# Arithmetic for Cryptography

# Principles of Pseudorandom Number Generation

- **The Use of Random Numbers**
- Many uses of **random numbers** in cryptography
  1. nonces in authentication protocols to prevent replay
  2. session keys
  3. public key generation
  4. keystream for a one-time pad
- These applications give rise to two distinct requirements for a sequence of random numbers:
  1. Randomness – the generated sequence of numbers be random in some well defined statistical sense.
  2. Unpredictability - the requirement is not randomness but that the successive members of the sequence are unpredictable.

# Principles of Pseudorandom Number Generation

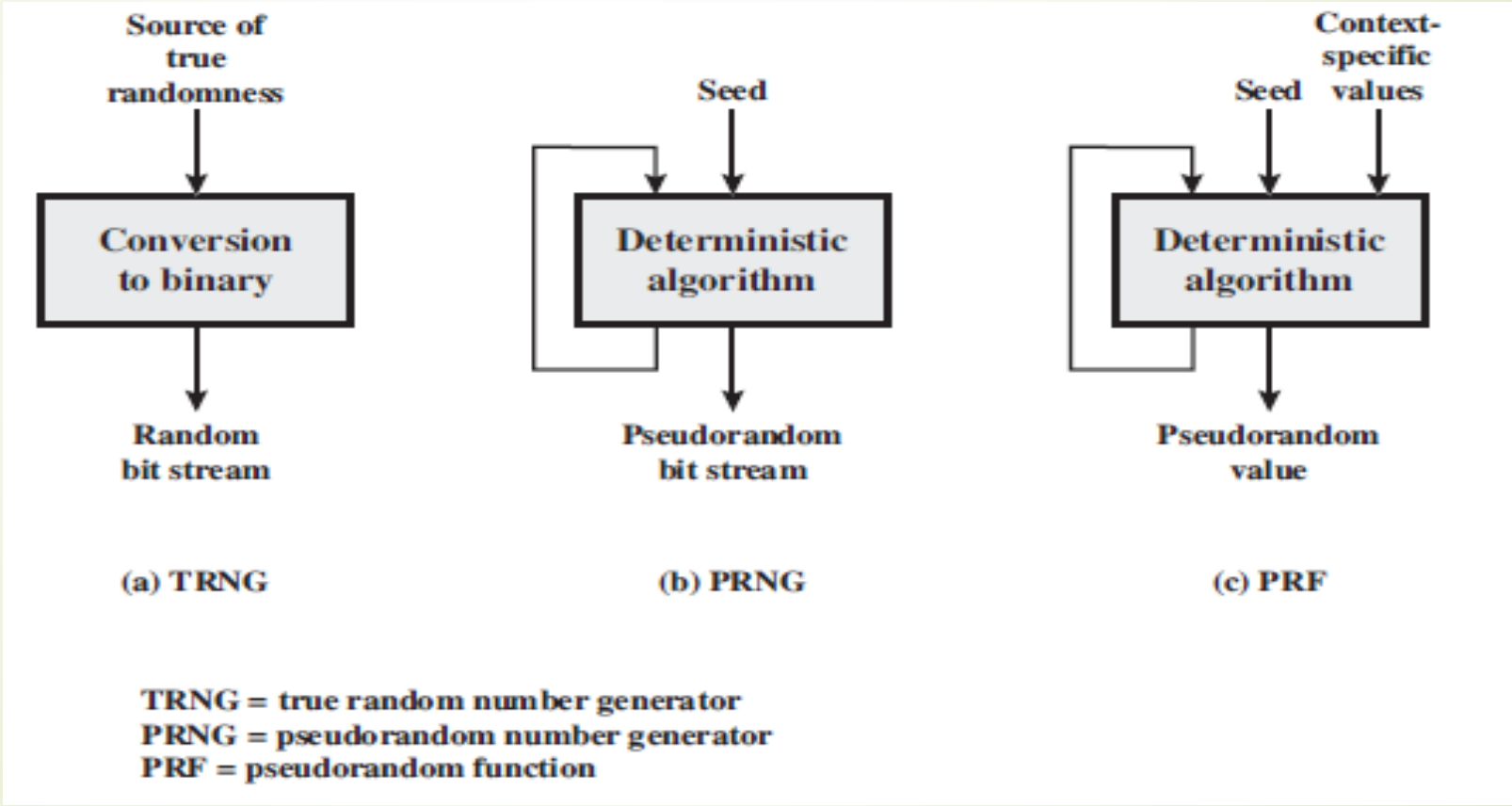
- **TRNGs, PRNGs, and PRFs**
- Cryptographic applications make use of many techniques for random number generation.
- These algorithms are deterministic and produce sequences of numbers that are not statistically random.
- If the algorithm is good, the resulting sequences will pass many reasonable tests of randomness. Such numbers are referred to as **pseudorandom numbers.**

