Public Key Cryptography

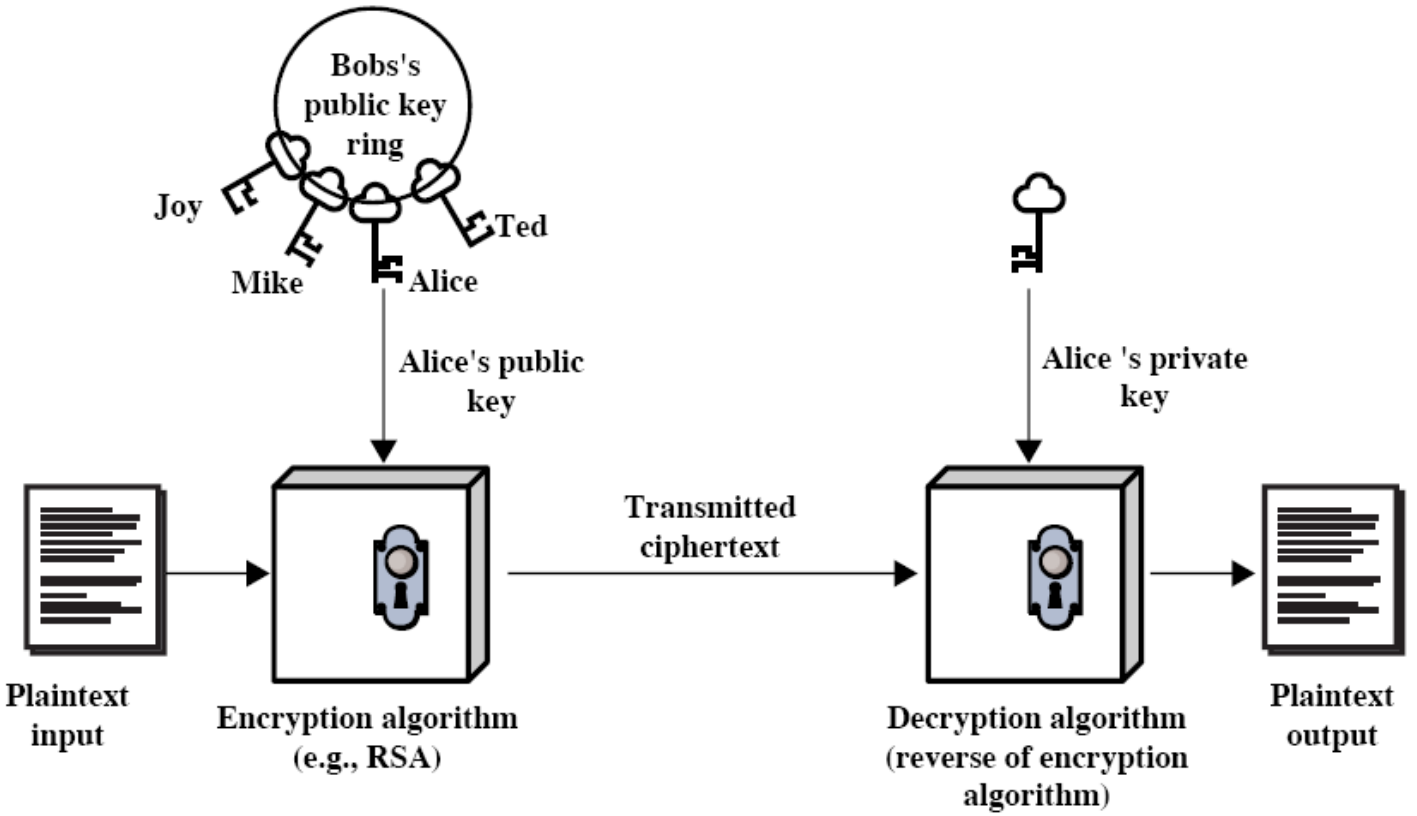
CSS 322 – Security and Cryptography

History of Public Key Systems

- Until public-key cryptosystems were discovered, cryptography relied on permutations and substitutions:
  - Caesar cipher, rotor machines, DES, …
- Diffie and Hellman published a public key system in 1976. Their motivation:
  - Symmetric key systems rely heavily on KDC being trustworthy and secure
  - Digital signatures are important
- Others (intelligence communities) claim to have discovered public key in 1960's and early 1970's

