Public Key Cryptography

Jibi Abraham

COEP TECHNOLOGICAL UNIVERSITY

Shivajinagar, Pune-411 005

(A Unitary Technological University of Govt. of Maharashtra)

Public-Key Cryptography

- Probably most significant advance in the 3000 years history of cryptography
- Public invention due to Whitfield Diffie & Martin Hellman at Stanford Uni in 1976
  - known earlier in the classified community
- Developed to address two key issues:
  - **key distribution** – how to have secure communications in general without having to trust a KDC with your key
  - **digital signatures** – how to verify a message comes intact from the claimed sender

Public-Key Cryptography

- **Public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**
- **Asymmetric** because
  - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures

Comparison

Symmetric	Asymmetric
Same algorithm with the same key is used for encryption and decryption	Same algorithm is used with a pair of keys, one for encryption and one for decryption
Key must be kept secret	One of the two keys must be kept secret
It must be impossible or at least impractical to decipher a message if no other information is available	It must be impossible or at least impractical to decipher a message if no other information is available
Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key	Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key

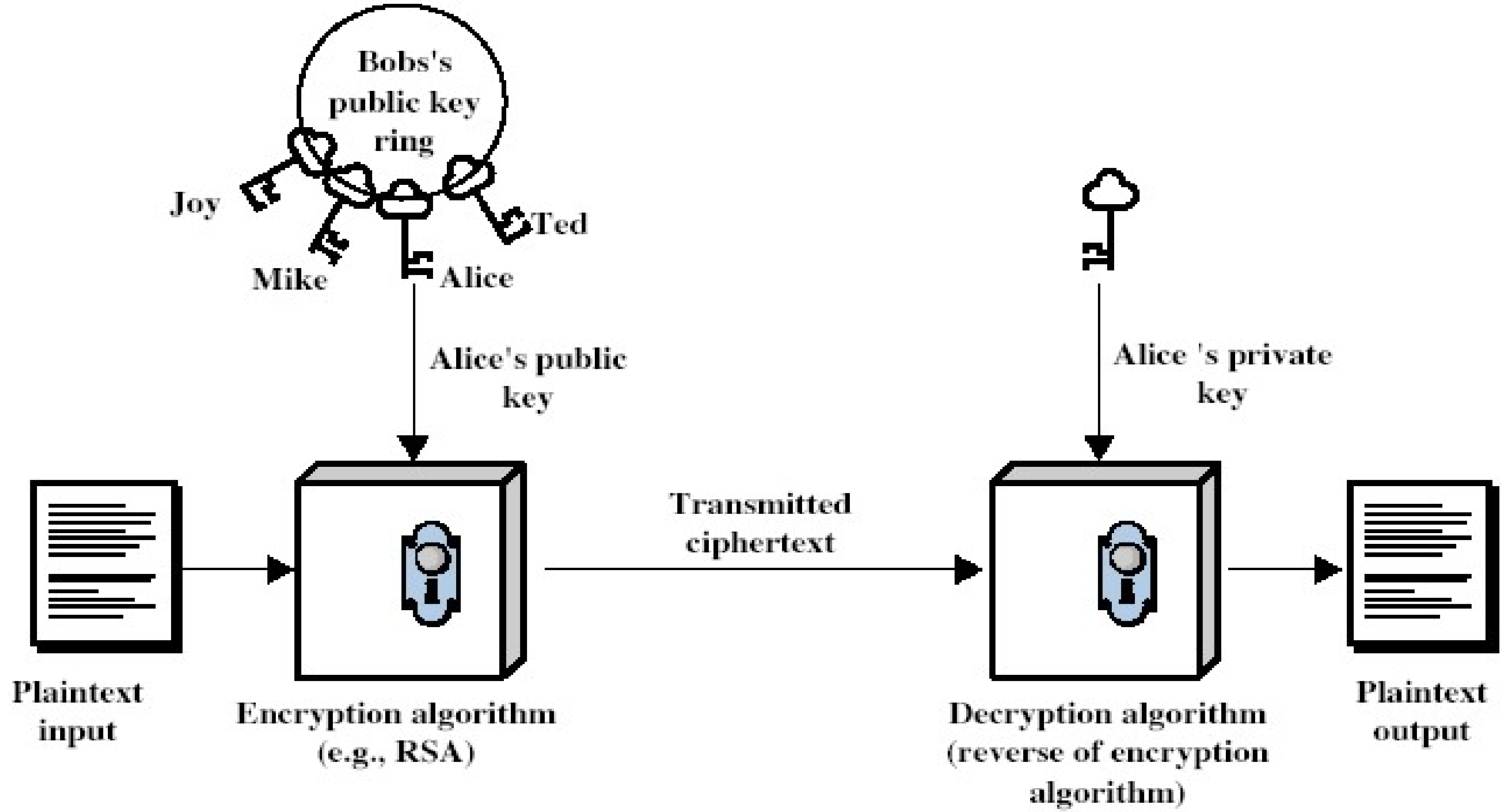
Public-Key Characteristics

- Public-Key algorithms rely on two keys with the characteristics that it is:
  - computationally infeasible to find decryption key knowing only algorithm and encryption key
  - computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
  - either of the two related keys can be used for encryption, with the other used for decryption (in some schemes)