- **TRNG**
  - true random number generator
  - takes as input a source that is effectively random, the source is often referred to as an **entropy source**.
  - TRNG may simply involve conversion of an analog source to a binary output.

# Principles of Pseudorandom Number Generation

- **PRNG**
  - PRNG takes as input a fixed value, called the **seed**, and produces a sequence of output bits using a deterministic algorithm.
  - output bit stream is determined solely by the input value or values.
- **Pseudorandom number generator:** An algorithm that is used to produce an open-ended sequence of bits is referred to as a PRNG.
- **Pseudorandom function (PRF)**
  - A PRF is used to produce a pseudorandom string of bits of some fixed length. Ex- symmetric encryption keys, nonces.
  - the PRF takes as input a seed plus some context specific values, such as a user ID or an application ID.
- Other than the number of bits produced, there is **no difference** between a PRNG and a PRF.

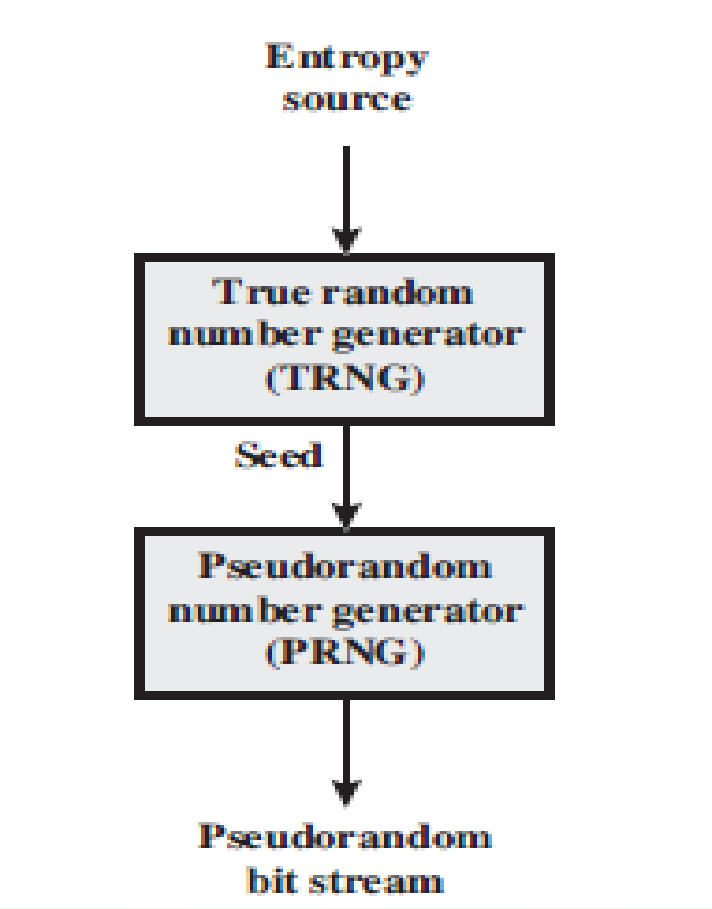
# Principles of Pseudorandom Number Generation