Public Key Encryption

- Public key uses two different keys
- Main concept:
  - Given the encryption key and algorithm, too hard to determine the decryption key



Public Key Encryption

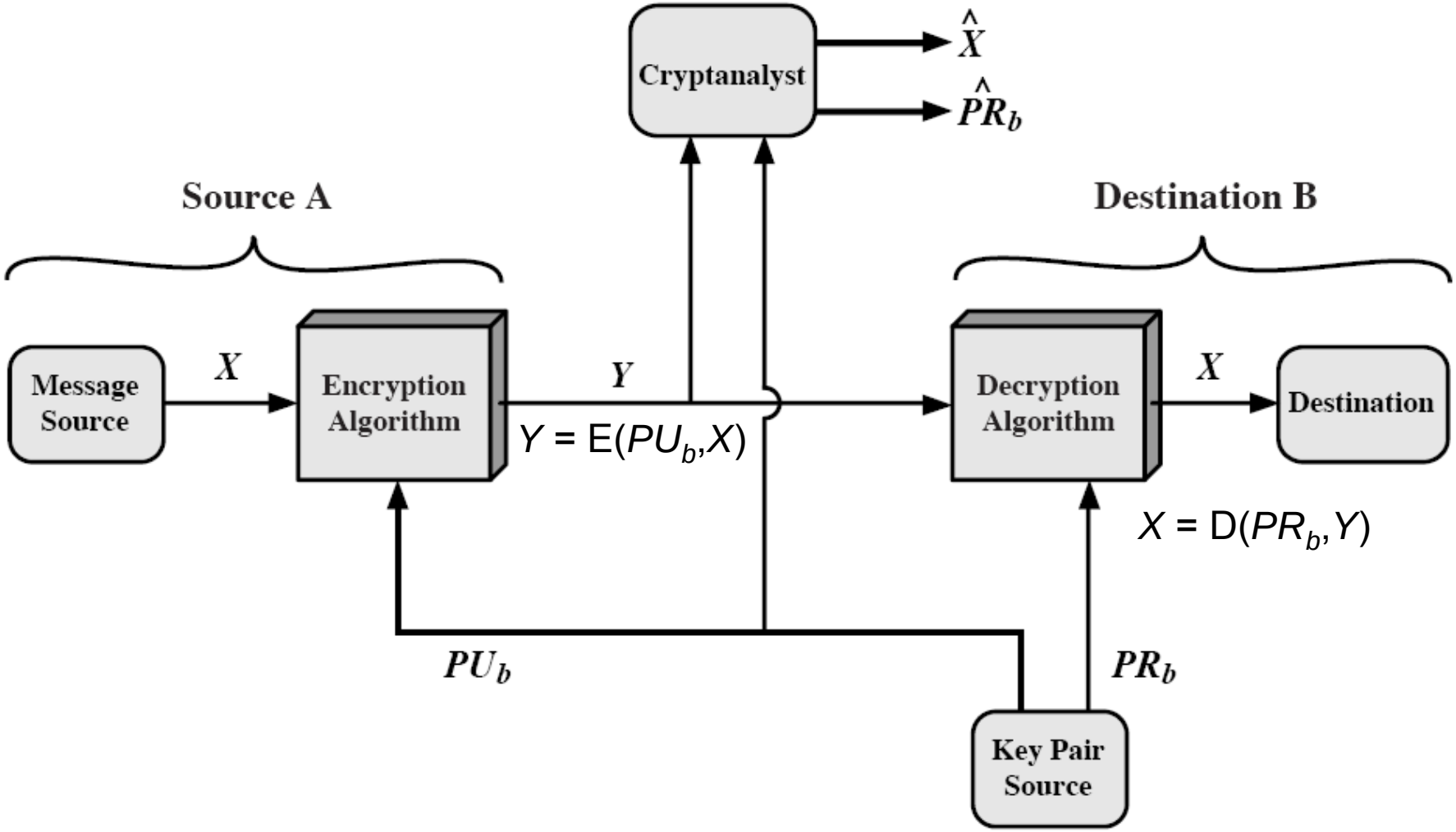
- Public key
  - Key used by sender to encrypt plaintext
  - Owned by the receiver
  - Anyone can know the public key
- Private (Secret) Key
  - Key used to decrypt ciphertext
  - Must be kept secret by the receiver
- The public key and private key are related
  - The pair belong to the receiver: (Public, Secret) or (P, S)

