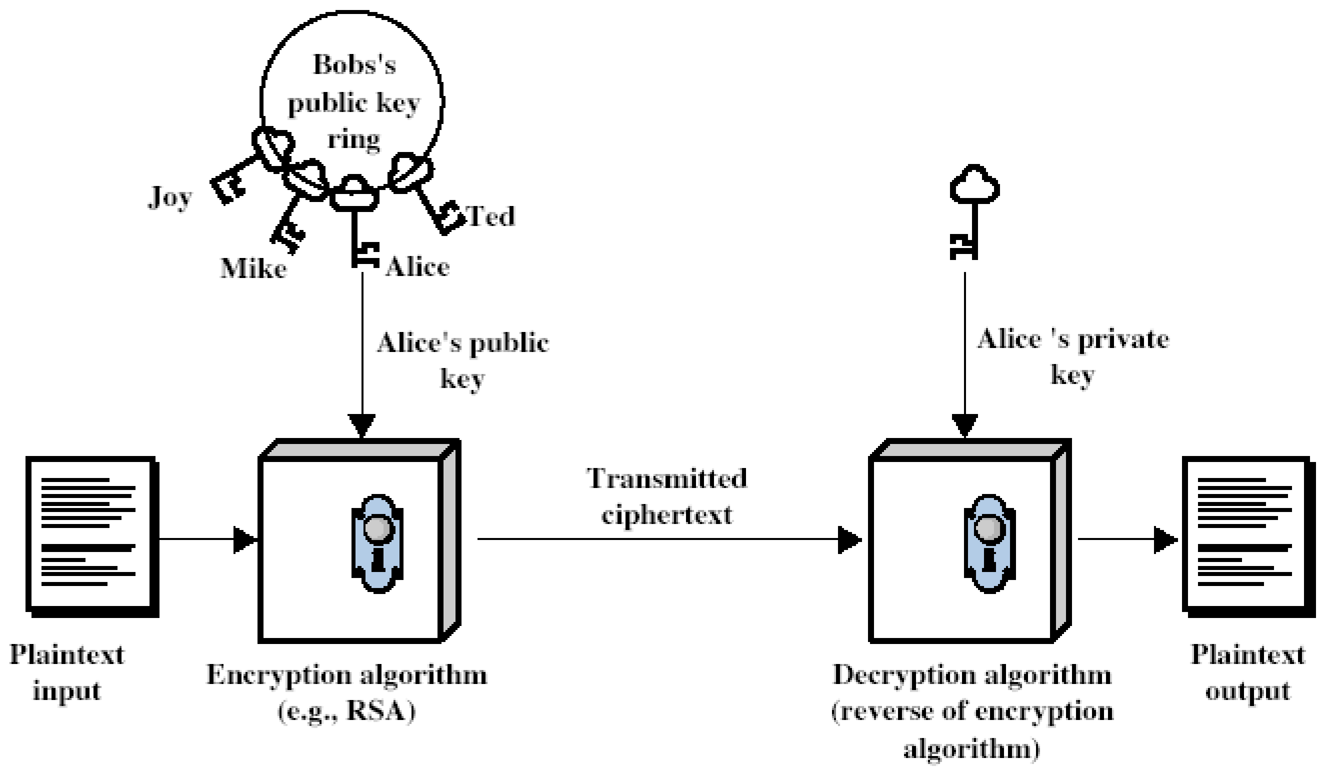
Public-Key Cryptography

- Uses **two** keys – a public & a private key
- **Asymmetric** since parties are **not** equal
- Uses clever application of number theoretic concepts
- Complements **rather than** replaces secret key crypto

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**
- is **asymmetric** because
  - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures

Public-Key Cryptography



Why Public-Key Cryptography?