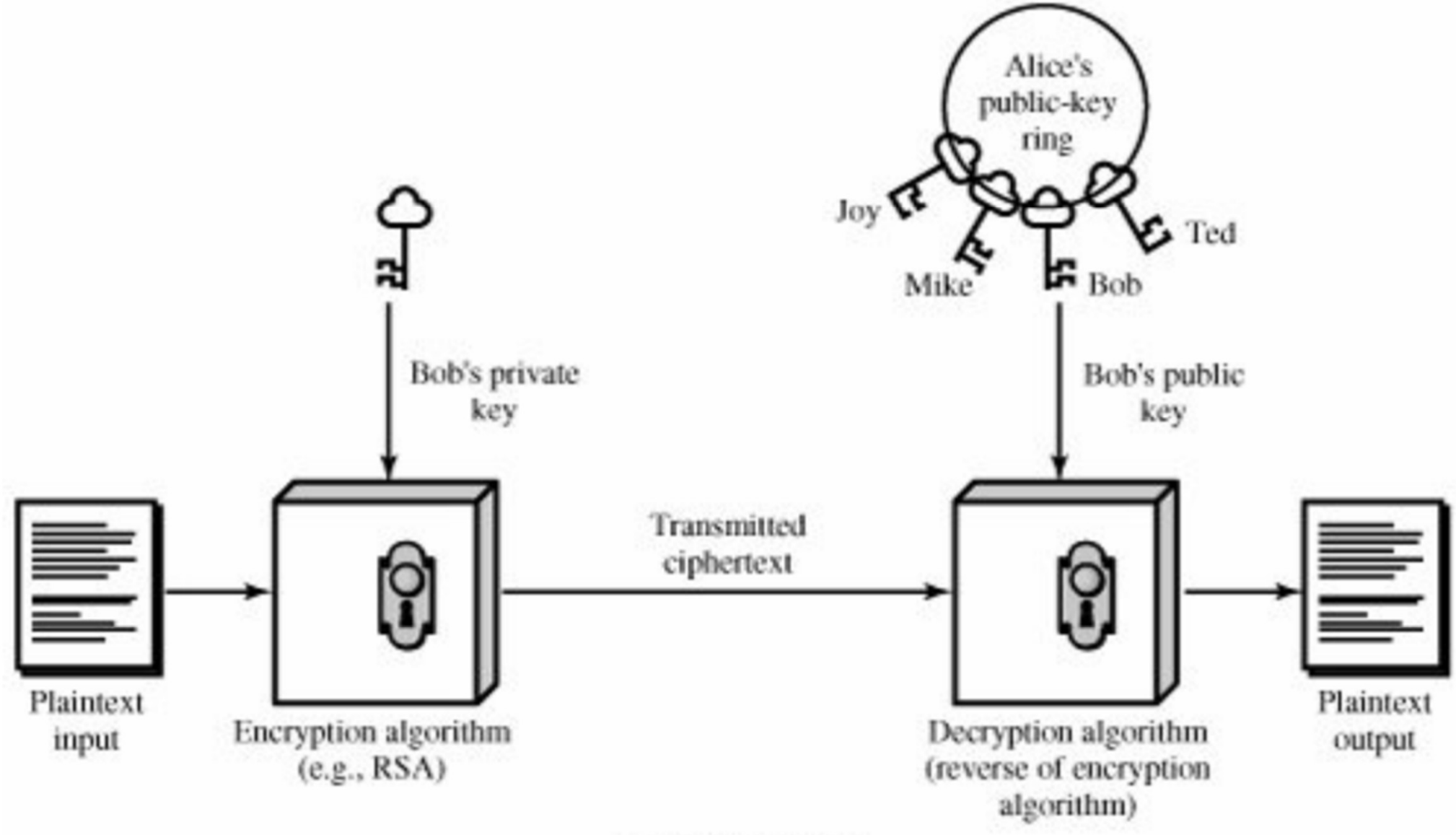
Public-Key Applications

- Can classify uses into 3 categories:
  - **encryption/decryption** (provide secrecy)
  - **digital signatures** (provide authentication)
  - **key exchange** (of session keys)
- Some algorithms are suitable for all uses, others are specific to one of the categories