Symmetric vs Public Key Encryption

- Symmetric
- Same algorithm with same key used for encrypt and decrypt
- Sender and receiver must share algorithm and key
- Key must be kept secret

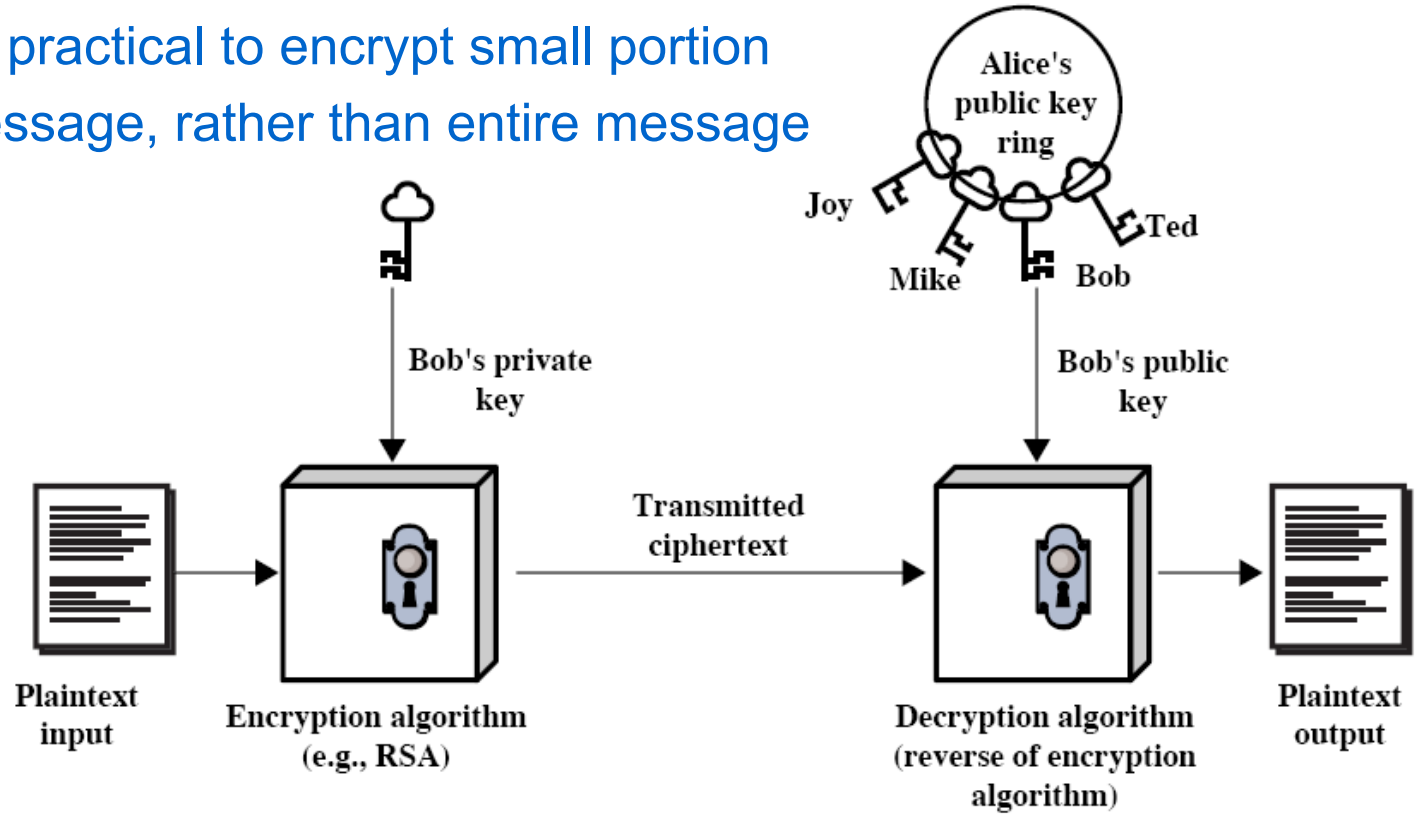
- Public
- One algorithm used for both encrypt and decrypt
- One key used for encrypt and another for decrypt
- Only one key must be secret