- 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 invention due to Whitfield Diffie & Martin Hellman at Stanford in 1976
  - known earlier in classified community

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 & 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 Cryptosystems

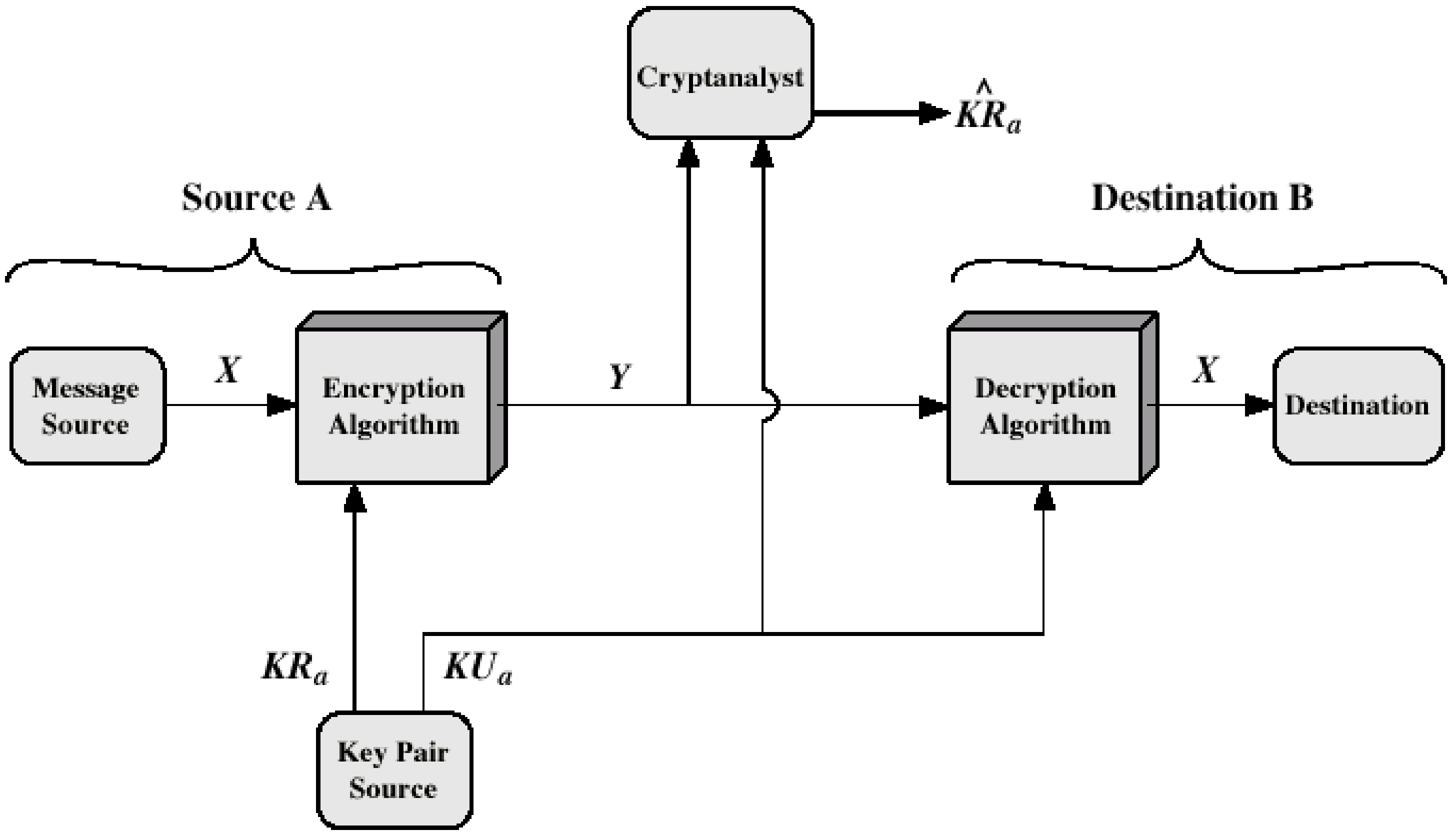


Figure 9.3 Public-Key Cryptosystem: Authentication

Public-Key Cryptosystems

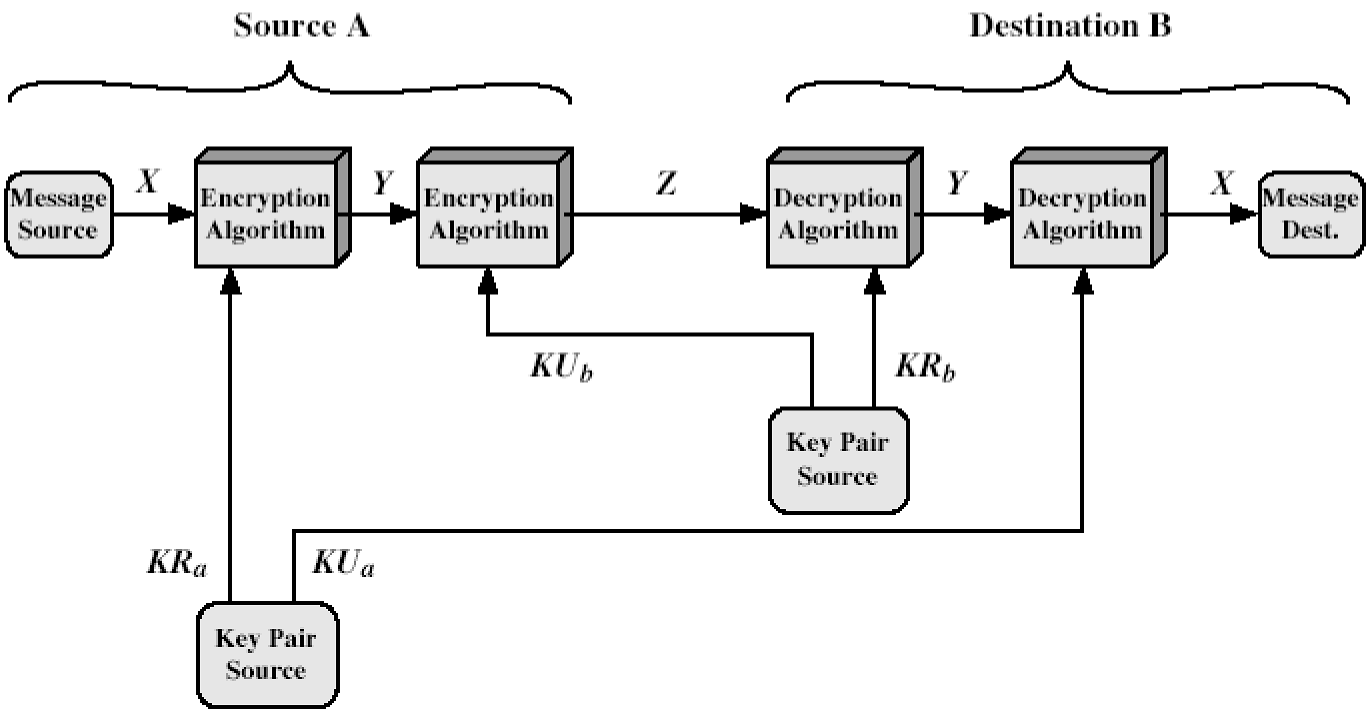


Figure 9.4 Public-Key Cryptosystem: Secrecy and Authentication

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

Security of Public Key Schemes

- like private key schemes brute force **exhaustive search** attack is always theoretically possible
- but keys used are too large (>512 bits)
  - not comparable to symmetric key sizes
- security relies on a **large enough** difference in difficulty between **easy** (en/decrypt) and **hard** (to cryptanalyze) problems
- more generally the **hard** problem is known, its just made too hard to do in practice
- requires the use of **very large numbers**
- hence is **slow** compared to secret key schemes

RSA

- by Rivest, Shamir & Adleman of MIT in 1977
  - patent expired in September 2000
- best known & widely used public-key scheme
- based on modular exponentiation
  - exponentiation takes $O((\log n)^3)$ bit operations (easy)
  - still, 1000 times slower than DES (hardware); 100 times slower in software
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
  - nb. factorization takes $O(e^{\log n \log \log n})$ operations (hard)

RSA Key Setup

- each user generates a public/private key pair by:
- selecting two large primes at random - p, q
- computing their system modulus $N=pq$
  - note $\varnothing(N)=(p-1)(q-1)$
- selecting the encryption key e
  - where $1<e<\varnothing(N)$, $\gcd(e,\varnothing(N))=1$
- solve following equation to find decryption key d
  - $ed=1 \bmod \varnothing(N)$ and $0\leq d\leq N$
- publish their public encryption key: KU={e,N}
- keep secret private decryption key: KR={d,p,q}