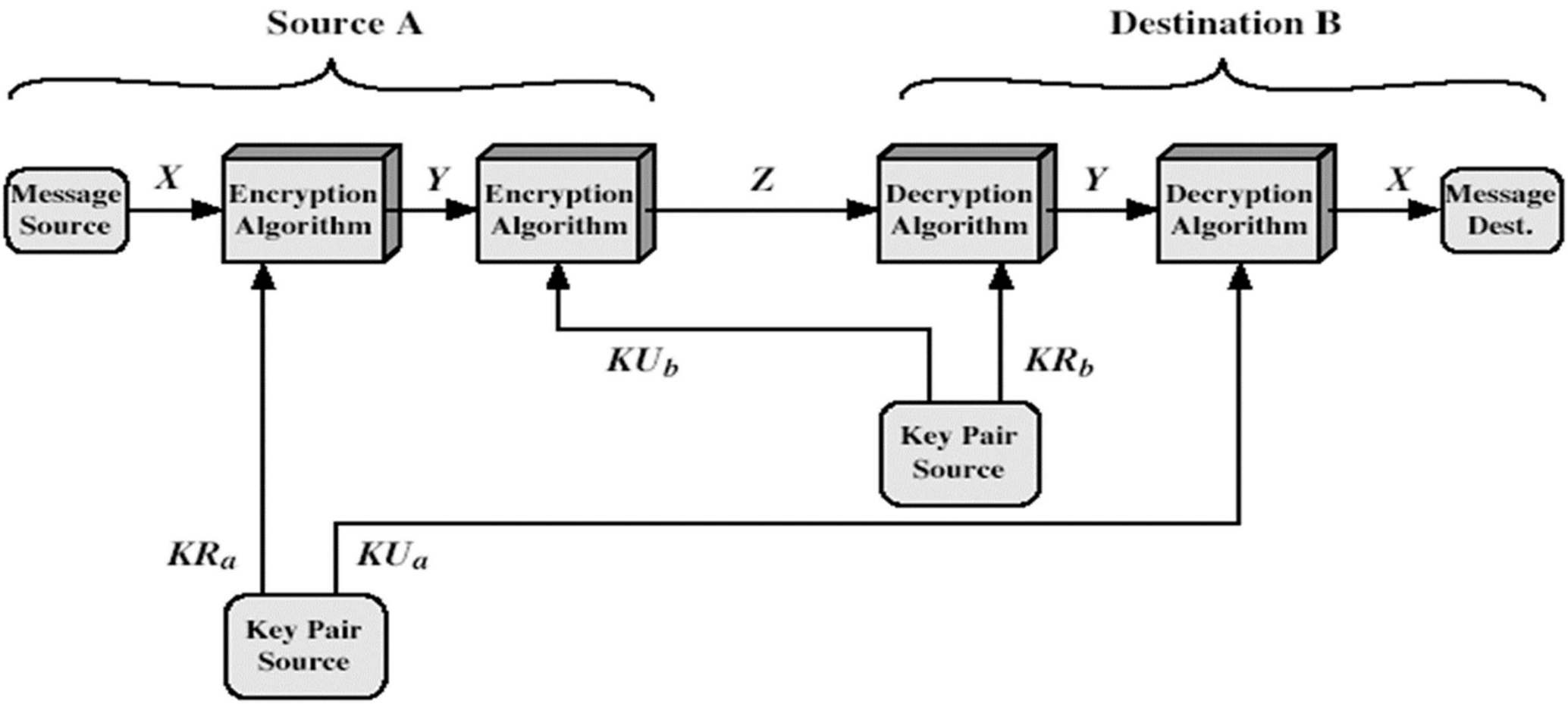
Encryption with Public-Key Cryptography



Authentication with Public-Key Cryptography



Authentication and Secrecy with PKC



Can we do any other order?

Security of Public Key Schemes

- Like private key schemes brute force **exhaustive search** attack is always theoretically possible
- But keys used are too large (>512bits)
- Security relies on a **large enough** difference in difficulty between **easy** (en/decrypt) and **hard** (cryptanalysis) problems
- More generally the **hard** problem is known, it is just made too hard to do in practise
- Requires the use of **very large numbers**
- Hence it is **slow** compared to private key schemes

RSA (Rivest, Shamir, Adleman)

- of MIT in 1977
- Support both public key encryption and digital signature
- Assumption/theoretical basis:
  - Factoring a big number is hard. So Secure
  - Factorization takes $O(e^{\log n \log \log n})$ operations (hard)
- Best known and widely used public-key scheme
- Based on exponentiation in a finite (Galois) field over integers modulo a prime
  - exponentiation takes $O(\log n^3)$ operations (easy)
- Uses large integers (eg. 1024 bits)

RSA Key Setup

- To generate key pair:
  - Pick large primes (>= 256 bits each) *p* and *q*
  - Let *n* = *p*q*, keep your *p* and *q* to yourself
  - For public key, choose *e* that is relatively prime to *ø(n) =(p-1)(q-1),*
    - **public key : KU= <e, *n*>**
  - For private key, find $d = e^{-1} \bmod ø(n)$, i.e., $e*d = 1 \bmod ø(n)$
    - **private key KR = <*d*, *n*>**

Why Does RSA Work?

- Given pub = <e, n> and priv = <d, n>
  - $n = p*q$, $\phi(n) = (p-1)(q-1)$
  - $e*d = 1 \bmod \phi(n)$
  - $x^{e*d} = x \bmod n$
  - encryption: $c = m^e \bmod n$
  - decryption: $m = c^d \bmod n = m^{e*d} \bmod n = m \bmod n = m$ (since $m < n$)
  - digital signature (similar)

RSA Example

Key Generation	
Select p, q	p, q both prime, p≠q
Calculate n = p×q	
Calculate $\phi(n) = (p-1)\times(q-1)$	
Select integer e	$\gcd(\phi(n),e) = 1;\ 1<e<\phi(n)$
Calculate d	
Public key	KU = {e, n}
Private key	KR = {d, n}

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

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

1. Select primes: p=17 and q=11
2. Compute $n = pq$ =17×11=187
3. Compute ø(n)=(p–1)(q-1)=16×10=160
4. Select e : gcd(e,160)=1; choose e=7
5. Public key KU={7,187}
6. Determine d*: de=1 mod 160 and d < 160
7. Value is d=23 since 23×7=161= 10×160+1
8. Private Key KR={23,187}, keep it secret

RSA Example

- Public key KU={7,187}, Private key KR={23,187}
- Plaintext message M = 88 (88<187)
- Encryption:

$C = 88^7 \bmod 187$

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

$88^1 \bmod 187 = 88$

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

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

$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187$

$= 894{,}432 \bmod 187 = 11$

RSA Example

- Public key KU={7,187}, Private key KR={23,187}
- Ciphertext message C = 11
- Decryption: $M = 11^{23} \bmod 187$ (23 = 10111)

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

$11^1 \bmod 187 = 11$ $11^2 \bmod 187 = 121$

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

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

$11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187$

$= 79{,}720{,}245 \bmod 187 = 88$

Encoding of a Message

- Sender converts the message into a number *m*
- One common conversion process uses the ASCII alphabet

A	B	C	D	E	F	G	H	I	J	K	L	M
65	66	67	68	69	70	71	72	73	74	75	76	77
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
78	79	80	81	82	83	84	85	86	87	88	89	90

- message "HELLO" would be encoded as 7269767679

RSA Use

- Given pub = <e, n> and priv = <d, n>
  - encryption: $c = m^e \bmod n$, $m < n$
  - decryption: $m = c^d \bmod n$
  - signature: $s = m^d \bmod n$, $m < n$
  - verification: $m = s^e \bmod n$

Exponentiation

- Can use the direct Square and Multiply Algorithm
- Need a fast, efficient algorithm for exponentiation
- Concept is based on repeatedly squaring base and multiplying in the ones that are needed to compute the result
- Look at the binary representation of the exponent
  - n=5 (101)
    - $7^5 = 7^4.7^1 = 3.7 = 10 \bmod 11$
  - N=129 (10000001)
  - eg. $3^{129} = 3^{128}.3^1 = 5.3 = 4 \bmod 11$

Exponentiation Algorithm

- Algorithm for Computing a^b mod n
- b is expressed as a binary number $b_k b_{k-1} \ldots b_0$

```
c ← 0; d ← 1
for i ← k downto 0
    do
        d ← (d × d) mod n
        if   b_i = 1
            then
                d ← (d × a) mod n
return d
```

