse) without special information, called the "trapdoor".



Trapdoor function

- In mathematical terms, *f* is a trapdoor function if there exists some secret information *K*, such that given *f(x)* and *K* it is easy to compute *x*.
  - Consider taking an engine apart: not very easy to put it together again unless you had the assembly instructions (the trapdoor).
  - A mathematical example: the multiplication of two large prime numbers. Multiplication is easy; but factoring the resultant product can be very difficult.

Trapdoor function

- This monodirectional “trap function” f maps a domain into an interval such that each function value has an unambiguous inverse, and such that:
  - $Y = f_k(X)$ easy;
  - $X = f_k^{-1}(Y)$ easy if *k* and *Y* are known;
  - $X = f_k^{-1}(Y)$ hard if *Y* is known, but *k* unknown ;
- The precise meanings of "easy" and "hard" can be specified mathematically:

Trapdoor function

- *easy*: a problem that we can solve within a polynomial time with respect to the input length: if input is n bits, the time to compute a function is proportional to n^a where a is a fixed constant (Class P problems);
- *hard*: a problem that we can solve only within a time larger than polynomial (hopefully exponential): if input is n bits, the time to compute a function is proportional to 2^{an}.
- To determine the level of complexity of a problem is extremely complicated !!!!

Security of Public Key Schemes

- Security relies on a large enough difference in difficulty between **easy** (en/decrypt) and **hard** (cryptanalysis) problems
- The hard problem is known, it’s just made too hard to solve it in practice
  - requires the use of very large numbers
  - hence public key crypto is slower than secret key schemes
- Like secret key schemes brute force attack is always theoretically possible, but keys used are too large (>= 1024bits)

From a trapdoor to a cryptosystem

- Construct public-key cryptosystem from trapdoor-one way function f:

  - Encryption requires evaluation of f
  - Decryption uses trapdoor to invert f

  - Trapdoor is secret key
  - Attacker has to invert f

In search for a trapdoor

- Examples of potential trapdoor one-way functions
  - $f(x,a,n) = y = x^a \bmod n$
    - Hard problem: compute $x = f^{-1}(y,a,n)$
    - Trapdoor: factors of $n=pq$
    - Basis for RSA encryption
  - $f(g,x,p) = y = g^x \bmod p$
    - Hard problem: $x = \log_g(y)$
    - Basis for ElGamal encryption, and DH key exchange

RSA algorithm

- Invented by Rivest, Shamir & Adleman at MIT in 1977
- Best known & widely used public-key scheme
- Security based on the intractability of the integer factorization problem.

RSA algorithm

- Currently used in a wide variety of products, platforms, and industries around the world.
  - RSA is built into current operating systems by Microsoft, Apple, Sun, and Novell.
  - In hardware, RSA can be found in secure telephones, on Ethernet network cards, and on smart cards.
  - RSA is incorporated into all of the major protocols for secure Internet communications, including S/MIME, SSL, and S/WAN.

RSA algorithm

- RSA is a block cipher:
- The plaintext is divided into blocks, where each block is represented as an integer value between 0 and $n-1$, n being the modulus.
  - n is a very big number, represented with k bits, i.e. $2^{k-1} < n < 2^k$
  - Usually $k=1024$ bits, i.e. n is composed by 309 decimal figures ($n < 2^{1024}$).
- The ciphertext is obtained by a proper exponentiation of the plaintext modulo n.

RSA Key Setup

Each user generates a public/private key pair by:

- selecting two large primes at random: p,q
- computing the system modulus $n=pq$
- compute Euler totient function $ø(n)=(p-1)(q-1)$
- selecting at random a value e
  - where $1 < e < ø(n)$, $\gcd(e, ø(n))=1$
- solving the following equation to find a value d:
  - $ed = 1 \bmod ø(n) \iff d = e^{-1} \bmod ø(n)$, $0<d<n$

RSA Key Setup

- *e* is called the encryption exponent, *d* the decryption exponent, *n* the modulus.
- Each user:
  - publishes the public key: KU={e,n}
  - keeps secret the private key: KR={d,n}
- So if we encrypt with the recipient's public key:
  - Sender will know e and n
  - Recipient will know d and n

RSA encryption/decryption

- To encrypt a message block m $(0<m<n)$, the sender:
  - obtains public key of recipient $KU=\{e,n\}$
  - computes: $c = m^e \bmod n$
- To decrypt the ciphertext c the owner:
  - uses his private key $KR=\{d,n\}$
  - computes: $m=c^d \bmod n$

- **Remember: message block is represented as an integer m smaller than the modulus n and relatively prime with n (for security reason) !**

Why does RSA work ?

Because of Euler's Theorem in number theory:

- given two prime numbers p and q, n and m integers such that $n=pq$, and $m<n$:
- $m^{k\phi(n)+1} = m \bmod n$, where $\phi(n)$ is the Euler totient function,

Why does RSA work ?