RSA Use

- to encrypt a message M the sender:
  - obtains **public key** of recipient $KU=\{e,N\}$
  - computes: $C=M^e \bmod N$, where $0 \leq M<N$
- to decrypt the ciphertext C the owner:
  - uses their private key $KR=\{d,p,q\}$
  - computes: $M=C^d \bmod N$
- note that the message M must be smaller than the modulus N (block if needed)

RSA Example

1. Select primes: p=17 & q=11
2. Compute $n = pq$ =17×11=187
3. Compute $\varnothing(n)=(p-1)(q-1)$=16×10=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 23×7=161= 10×160+1
6. Publish public key KU={7,187}
7. Keep secret private key KR={23,17,11}

RSA Example cont

- sample RSA encryption/decryption is:
- given message `M = 88` (nb. `88<187`)
- encryption:

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

- decryption:

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

Exponentiation

- can use the Square and Multiply Algorithm
- 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 binary representation of exponent
- only takes $O(\log_2 n)$ multiples for number n
  - eg. $7^5 = 7^4(7^1) = 3(7) = 10 \bmod 11$
  - eg. $3^{129} = 3^{128}(3^1) = 5(3) = 4 \bmod 11$

Exponentiation

```
c ← 0; d ← 1
for i  ← k downto 0
        do   c ← 2 × c
             d ← (d × d) mod n
             if     b_i = 1
                    then    c ←  c + 1
                            d ← (d × a) mod n
return d
```

RSA 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 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 Inverse algorithm to compute the other

RSA Security

- four 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)

Timing Attacks

- developed in mid-1990's
- exploit timing variations in operations
  - eg. multiplying by small vs large number
  - or faults 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

Key Management

- public-key encryption helps address key distribution problems
- have two aspects of this:
  - distribution of public keys
  - use of public-key encryption to distribute secret keys