Example

- To find M^{250}
- 250 = 11111010 k =7

i	e_i	Step 2a	Step 2b
7	1	M	M
6	1	$(M)^2 = M^2$	$M^2 \cdot M = M^3$
5	1	$(M^3)^2 = M^6$	$M^6 \cdot M = M^7$
4	1	$(M^7)^2 = M^{14}$	$M^{14} \cdot M = M^{15}$
3	1	$(M^{15})^2 = M^{30}$	$M^{30} \cdot M = M^{31}$
2	0	$(M^{31})^2 = M^{62}$	M^{62}
1	1	$(M^{62})^2 = M^{124}$	$M^{124} \cdot M = M^{125}$
0	0	$(M^{125})^2 = M^{250}$	M^{250}

- Number of multiplication = 7+5

Features of RSA components

- Users of RSA must:
  - determine two primes at random - p, q
  - select either e or d and compute the other
- Primes p and q must not be easily derived from modulus n=p.q
  - means must be sufficiently large
  - typically guess and use probabilistic test
- Exponents e, d are inverses, so use the Inverse algorithm to compute the other

RSA Security

- Three approaches to attacking RSA:
  - Brute force key search (infeasible given size of numbers)
  - Mathematical attacks (based on difficulty of computing ø(N), by factoring modulus N)
  - Timing attacks (on running of decryption)

Factoring Problem

- Mathematical approach takes 3 forms:
  - pub = <*e, n*> $d = e^{-1}$ mod $\varnothing(N)$
  - factor n=p.q, hence find $\varnothing(N)$ and then d
  - Determine $\varnothing(N)$ directly and find d
  - Find d directly
- Currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - as of Aug-99 best is 130 decimal digits (512) bit with GNFS
  - biggest improvement comes from improved factoring algorithms from “Quadratic Sieve” to “Generalized Number Field Sieve”
  - barring dramatic breakthrough 1024+ bit RSA secure
    - ensure p, q of similar size and matching other constraints

Timing Attacks

- Developed in mid-1990's
- Exploit timing variations in operations
  - eg. multiplying by small vs large number
  - or IF's varying which instructions executed
- Infer operand size based on time taken
- RSA exploits time taken in exponentiation
- Countermeasures
  - use constant exponentiation time
  - add random delays
  - blind values used in calculations like padding

Is RSA Secure?

- Factoring 512-bit number is very hard!
- But if you can factor big number *n* then given public key <*e*,*n*>, you can find *d*, hence the private key by:
  - Knowing factors *p,* q, such that, n = *p*q*
  - Then *ø(n) =(p-1)(q-1)*
  - Then *d* such that *e*d* = 1 mod *ø(n)*
- Threat
  - Moore's law
  - Refinement of factorizing algorithms
  - Quantum Cryptography
- For the near future, a key of 1024 or 2048 bits needed

Shor's Quantum Algorithm

- One way to crack RSA encryption is by factoring n
- No classical algorithm is known that can factor in polynomial time.
- **Shor's algorithm** is a quantum factoring algorithm with $O((\log n)^3)$ time (polynomial) and $O(\log n)$ space
- Algorithm is significant because it implies that RSA might be easily broken, given a sufficiently large quantum computer
- To factorize an integer 2048 bits long used as an RSA key, the Shor algorithm needs to be run on a quantum computer with millions of qubits, which will take decades to build

Symmetric (DES) vs. Public Key (RSA)

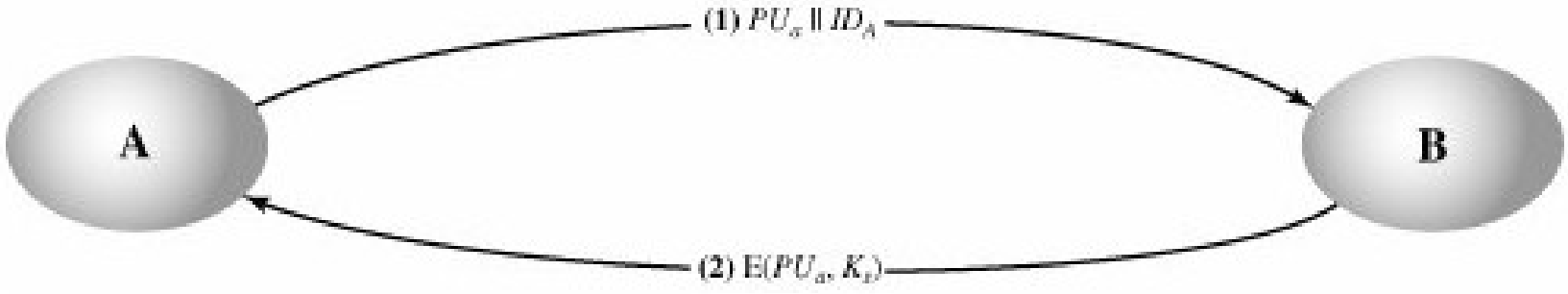
- Exponentiation of RSA is expensive !
- AES and DES are much faster
  - 100 times faster in software
  - 1,000 to 10,000 times faster in hardware
- RSA often used in combination in AES and DES
  - Pass the *session key* with RSA

Public-Key Distribution of Secret Keys

- Public-key algorithms are slow in encryption and decryption
- So few users make exclusive use of public key for message encryption.
- Usually prefer private-key encryption to protect message contents
- Private-key encryption need a session key to be shared by the communicating parties
- Public-Key encryption helps in the distribution of Secret Keys between the communicating parties

Simple Secret Key Distribution

- Proposed by Merkle in 1979



  - A generates a new temporary public-private key pair {PU_a, PR_a}
  - A sends B the public key and A's identity
  - B generates a session key K_s sends it to A encrypted using the supplied public key
  - A decrypts the session key and both use

- Problem is that an opponent can intercept and impersonate both halves of protocol