In RSA we have:

- $n = pq$
- $ø(n) = (p-1)(q-1)$
- Integers e and d are chosen to be inverse mod $ø(n)$
- Then $ed=1+kø(n)$ for some k

Hence :

$$c^d = (m^e)^d = m^{1+kø(n)} = m \bmod n = m$$

since $0<m<n$

RSA - toy example

- Select primes: p =17, q =11
- Compute $n = pq = 17\times11=187$
- Compute $\phi(n)=(p-1)(q-1)=16\times10=160$

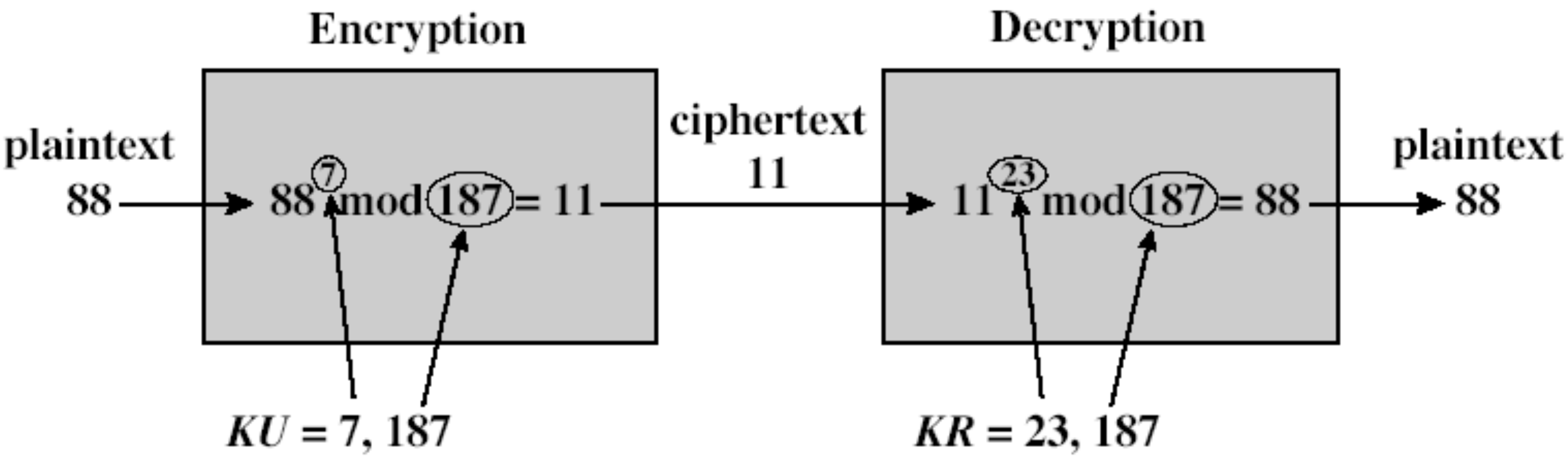
- Select e : 1< e <160, gcd(e,160)=1; choose e = 7
- Determine d: de=1 mod 160 and d < 160
  - d = 23 since 23×7=161 = 1 mod 160
- Publish public key: **KU= {7,187}**
- Keep secret private key: **KR= {23,187}**

RSA – toy example

- given message $m = 88$ (n.b. $88 < 187$)
- encryption:

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

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



Computational aspects: enc/dec

- Encryption/decryption require the computation of exponentiation between large integers mod n.
- A fast, efficient algorithm for exponentiation exists
- Due to the modular operator properties, we can compute ((a x b) mod n) as [(a mod n) x (b mod n)] mod n

- Example: 7^5 mod 11 = $7^4 7^1$ mod 11 = $(7^2 7^2)7^1$ mod 11 = [(($7^2 7^2$) mod 11) x 7mod11] mod 11 = [(49 mod 11)(49 mod 11) mod 11 x7] mod 11 = [((5x5)mod11)x7] mod 11 = 3x7 mod 11 = 10
- Exercise: compute 3^{129} mod 11

Computational aspects: Key Generation

- Users of RSA must:
  - determine two primes at random p,q
  - select either e or d and compute the other
- Primes p,q must be secure, i.e. not easily recoverable from modulus $n=pq$
  - Prime numbers must be sufficiently large
  - An efficient method to obtain big prime numbers does not exist

Computational aspects: Key Generation

- Exponents e, d are inverse each other, so, chosen e value, it is possible to use the extended Euclidean algorithm to compute d:
    - e: $\gcd(e,ø(n)) = 1$ (randomly generated)
    - $d = e^{-1} \bmod ø(n)$ (ext. Euclidean alg.)
- Possible (easy) iff p and q are known

Euclidean GCD algorithm

- Let a and b be two integers ($a > b$)
- If q is a divisor of a and b it also divides $r = a \bmod b$
- The we can proceed as follows