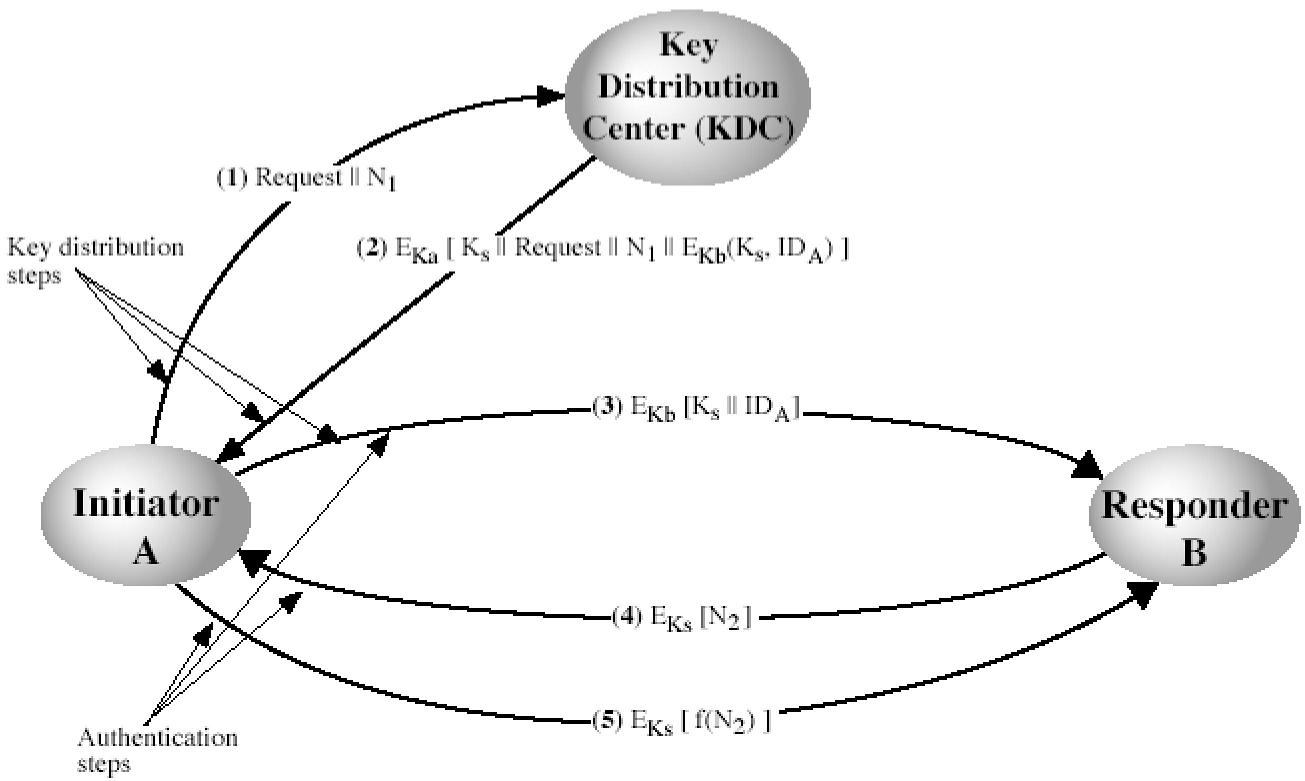
Key Distribution

- symmetric schemes require both parties to share a common secret key
- issue is how to securely distribute this key
- often secure system failure due to a break in the key distribution scheme

Key Distribution

- given parties A and B have various **key distribution** alternatives:
  1. A can select key and physically deliver to B
  2. third party can select & deliver key to A & B
  3. if A & B have communicated previously can use previous key to encrypt a new key
  4. if A & B have secure communications with a third party C, C can relay key between A & B

Key Distribution Scenario



Key Distribution Issues

- hierarchies of KDC's required for large networks, but must trust each other
- session key lifetimes should be limited for greater security
- controlling purposes keys are used for
  - lots of keys to keep track of
  - binding management information to key

Random Numbers

- many uses of **random numbers** in cryptography
  - Nonces in authentication protocols to prevent replay
  - session keys
  - public key generation
  - keystream for a one-time pad
- in all cases its critical that these values be
  - statistically random
    - with uniform distribution, independent
  - unpredictable: cannot infer future sequence on previous values