Simple Secret Key Distribution

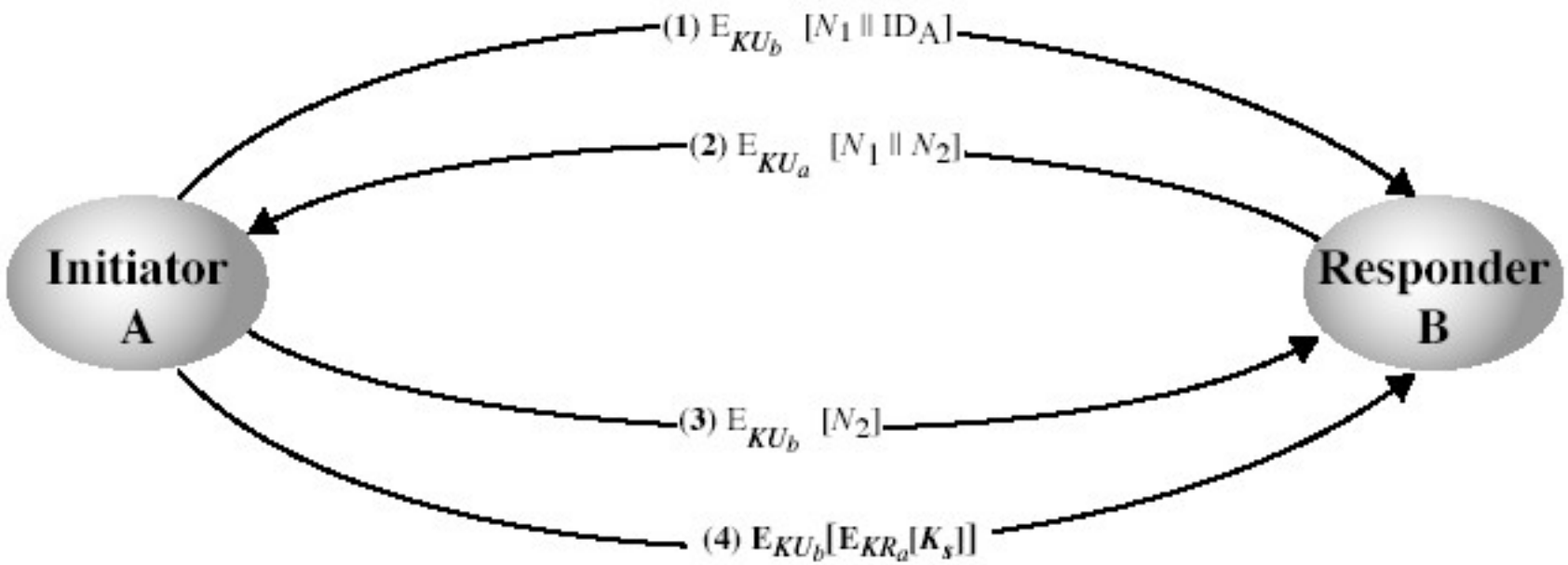
- Man-in-the-middle attack

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message intended for B consisting of PU_a and an identifier of A, ID_A.
2. E intercepts the message, creates its own public/private key pair $\{PU_e, PR_e\}$ and transmits $PU_e||ID_A$ to B
3. B generates a secret key, K_s, and transmits $E(PU_e, K_s)$.
4. E intercepts the message and learns K_s by computing $D(PR_e, E(PU_e, K_s))$.
5. E transmits $E(PU_a, K_s)$ to A

- The result is that both A and B know K_s and are unaware that K_s has also been revealed to E. A and B can now exchange messages using K_s E simply eavesdrops.

Public-Key Distribution of Secret Keys

- First securely exchanged public keys using a previously established method



Diffie-Hellman Key Exchange

- First public-key scheme proposed
  - to establish a common secret key known only to the two participants
  - No prior knowledge like session key or public key
- by Diffie and Hellman in 1976 along with the exposition of public key concepts
  - note: now know that James Ellis (UK CESG) secretly proposed the concept in 1970
- Used in a number of commercial products
- Based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) - easy
- Security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard

Diffie-Hellman Key Setup

- Both users agree on global parameters:
  - Large prime integer or polynomial q
  - α a primitive root mod q
- Each user (eg. A) generates their private key, public key
  - chooses a **private key**: $x_A < q$
  - compute their **public key**: $y_A = \alpha^{x_A} \bmod q$
- Each user makes public that key y_A

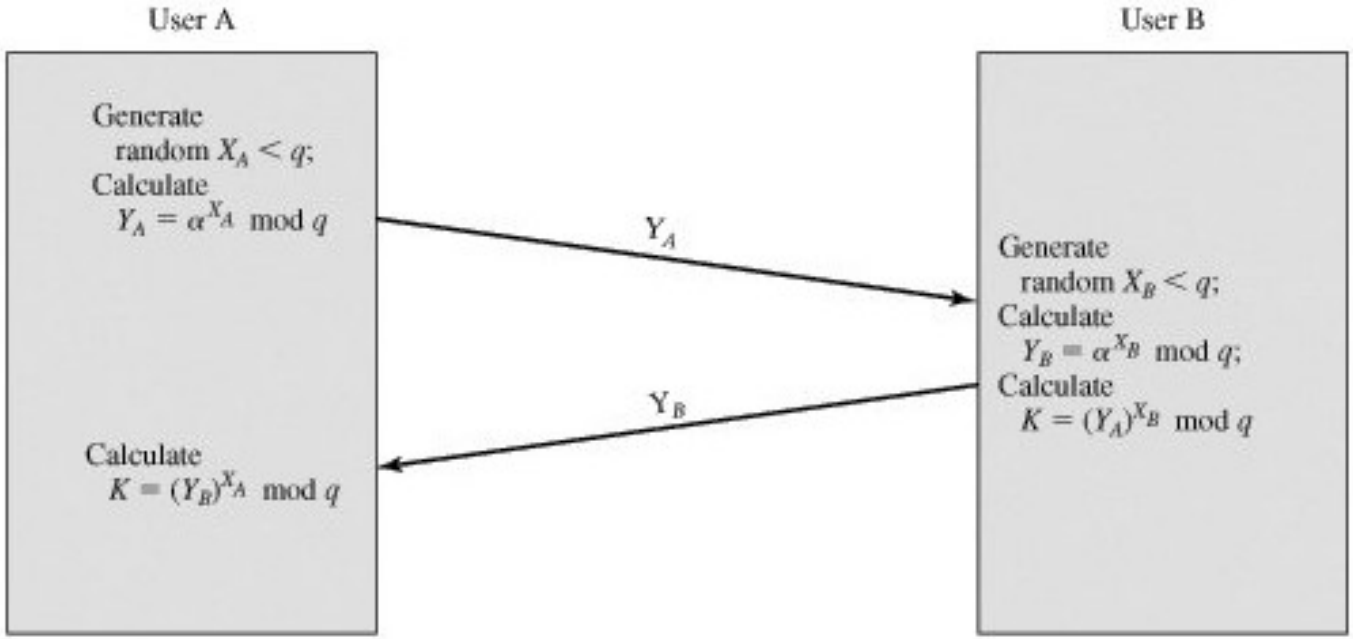
Diffie-Hellman Key Exchange

- (Pub Key, Pr Key): A – (y_A, x_A) B – (y_B , x_B)
- Let the shared key for users A and B be K:

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

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

- To get K, the attacker needs either x_A or x_B, must solve a discrete log



COEP Tech

TY

Algorithm

Global Public Elements

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

User A Key Generation

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

User B Key Generation

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

Calculation of Secret Key by User A

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

Calculation of Secret Key by User B

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

COEP Tech

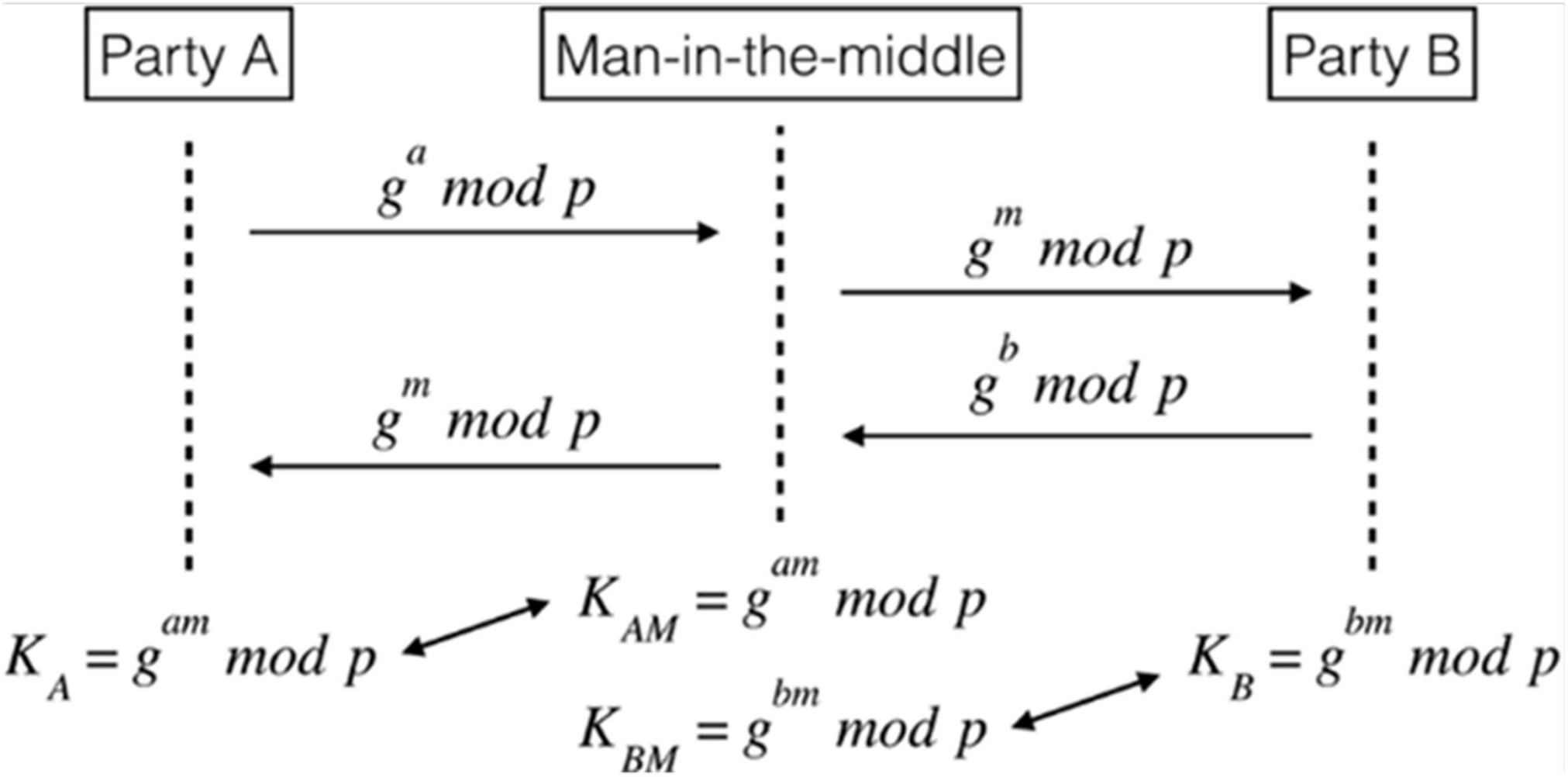
Diffie-Hellman Example

- Alice and Bob agree on prime q=353 and α=3
- Select random secret keys:
  - A chooses x_A=97, B chooses x_B=233
- 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)

Man-in-the-Middle Attack



Man-in-the-Middle Attack

- Suppose Alice and Bob wish to exchange keys and Eve is the adversary. The attack proceeds as follows:
  - Eve prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}.
  - Alice transmits Y_A to Bob.
  - Eve intercepts Y_A and transmits Y_{D1} to Bob. Eve also calculates $K2 = (Y_A)^{X_{D2}} \bmod q$.
  - Bob receives Y_{D1} and calculates $K1 = (Y_{D1})^{X_E} \bmod q$.
  - Bob transmits X_A to Alice.
  - Eve intercepts X_A and transmits Y_{D2} to Alice. Eve calculates $K1 = (Y_B)^{X_{D1}} \bmod q$.
  - Alice receives Y_{D2} and calculates $K2 = (Y_{D2})^{X_A} \bmod q$.