Privacy with Public Key Encryption

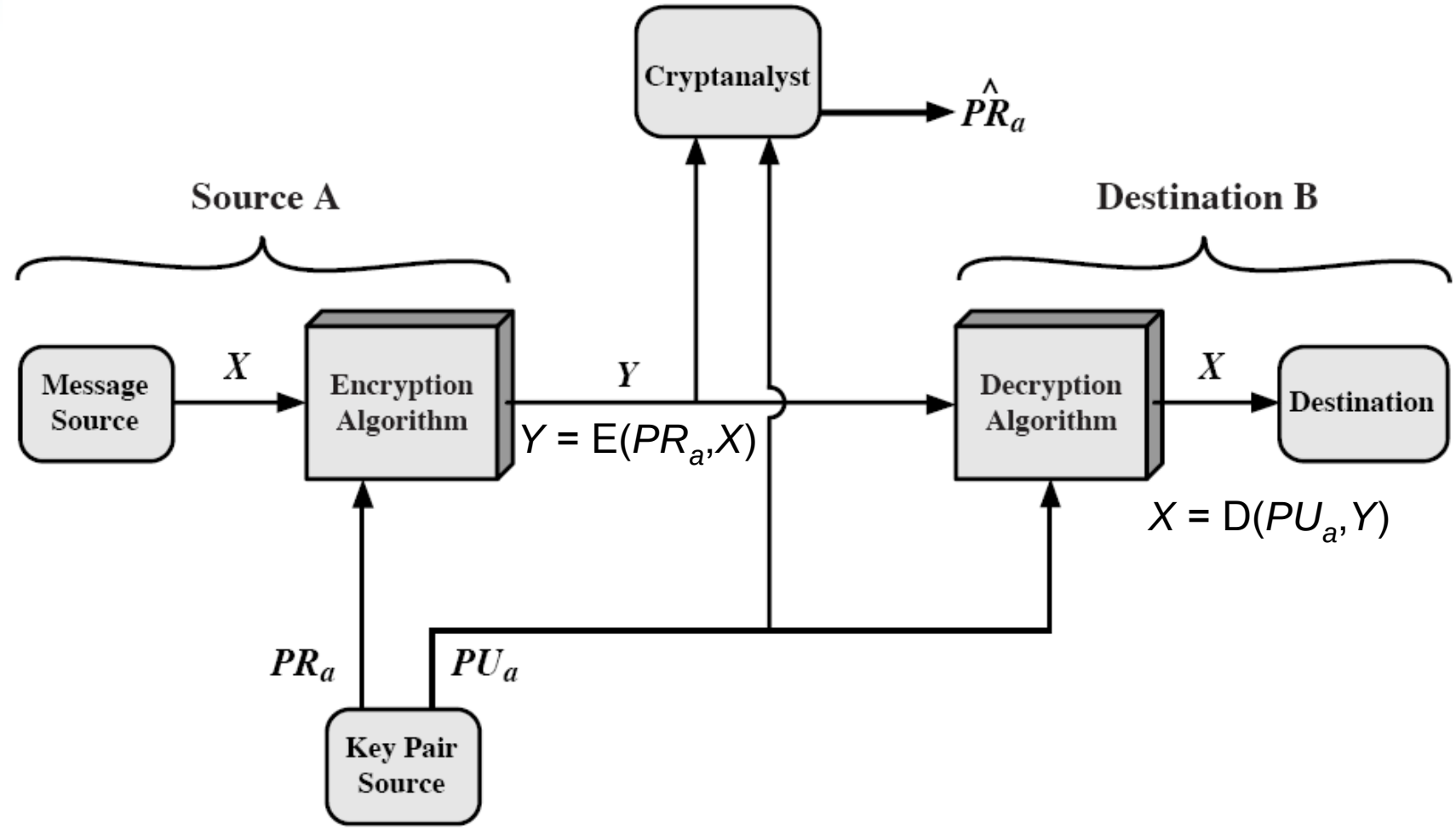


Public Key Authentication

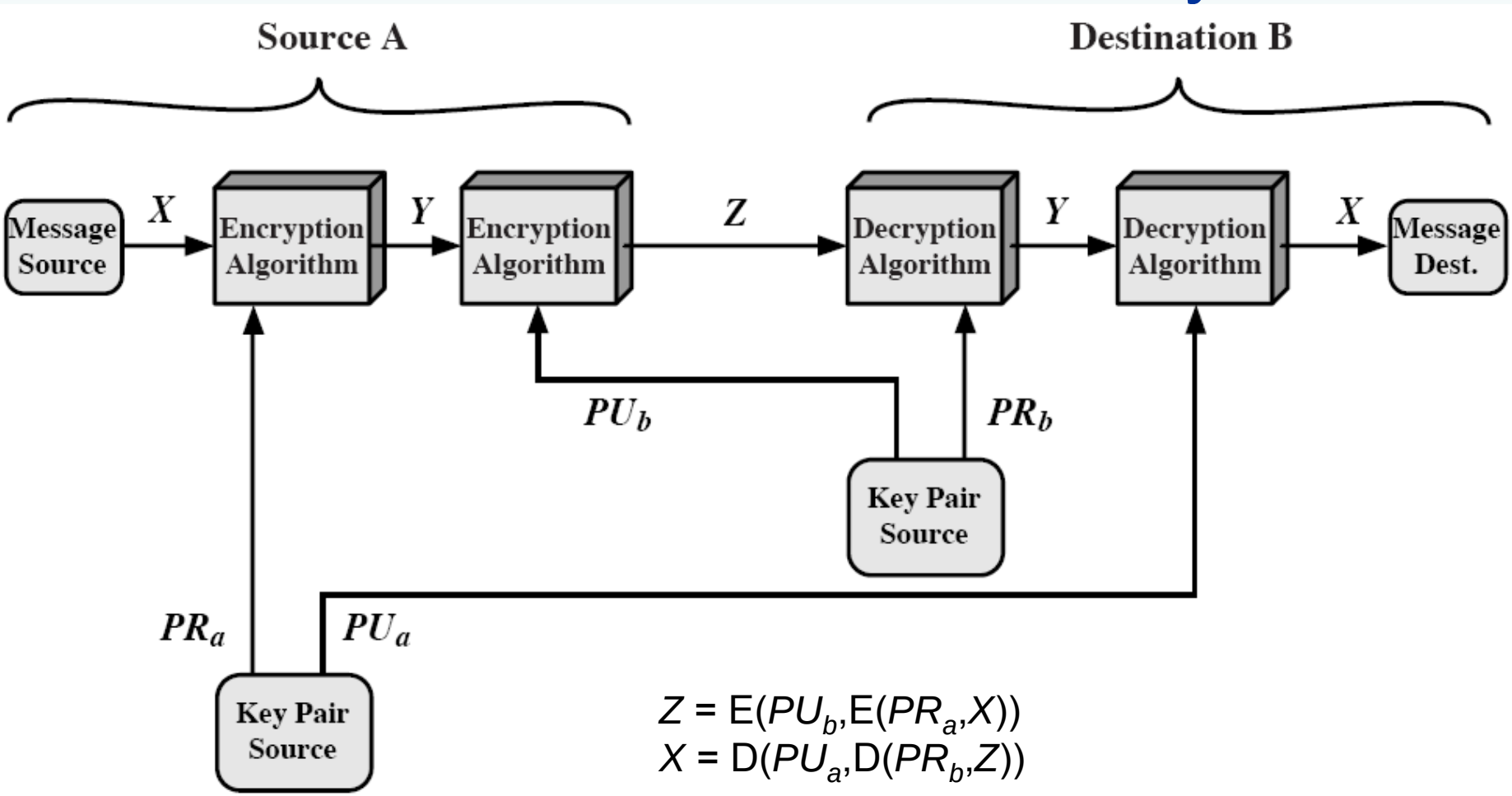
- Assuming if any key is used for encryption, the other key can be used for encryption
- Public key cryptography can be used for authentication
  - Encrypted message is *digital signature*
  - More practical to encrypt small portion of message, rather than entire message