$r_1 = a \bmod b$

if $r_1 = 0$ MCD = b

else $(a,b) \to (b,r_1)$

…

$r_n = r_{n-2} \bmod r_{n-1}$

if $r_n = 0$ MCD = r_n

else $(r_{n-2}, r_{n-1}) \to (r_{n-1}, r_n)$

- Convergence is ensured

Extended euclidean GCD algorithm

- Going backword it is always possible to write MCD as an integer linear combination of a and b, that is:

  $MCD = s \cdot a + t \cdot b$

- We can find s and t proceeding as fgollows:

  $r_{n-1} = r_{n-3} - q_{n-2}\, r_{n-2} = r_{n-3} - q_{n-2}\,(r_{n-4} - q_{n-3}\, r_{n-3})$

  $= (1+ q_{n-2} q_{n-3})\, r_{n-3} - q_{n-2}\, r_{n-4}$

  $= (1+ q_{n-2} q_{n-3})(r_{n-5} - q_{n-4}\, r_{n-4}) - q_{n-2}\, r_{n-4} \ \dots$

- The extended Euclidean GCD can be used to find the modular inverse for coprime numbers

  GCD(a,n) = 1 -> 1 = sa-tn -> sa = tn +1

  sa mod n = 1 -> s = a^{-1} mod n

RSA example: key generation

- Let the primes 5 and 11 to be our p and q.
- n = 55, and ø(55) = (5-1)(11-1) = 4x10 = 40.
- Now, we need to find e, d such that: ed = 1 (mod 40).
  - There are many pairs fitting this equation. We need to find one of them.
  - Our only constraint is that e and d are both relatively prime to ø(55) = 40. So, we can't use numbers that are multiples of 2 and/or 5. Ideally, in fact, we'd prefer that e and d be relatively prime to each other. Let us try with e = 7

RSA example: key generation

- Now we need to find d such that 7d = 1 (mod 40). This means find d and K such that :
  - 7d = 40K + 1.
  - The first value for d that works is 23
  - 7 * 23 = 161 = 4 * 40 + 1. So we have e = 7 for d = 23
- Publish public key: **`KU = {7,55}`**
- Keep secret private key: **`KR = {23,55}`**

RSA example: the plaintext

- To put the cipher at work, we must recall that the values we use for the plaintext m must be less than $n=55$, and also relatively prime to 55.
- We also do not want to use $m = 1$, because 1 raised to any power whatsoever is going to remain 1.
- Finally, the same holds true for $n - 1$, because $n - 1$ is congruent to $-1 \bmod n$.

- Then the valid messages are the numbers m such that:
  – $1 < m < 54$
  – Not multiple of $5,11$.

RSA example: the plaintext

- So, we'll take what's left and create the following character set:
  - 2 3 4 6 7 8 9 12 13 14 16 17 18
  - A B C D E F G H I J K L M
  - 19 21 23 24 26 27 28 29 31 32 34 36 37
  - N O P Q R S T U V W X Y Z
  - 38 39 41 42 43 46 47 48 49 51 52 53
  - sp 0 1 2 3 4 5 6 7 8 9 *

RSA example: encryption

- The message we will encrypt is VENIO :
- VENIO = 31, 7, 19, 13, 21
- To encode it, we simply need to raise each number to the power of e modulo n.
- V = 31^7 (mod 55) = 27512614111 (mod 55) =26
- E = 7^7 (mod 55) = 823543 (mod 55) =28
- N = 19^7 (mod 55) = 893871739 (mod 55) =24
- I = 13^7 (mod 55) = 62748517 (mod 55) = 7
- O = 21^7 (mod 55) = 1801088541 (mod 55) = 21
- The encrypted message is 26, 28, 24, 7, 21 = RTQEO

M. Barni, University of Siena

RSA example: decryption

- To decrypt the message RTQEO we repeat the same process using d instead than e
- R = 26^{23} (mod 55) = 350257144982200575261531309080576 (mod 55) = 31
- T = 28^{23} (mod 55) = 1925904380037276068854119113162752 (mod 55) = 7
- Q = 24^{23} (mod 55) = 55572324035428505185378394701824 (mod 55) = 19
- E = 7^{23} (mod 55) = 27368747340080916343 (mod 55) = 13
- O = 21^{23} (mod 55) = 2576580875108218291929075869661 (mod 55) = 21
- Yielding: 31, 7, 19, 13, 21 = VENIO

A not-so-simple example

- This time, to make life slightly less easy, we group the characters into blocks of three and compute a representative integer for each block.
- ATTACKxATxSEVEN = ATT ACK XAT XSE VEN
- We could represent our blocks of three characters in base 26 using A=0, B=1, C=2, ..., Z=25
  - ATT = 0 x 26^2 + 19 x 26^1 + 19 = 513
    ACK = 0 x 26^2 + 2 x 26^1 + 10 = 62
    XAT = 23 x 26^2 + 0 x 26^1 + 19 = 15567
    XSE = 23 x 26^2 + 18 x 26^1 + 4 = 16020
    VEN = 21 x 26^2 + 4 x 26^1 + 13 = 14313