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

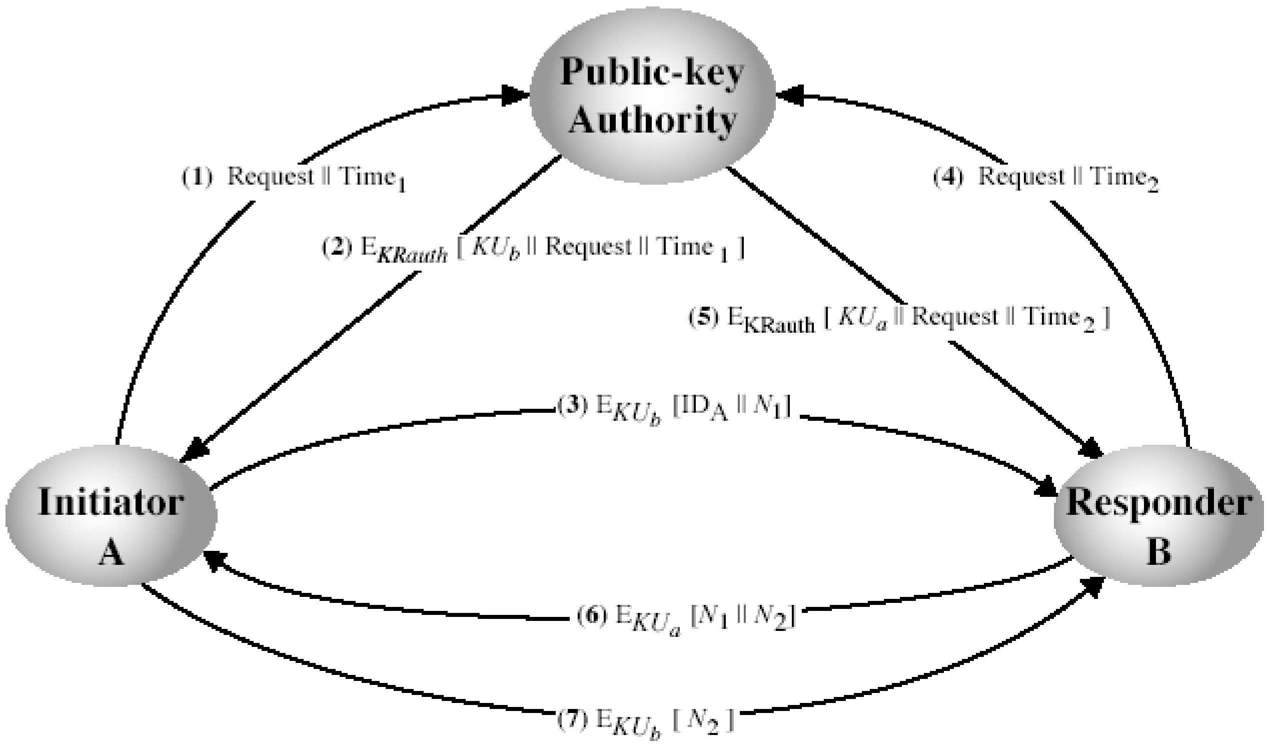
Publicly Available Directory

- 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

- improve security by tightening control over distribution of keys from directory
- has properties of directory
- and 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

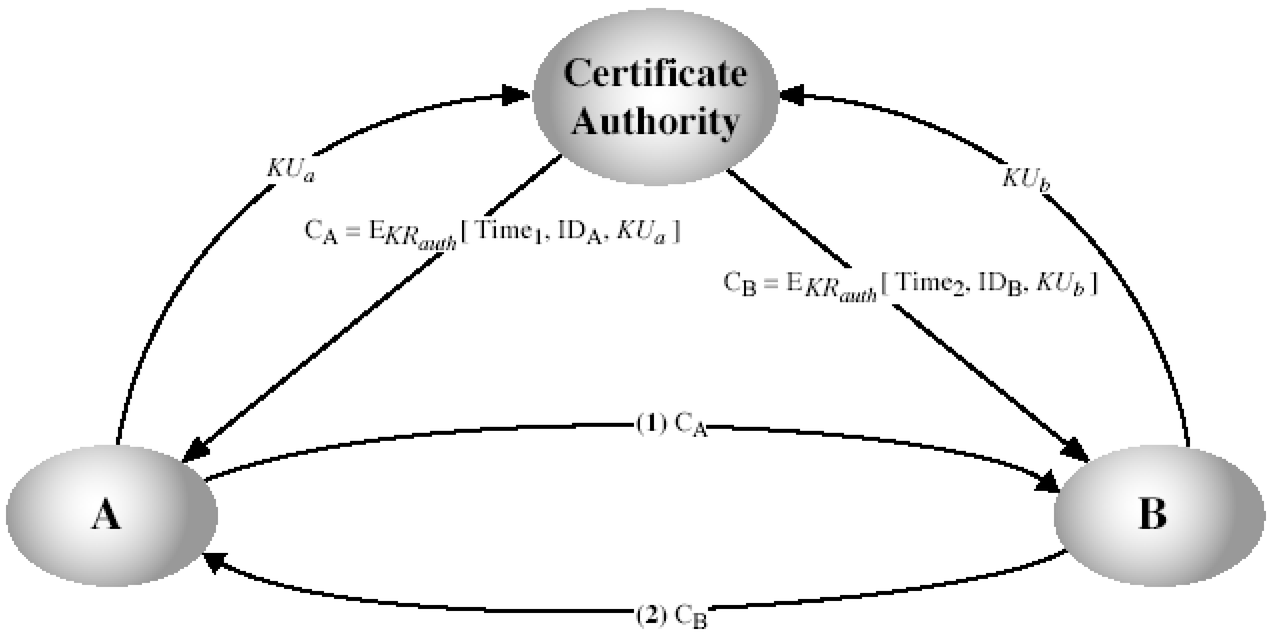
Public-Key Authority



Public-Key Certificates

- certificates allow key exchange without real-time access to public-key authority
- a certificate binds **identity** to **public key**
  - usually with other info such as period of validity, rights of use etc
- with all contents **signed** by a trusted Public-Key or Certificate Authority (CA)
- can be verified by anyone who knows the public-key authority's public-key