Man-in-the-Middle Attack

- At this point, Bob and Alice think that they share a secret key, but instead Bob and Eve share secret key K1 and Alice and Eve share secret key K2.
- All future communication between Bob and Alice is compromised in the following way:
  - Alice sends an encrypted message M: E(K2, M).
  - Eve intercepts the encrypted message and decrypts it, to recover M.
  - Eve sends Bob E(K1, M) or E(K1, M'), where M' is any message. In the first case, Eve simply wants to eavesdrop on the communication without altering it. In the second case, Eve wants to modify the message going to Bob.
- The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates

Computational Problems of DH

- Publish large prime p, primitive root α
- Alice's secret exponent: x
- Bob's secret exponent: y
  - $0 < x,y < p-1$
- Alice sends $\alpha^x \pmod p$ to Bob
- Bob sends $\alpha^y \pmod p$ to Alice
- Each know key $K=\alpha^{xy}$
- Eve sees α, p, α^x , α^y ; why can't she determine α^{xy}?

- Which is hardest?

1. *Discrete logs:*
   "Given $a^x = b \pmod p$, find x

2. *Computational Diffie-Hellman problem:*
   "Given a, p, $a^x \pmod p$, $a^y \pmod p$, find $a^{xy} \pmod p$"

3. *Decision Diffie-Hellman problem:*
   "Given a, p, $a^x \pmod p$, $a^y \pmod p$, and any $c \neq 0 \pmod p$.
   Verify that $c=a^{xy} \pmod p$"

Distribution of Public Keys

- Can be considered as using one of:
  - Public announcement
  - Publicly available directory
  - Public-key authority
  - Public-key certificates

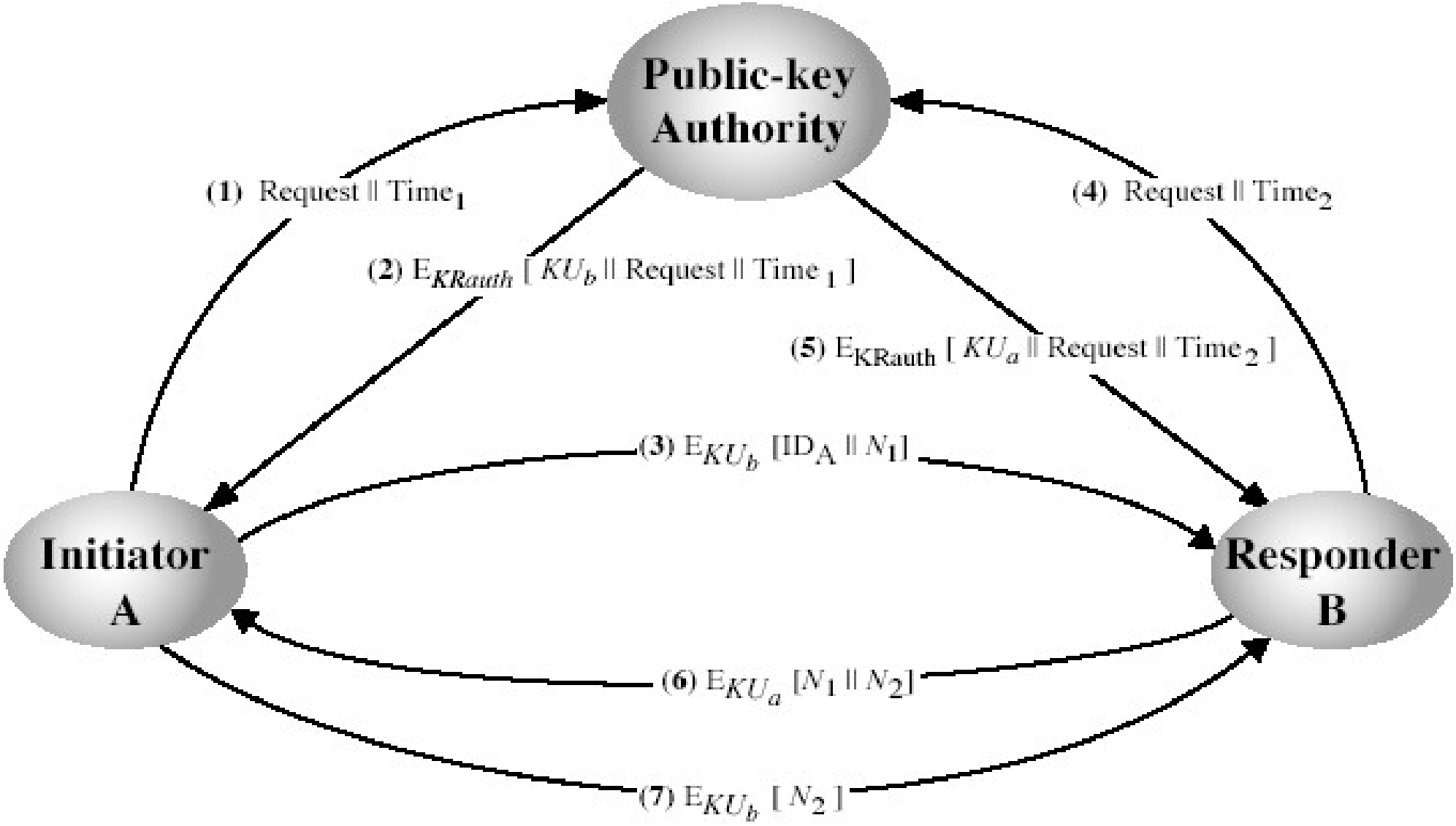
Public Announcement

- Users distribute public keys to recipients or broadcast to community at large
  - eg. append PGP keys to email messages or post to news groups or email list
- 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 for authentication

Publicly Available Directory

- Can obtain greater security by registering keys with a public directory
- Directory must be trusted with properties:
  - contains {name, public-key} entries
  - participants register securely with directory
  - participants can replace key at any time
  - directory is periodically published
  - directory can be accessed electronically
- Still vulnerable to tampering or forgery