A not-so-simple example

- In this system of encoding, the maximum value of a group (ZZZ) would be $26^3-1 = 17575$, so we require a modulus n greater than this value.
- We can use $p=137$ and $q=131$ (we cheated by looking for suitable primes around $\sqrt{n}$, which is not good for security reasons)
- $n = pq = 137 \times 131 = 17947$
  $\phi(n) = (p-1)(q-1) = 136 \times 130 = 17680$
- Select $e = 3$
  check $\gcd(e, p-1) = \gcd(3, 136) = 1$, OK and
  check $\gcd(e, q-1) = \gcd(3, 130) = 1$, OK.
- Compute $d = e^{-1} \bmod \phi(n) = 3^{-1} \bmod 17680 = 11787$.
- Hence public key = (17947, 3), private key = (17947, 11787).

A not-so-simple example

- To encrypt the first integer representing ATT, we have $c = m^e \bmod n = 513^3 \bmod 17947 = 8363$.
- We can verify that our private key is valid by computing
  $m' = c^d \bmod n = 8363^{11787} \bmod 17947 = 513$.
- Overall, our plaintext is represented by the set of integers $m = \{513, 62, 15567, 16020, 14313\}$
- Yielding $c = m^e \bmod n = \{8363, 5017, 11884, 9546, 13366\}$
- You are welcome to compute the inverse of these integers using $m = c^d \bmod n$ to verify that RSA works

Practical considerations

- If we know only the public key, how can we be sure that GCD(m,n) = 1?
  - Use Euclide's algorithm…
  - Note that Pr{GCD(m,n) = 1} = ø(n)/n = (p-1)(q-1)/pq
  - Pr{GCD(m,n) ≠ 1} = 1 - (p-1)(q-1)/pq = (p+q-1)/pq
  - If p,q have 512 bits, Pr{GCD(m,n) ≠ 1} ~ 2^{-511} !!
  - The probability of picking a wrong message is almost zero, so usually we do not care

How fast is RSA algorithm ?

- It is common to choose a small public exponent for the public key
  - This makes encryption faster than decryption and verification faster than signing
- DES and other block ciphers are much faster than the RSA algorithm.
  - DES is generally at least 100 times faster in sw and 1,000 ÷10,000 times faster in hw

RSA Security

Theorem: Computing the secret key from the public key is computationally equivalent to factoring n.

No efficient factorization algorithms is known

– general number field sieve (GNFS) algorithm:

– $O(\exp(k^{1/3}(\log k)^{2/3}))$ complexity

– k is the number of bits of n

Exact security of RSA is unknown

– more efficient factorization algorithms may be found

– pay attention to choose secure primes

RSA Security

- Three approaches to attack RSA
  - brute force key search (difficult given key size)
  - mathematical attacks (it is difficult to compute $ø(n)$, by factoring modulus n)
  - timing attacks (based on measuring the time to run the decryption)
- Yet, care must be taken to use RSA properly

Common Modulus attack

- Suppose that RSA is used by several parties who share a common modulus (but different e and d)
- We can show that if the public exponents of the participants are relatively prime, an attacker can recover the message sent to at least two parties.

Common Modulus attack

- Assume Alice and Bob generated keys using the same modulus n: $(e1, d1)$ and $(e2, d2)$
- Also suppose that $GCD(e1,e2)=1$
- Assume a third user sends to Alice and Bob the same message m:
  - $c_1 = m^{e1} \bmod n$,
  - $c_2 = m^{e2} \bmod n$

Common Modulus attack

- $c_1 = m^{e1} \bmod n$,
- $c_2 = m^{e2} \bmod n$
- if $\gcd(e1,e2)=1$, then it is possible to compute a,b so that $(e1)\, a + (e2)\, b = 1 \bmod n$ (extended Euclidean algorithm)
- then
- $c_1^a\, c_2^b = m^{e1\, a + e2\, b} \bmod n = m \bmod n = m$

Never send identical messages to receivers with the same modulus and relatively prime encryption exponents

Adaptive chosen-ciphertext attack

- Suppose that an active adversary wishes to decrypt $c=m^e \bmod n$ intended for the user A.
- Suppose that A is available to decrypt an arbitrary ciphertext for the adversary, other than c itself.
- The adversary can select a random integer x and compute $c' = cx^e \bmod n = (mx)^e \bmod n$.
- Upon presentation of c', A will compute for the adversary $m' = mx \bmod n$.
- The adversary can then compute $m=m'x^{-1} \bmod n$.
- This attack can be circumvented by imposing some structural constraints on plaintext messages.