Public-Key Certificates



Public-Key Distribution of Secret Keys

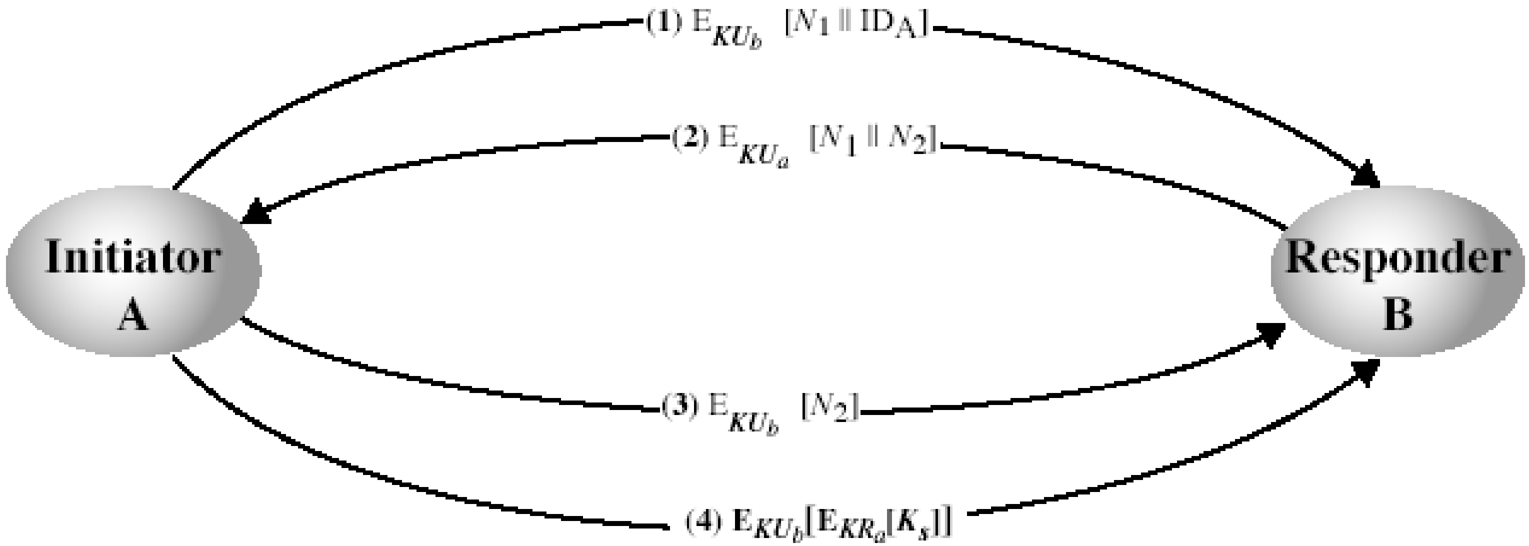
- use previous methods to obtain public-key
- can use for secrecy or authentication
- but public-key algorithms are slow
- so usually want to use private-key encryption to protect message contents
- hence need a session key
- have several alternatives for negotiating a suitable session

Simple Secret Key Distribution

- proposed by Merkle in 1979
  - A generates a new temporary public key pair
  - A sends B the public key and their identity
  - B generates a session key K 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

Public-Key Distribution of Secret Keys

- if have securely exchanged public-keys:



Diffie-Hellman Key Exchange

- agreement more than exchange
- first public-key type scheme proposed
- by Diffie & 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
- is a practical method for public agreement of a secret key
- used in a number of commercial products

Diffie-Hellman Key Exchange

- a public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key known only to the two participants
- value of key depends on the participants (and their private and public key information)
- 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 Setup

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

Diffie-Hellman Key Exchange

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

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

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

  $= y_B^{x_A} \bmod q$ (which **A** can compute)
- K_{AB} is used as session key in private-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 an x, must solve discrete log
- note active attack possible

Diffie-Hellman Example

- users Alice & Bob who wish to swap keys:
- agree on prime $q=353$ and $\alpha=3$
- select random secret keys:
  - A chooses $x_A=97$, B chooses $x_B=233$
- compute public keys:
  - $y_A=\mathbf{3}^{97} \bmod 353 = 40$ (Alice)
  - $y_B=\mathbf{3}^{233} \bmod 353 = 248$ (Bob)
- compute shared session key as:

$K_{AB}= y_B^{x_A} \bmod 353 = \mathbf{248}^{97} = 160$ (Alice)

$K_{AB}= y_A^{x_B} \bmod 353 = \mathbf{40}^{233} = 160$ (Bob)

Thank You…