Public Key Authentication



Authentication and Privacy



Applications of Public Key Crypto

- Encryption/Decryption (for privacy)
  - Sender encrypts message with recipients public key
- Digital Signature
  - Sender "signs" message with own private key
    - May sign entire message or a small part, for example, message hash
- Key Exchange
  - Sender and receiver cooperate to exchange session (often symmetric private) keys
    - Can use one or both private keys of sender and receiver

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 Public Key Crypto

- Most public key algorithms must meet these requirements:
  - Computationally easy for B to generate pair of keys: PU_b and PR_b
  - Computationally easy for A, knowing PU_b and message M, to generate ciphertext:

$$C = \mathrm{E}(PU_b, M)$$

  - Computationally easy for B to decrypt ciphertext using PR_b:

$$M = \mathrm{D}(PR_b, C) = \mathrm{D}[PR_b, \mathrm{E}(PU_b, M)]$$

  - Computationally hard (infeasible) for attacker knowing PU_b and C, to recover original message M
  - Computationally hard (infeasible) for attacker knowing PU_b to determine PR_b
  - Keys can be applied in either order:

$$M = \mathrm{D}[PU_b, \mathrm{E}(PR_b, M)] = \mathrm{D}[PR_b, \mathrm{E}(PU_b, M)]$$

Requirements of Public Key Crypto

- Need a one-way function
  - Easy to compute the function
  - Hard (infeasible) to compute the inverse of the function
- What is easy or hard?
  - Computational complexity of an algorithm gives an indication
    - Can it be solved in polynomial time as a function of input?
- Function should be:

$Y = f_k(X)$	easy, if k and X are known
$X = f_k^{-1}(Y)$	easy, if k and Y are known
$X = f_k^{-1}(Y)$	infeasible, if Y is known (but not k)

Attacking Public Key Systems

- Brute force attack on keys
  - Use large keys to avoid such attacks
  - But public key algorithms are less efficient with large keys
  - Mostly used for key management and signatures (rather than bulk data encryption)
- Compute private key, given the public key
  - Don’t know how to do it yet …

RSA

- Created by Ron Rivest, Adi Shamir and Len Adleman in 1978
  - Patented by MIT (patent has expired)
  - Developed by RSA Security, which sells many RSA products
- Most widely used public key system
- Block cipher
  - Plaintext and cipher text are integers

RSA Algorithm

- Plaintext is encrypted in blocks
  - Plaintext and ciphertext are integers with values less than *n*
- Assume a block size of *i* bits where: $2^i < n \leq 2^{i+1}$
- Encryption of message *M*:

$$C = M^e \bmod n$$

- Decryption of ciphertext *C*:

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

- Sender and receiver know *n*
- Sender knows *e*
- Receiver knows *d*
- *PU* = {*e*, *n*} and *PR* = {*d*, *n*}

RSA Algorithm

- For the RSA algorithm to work, there are several requirements
  1. Possible to find values for *e*, *d* and *n*, such that $M^{ed} \bmod n = M$
  2. Easy to calculate $M^e \bmod n$ and $C^d \bmod n$
  3. Infeasible to determine *d*, given *e* and *n*
- Requirement 1 is met if *e* and *d* are relatively prime
- To generate *e*, *d* and *n* to meet Req 1.:
  - *p*, *q* two prime numbers chosen and kept private

$$n = pq$$

$$1 < e < \phi(n)$$

$$ed \equiv 1 (\bmod \phi(n)) \text{ or } d \equiv e^{-1} (\bmod \phi(n))$$

  - *n* and *e* are public (*e* chosen); *d* is private

RSA Algorithm

Key Generation

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption

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

Decryption

Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

Computational Efficiency of RSA

- How complex is RSA to calculate ciphertext and plaintext?
  - Encryption and decryption require exponentiation ($M^e \bmod n$)
    - With M and e large integers (for example, M is 300 decimal digits), the result of M^e would be very large
    - But with modular arithmetic, RSA makes it simpler:

$$[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$$

    - There are also other ways to increase efficiency of operating on such large numbers
  - Size of d
    - Cannot be too small otherwise brute force and other attacks
    - Decryption can be made faster using Chinese Remainder Theorem and Fermat's Theorem
  - Choosing keys
    - Since n is public, and $n = pq$, p and q must be large prime numbers
    - No easy way to choose large prime numbers
      - Choose random number and apply test (e.g. Miller-Rabin) to see if prime

Security of RSA

- Brute Force Attacks
  - Same as all cryptosystems – choose a large key (but leads to slower performance)
- Factoring Attacks
  - *n* is known and *n* =*pq*; if we can factor a large number *n* into primes (*p* and *q*), then can break RSA
  - Factoring large numbers is hard, but improvements in factoring algorithms and computer speeds, makes it easier…

Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-years	Algorithm
100	332	April 1991	7	quadratic sieve
110	365	April 1992	75	quadratic sieve
120	398	June 1993	830	quadratic sieve
129	428	April 1994	5000	quadratic sieve
130	431	April 1996	1000	generalized number field sieve
140	465	February 1999	2000	generalized number field sieve
155	512	August 1999	8000	generalized number field sieve
160	530	April 2003	—	Lattice sieve
174	576	December 2003	—	Lattice sieve
200	663	May 2005	—	Lattice sieve

RSA Challenge

- RSA Security (the company) provide rewards for people who can factor large numbers

Challenge Number	Prize ($US)	Status	Submission Date	Submitter(s)
RSA-576	$10,000	Factored	December 3, 2003	J. Franke et al.
RSA-640	$20,000	Factored	November 2, 2005	F. Bahr et al.
RSA-704	$30,000	Not Factored		
RSA-768	$50,000	Not Factored		
RSA-896	$75,000	Not Factored		
RSA-1024	$100,000	Not Factored		
RSA-1536	$150,000	Not Factored		
RSA-2048	$200,000	Not Factored		

Security of RSA

- Timing Attacks
  - If can observe how long an implementation (software or hardware) takes to perform individual calculations of decryption, then can possibly obtain private key
    - Ciphertext only attack
  - Countermeasures include:
    - Make sure all calculations take the same time
    - Give all calculations a random delay
    - Use random number to modify ciphertext
  - All countermeasures lead to performance loss
    - Example: RSA Security products which include countermeasures lead to 2 to 10% performance loss
- Chosen Ciphertext Attack
  - If attacker can choose ciphertexts, and obtain decrypted plaintexts then can take advantage of RSA algorithm to derive key
  - Implementations have ways of padding plaintext to avoid this