El Gamal

- **El Gamal encryption system** is based on the discrete logarithm problem,
- Described by Taher El Gamal in 1984.
- Implemented in GnuPG. A similar signature scheme is used in DSA (Digital Signature Algorithm, standardized in 1993).

El Gamal

- Let G be a cyclic group of order q, with generator g
- Usually G is Z_p^*, the multiplicative group of integers modulo p, where p is a big prime ($q=p-1$)
- Let x be a random number taken in {2 ... p-2}, compute $h=g^x \bmod p$
- **Public key: (g,h,p)**
- **Private key: x**

El Gamal Encryption

- To encrypt a message m under Alice's public key (g,h,p),
- Bob converts m into an integer in G={1 … p-1}
- Then he chooses a random y in {2 … p-2}, and computes $c_1 = g^y \bmod p$ and $c_2 = mh^y \bmod p$.
- Bob sends the ciphertext (c_1,c_2) to Alice.
  - $E(m)= (c_1,c_2) := (g^y \bmod p ,mh^y \bmod p)$

El Gamal Encryption

- Encryption is probabilistic !!!
- This means that a single plaintext can be encrypted to many possible ciphertexts: for same m and different y, E(m) is different !
- So we should write E(m,y)
- A general ElGamal encryption produces a 2:1 expansion in size from plaintext to ciphertext.
- Encryption requires 2 exponentiations (slow!)

El Gamal Decryption

- To decrypt a ciphertext (c_1,c_2) with her private key x, Alice computes:
- $D(c_1,c_2)=c_2(c_1)^{-x} \bmod p$

$$c_2(c_1)^{-x} = mh^y \cdot (g^y)^{-x} = mh^y \cdot g^{-xy} = mg^{xy} \cdot g^{-xy} = m$$

- Remark: knowledge of the random number y is not needed !

El gamal: toy example

Key generation:

Choose prime number $p = 2357$, $g = 2$, private key $x = 1751$ and compute: $h = g^x \bmod p = 2^{1751} \bmod 2357 = 1185$.

Encryption:

to encrypt the message $m = 2035$, choose $y = 1520$ and compute:

$c_1 = g^y \bmod p = 2^{1520} \bmod 2357 = 1430$

$c_2 = m\, h^y \bmod p = 2035 \mathrm{x} 1185^{1520} \bmod 2357 = 697$

Decryption. Compute

$c_1^{-x} = c_1^{p-1-x} = 1430^{605} \bmod 2357 = 872$

$m = c_1^{-x}\, c_2 = 872 * 697 \bmod 2357 = 2035$

Key Management

- Public-key encryption helps addressing secret key distribution problems
- Two aspects of public key methods used in key distribution applications:
  - distribution of public keys
  - use of public-key encryption to distribute secret keys

Distribution of Public Keys

- All proposed solutions can be classified as belonging to one of the following classes:
  - Public announcement
  - Publicly available directory
  - Public-key distribution authority
  - Public-key certificates

Public Announcement

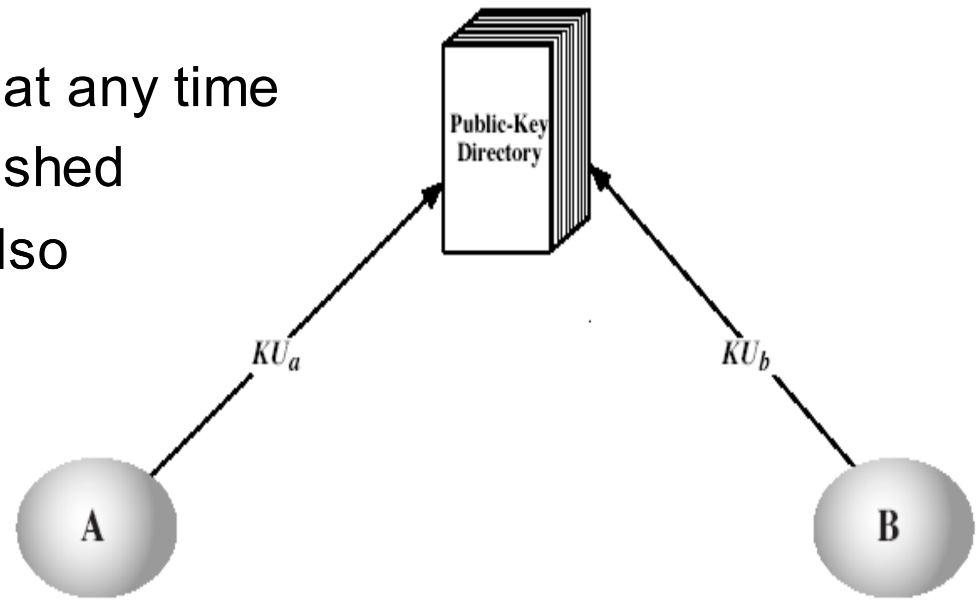
- Users distribute public keys to recipients or broadcast to all the community
  - Append Pretty Good Privacy (PGP) keys to email messages or post to news groups or mailing lists