Public-Key Authority



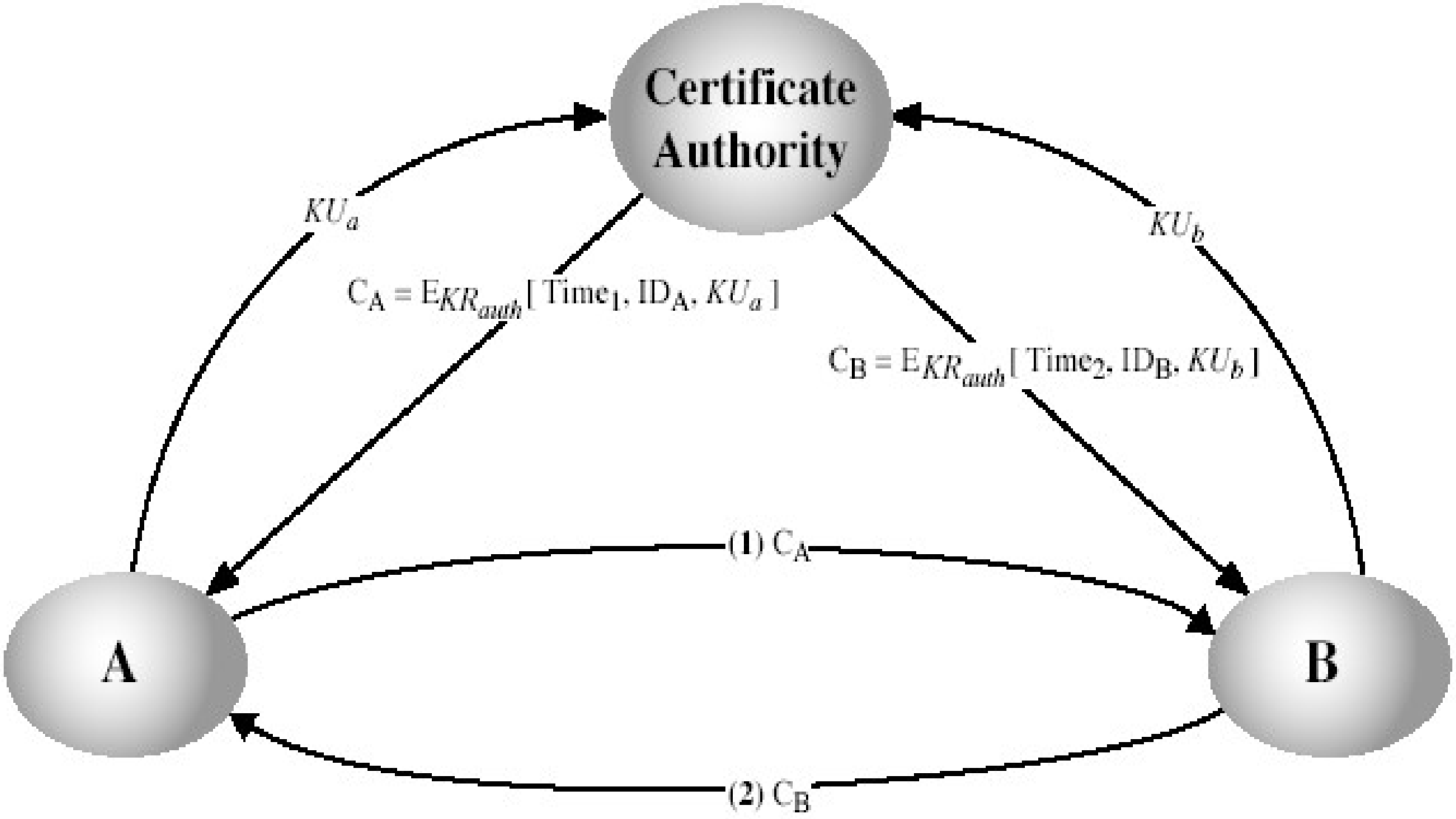
Public-Key Authority

- Improve security by tightening control over distribution of keys from directory
- Requires users to know public key for the directory
- Then users interact with directory to obtain any desired public key securely
  - does require real-time access to directory when keys are needed
- Drawbacks
  - Public Key authority could be a bottleneck
  - Public keys maintained by the authority is vulnerable to tampering

Public-Key Certificates

- Certificates allow key exchange without real-time access to public-key authority
- A certificate binds **identity** to **public key**
- With all contents **signed** by a trusted Public-Key or Certificate Authority (CA)
  - Certifies the identity
  - Only the CA can make the certificates
- Certificate can be verified by using CA's public key.
- Like a credit card, cancels the certificate upon expiry or compromise of private key

Public-Key Certificates



ElGamal Cryptosystem

- ElGamal Cryptosystem is an entire public-key cryptosystem like RSA, but based on discrete logs
- Bob chooses prime p, primitive root α, private key *x*
  - P is large, hence secure and > w = plain text message
  - Bob computes $\beta \equiv \alpha^x \pmod p$
  - Bob publishes (α, p, β) and holds the secret *x*
- Example:-
- p = 11, α = 2 and x = 3
- $\beta \equiv \alpha^x \pmod p = 2^3 \bmod 11 = 8$
- public key: (α, p, β)= (2,11, 8) and private key: x = 3

ElGamal Key Encryption

- Alice chooses secret k, plain text w
- Computes and sends to Bob the pair (a, b) where
  - $a \equiv \alpha^k \pmod p$
  - $b \equiv \beta^k w \pmod p$
- Example:-
- Let k= 4 and w = 7
- public key: $(\alpha, p, \beta) = (2, 11, 8)$
- $a = \alpha^k \pmod p = 2^4 \bmod 11 = 5$
- $b \equiv \beta^k w \pmod p = 2^8 \cdot 7 \bmod 11 = 6$
- Hence Cipher Text is (5, 6)

ElGamal Key Decryption

- Bob received the pair (a, b)
- Compute the plain text

$$w = \frac{b}{a^x} \bmod p = ba^{-x} \bmod p.$$

- Example:-
- (a, b) = (5, 6), p = 11 and d = 3
- $w = (6/5^3) \bmod 11$
- $w = (6/4) \bmod 11$ ($4^{-1} \bmod 11 = 3$)
- $w = (6.3) \bmod 11 = 7$