Public Announcement

- Major weakness is forgery
  - anyone can create a key claiming to be someone else and broadcast it
  - until forgery is discovered can masquerade as claimed user

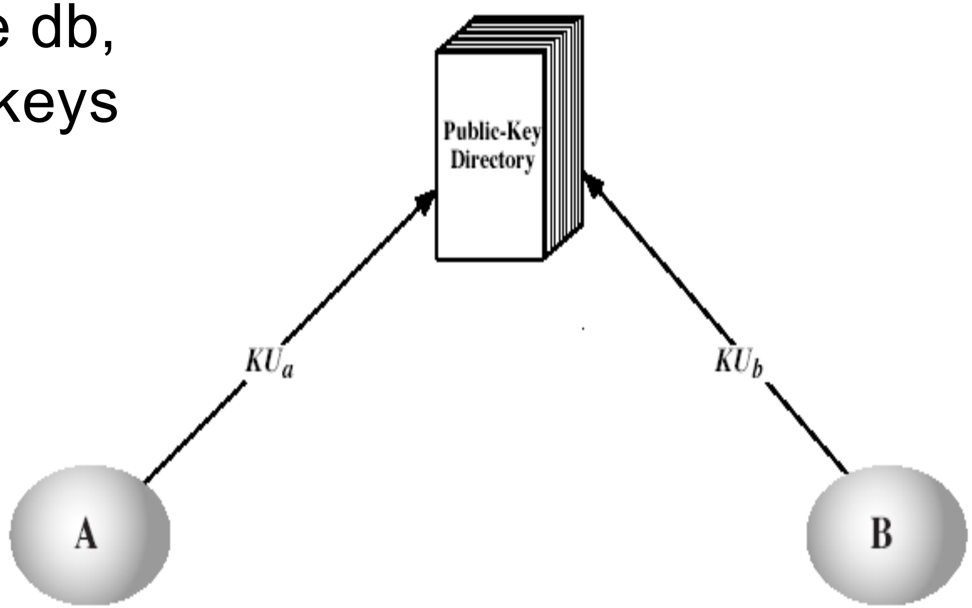
Publicly available directory

- A dynamic and public directory of keys, managed by a trusted organization.
- Properties:
  - it contains {name, public-key} entries
  - participants register *securely* the public key with directory
  - participants can replace key at any time
  - directory is periodically published
  - directory can be accessed also electronically



Publicly available directory

- Greater security by registering keys with a public directory than with announcement
- Still vulnerable to tampering or forgery:
  - if someone can violate the db, can distribute fake public keys



Public-Key Authority

- Improves security by tightening control over distribution of keys from directory: an authority manages the directory.
- Requires users to know public key of the authority
- Then users interact with directory to obtain any desired public key securely
  - does require real-time access to directory when keys are needed
  - Secure interaction with authority can be complicated

Public-Key Certificates

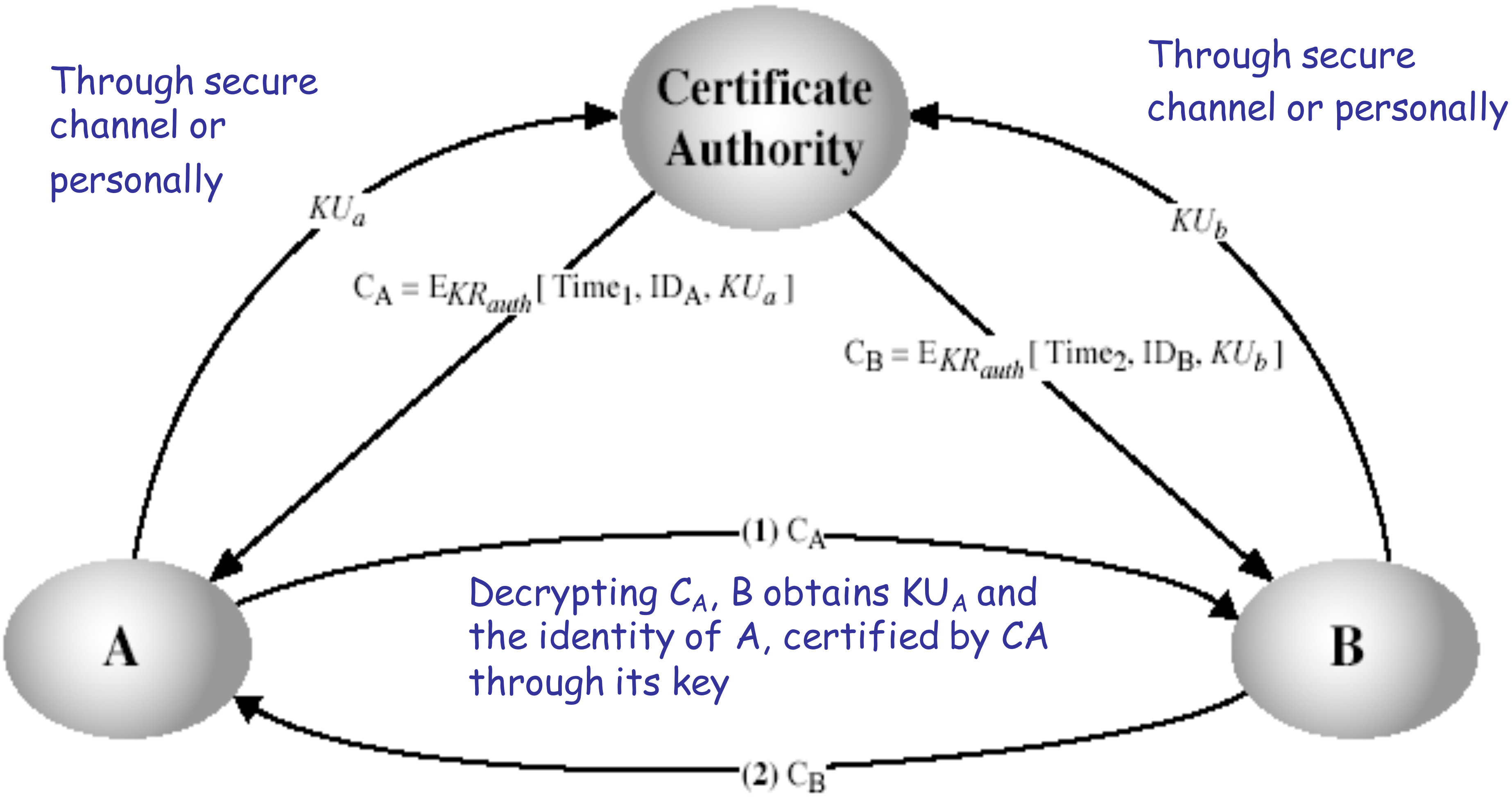
- Certificates allow key exchange without real-time access to public-key authority, but with same reliability
- A certificate binds **identity** to **public key**
  - usually with other info such as period of validity, rights of use, etc
- Created and **signed** by a trusted Certificate Authority (CA), delivered to the user
- To distribute his/her public key, a user sends the certificate

Public-Key Certificates

In this a way:

- each user can read a certificate to determine the name & public key of certificate’s owner;
- every user can verify that the certificate has been created by the CA, if he knows the CA public-key
- only the CA can create or update a certificate.
- every participant can verify that his/her own certificate is updated.

Public-Key Certificates Exchange

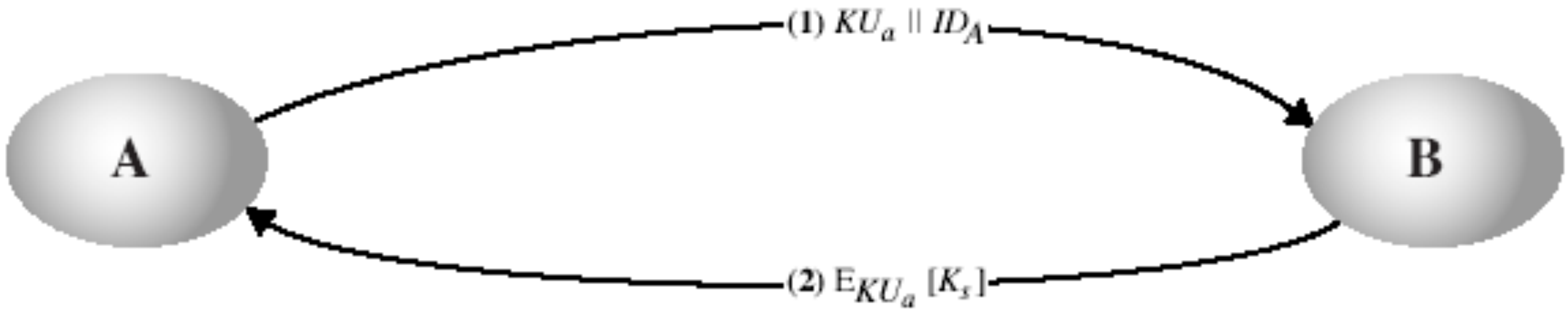


Use of Public-Key to share Secret Keys

- Public-key, obtained with previous methods, can be used for secrecy or authentication
- Public-key algorithms are slow, so usually users prefer to use secret-key encryption.
- A session key is exchanged through a public key protocol.
  - several alternatives for negotiating a suitable session

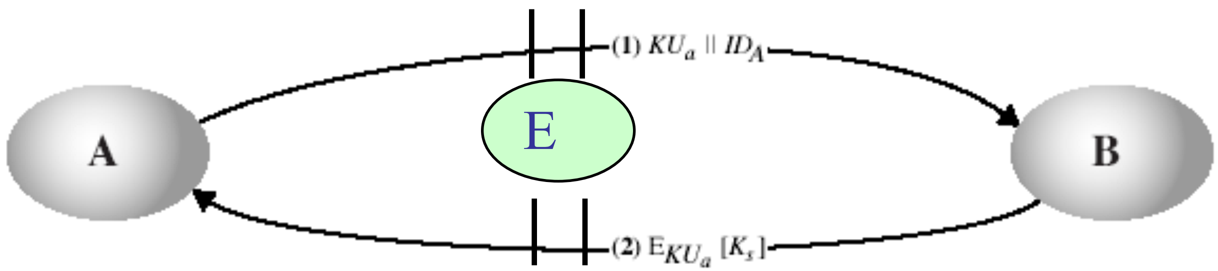
Simple Secret Key Distribution

- A generates a temporary key pair (KU_a, KR_a)
- A sends to B his public key and his identity
- B generates a session key K, sends it to A encrypted using the supplied public key
- A decrypts the session key K_S and both can use it



Simple Secret Key Distribution

- Vulnerable to man in the middle attack: an opponent can intercept and impersonate both users:
  - E can intercept (1), create keys $\{KU_e, KR_e\}$ and send $KU_e \| ID_A$ to B
  - B generates K_s, and send $E_{KUe}[K_s]$ to A
  - E intercepts the message and decrypts it obtaining K_s.
  - E transmits $E_{KUa}[K_s]$ to A
  - Now A and B have K_s, but they don't know that also E knows it, and that he can intercept their messages.



Diffie-Hellman Key Exchange

- First public-key type scheme proposed by Diffie & Hellman in 1976 along with the exposition of public key concepts
  - note: now known that James Ellis (UK CESG) secretly proposed the concept in 1970
- It is a practical method for public exchange of a secret key
- It is used in several commercial products

Diffie-Hellman Key Exchange

- It is a public-key based key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key known only to the two participants
- It is based on exponentiation modulo a prime - easy to do
- Security relies on the difficulty of computing discrete logarithms

Discrete Logarithm

- Given a prime number p:
  - *Primitive root of p* = a number whose powers (mod p) generate all the integers between 1 and p-1:
  - a mod p, a^2 mod p, a^3 mod p, …, a^{p-1} mod p are distinct and are a permutation of all the integers 1 ... p-1
- Given an integer b, and a primitive root of p, we define discrete logarithm of *b* for the base *a mod p*, the unique number i such that

  $b = a^i \bmod p, \quad 0 \leq i \leq p-1$

Diffie-Hellman Setup

- All users agree on global public parameters:
  - q: large prime integer
  - *a*: primitive root mod q
- Each user generates his/her pair of keys:
  - A randomly chooses a private key (integer number): $x_A < q$
  - Computes the **public key**: $y_A = a^{x_A} \bmod q$
  - A makes y_A public and keeps x_A secret
  - B does the same obtaining x_B and y_B

Diffie-Hellman Key Exchange

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

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

  $= y_A^{x_B} \bmod q$ (which **B** can compute by himself)

  $= y_B^{x_A} \bmod q$ (which **A** can compute by herself)

- K_{AB} is used as session key in a secret-key encryption scheme between Alice and Bob
- if Alice and Bob subsequently communicate, they will have the **same** key as before, unless they choose new public-keys.
- Attacker needs to know one between x_A or x_B, but this implies solving a discrete log problem

Diffie-Hellman Example

- Alice & Bob wish to share a secret key:
- They agree on prime $q=353$ and $a=3$
- Select random secret keys:
  - A chooses $x_A=97$, B chooses $x_B=233$
- Then compute public keys:
  - $y_A=3^{97} \bmod 353 = 40$ (Alice)
  - $y_B=3^{233} \bmod 353 = 248$ (Bob)
- Compute shared session key as:
  - $K_{AB}= y_B^{x_A} \bmod 353 = 248^{97} = 160$ (Alice)
  - $K_{AB}= y_A^{x_B} \bmod 353 = 40^{233} = 160$ (Bob)

Diffie-Hellman Example

- An attacker knows $q=353$, $a=3$, $y_A=40$, $y_B=248$
- The brute force attack consists in computing the exponentiation $3^x \bmod 353$, stopping when the result is 40 or 248 .
- The first result is $x = 97$
- The complexity is linear in the size of q (exponential in the number of bits k)
- With very big numbers ($k = 1024$) it is difficult!

References

- W. Stallings, *Crittografia e Sicurezza delle Reti*, Mc Graw Hill
  - (chapters 9,10)
- A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press
  - (chapter 8)
- RSA FAQs