# Principles of Pseudorandom Number Generation

- **PRNG Requirements**
- **RANDOMNESS**
  1. Uniformity
  2. Scalability
  3. Consistency
- **UNPREDICTABILITY**
  1. forward & backward unpredictability
  2. use same tests to check
- **CHARACTERISTICS OF THE SEED**
  1. secure
  2. if known adversary can determine output
  3. must be random or pseudorandom number

# Principles of Pseudorandom Number Generation



# Principles of Pseudorandom Number Generation

- **Algorithm Design**
- Two categories:
  1. **Purpose-built algorithms:** These are algorithms designed specifically and solely for the purpose of generating pseudorandom bit streams.
  2. **Algorithms based on existing cryptographic algorithms:** Cryptographic algorithms have the effect of randomizing input.
     - Three categories of cryptographic algorithms are used to create PRNGs:
       1. **Symmetric block ciphers**
       2. Asymmetric ciphers
       3. Hash functions and message authentication codes

# Pseudorandom Number Generators

- **Linear Congruential Generators**
- Proposed by Lehmer.
- The algorithm is parameterized with four numbers:

| | | |
|---|---|---|
| m | the modulus | m>0 |
| a | the multiplier | 0<a<m |
| c | the increment | 0<=c<m |
| $X_0$ | the starting value, or seed | $0<=X_0<m$ |

- The sequence of random numbers $\{X_n\}$ is obtained via iterative equation:

$$X_{n+1} = (aX_n + c) \bmod m$$

- If m, a, c and $X_0$ are integers, then this technique will produce a sequence of integers with each integer in the range $0<=X_n<m$.

# Pseudorandom Number Generators

- **Blum Blum Shub Generator**
- Based on public key algorithms
- procedure is:
  - choose two large prime numbers, p and q, that both have a remainder of 3 when divided by 4.
  - That is, $p \equiv q \equiv 3 (\mathrm{mod}\ 4)$
- the BBS generator produces a sequence of bits according to the following algorithm:

$$
\begin{aligned}
&X_0 = s^2 \bmod n \\
&\textbf{for } i = 1 \textbf{ to } \infty \\
&\quad X_i = (X_{i-1})^2 \bmod n \\
&\quad B_i = X_i \bmod 2
\end{aligned}
$$

# Pseudorandom Number Generators

- The BBS is referred to as a **cryptographically secure pseudorandom bit generator** (CSPRBG).
- Example of BBS operation
  - Here, n = 192649 = 383 x 503, and the seed s = 101355.

| $i$ | $X_i$ | $B_i$ |
|---|---|---|
| 0 | 20749 | |
| 1 | 143135 | 1 |
| 2 | 177671 | 1 |
| 3 | 97048 | 0 |
| 4 | 89992 | 0 |
| 5 | 174051 | 1 |
| 6 | 80649 | 1 |
| 7 | 45663 | 1 |
| 8 | 69442 | 0 |
| 9 | 186894 | 0 |
| 10 | 177046 | 0 |

| $i$ | $X_i$ | $B_i$ |
|---|---|---|
| 11 | 137922 | 0 |
| 12 | 123175 | 1 |
| 13 | 8630 | 0 |
| 14 | 114386 | 0 |
| 15 | 14863 | 1 |
| 16 | 133015 | 1 |
| 17 | 106065 | 1 |
| 18 | 45870 | 0 |
| 19 | 137171 | 1 |
| 20 | 48060 | 0 |

# Prime Numbers

- Central concern of number theory is the study of prime numbers.
- An integer p > 1 is a **prime number** if and only if its only divisors are 1 and p.
- Any integer a > 1 can be factored in a unique way as:

$$a = p_1^{a1} \times p_2^{a2} \times \ldots \times p_t^{at}$$

- Examples

$$91 = 7 \times 13$$

$$3600 = 2^4 \times 3^2 \times 5^2$$

$$11011 = 7 \times 11^2 \times 13$$

- The value of any given positive integer can be specified by simply listing all the **nonzero exponents** in the foregoing formulation.

The integer 12 is represented by $\{a2 = 2, a3 = 1\}$.

The integer 18 is represented by $\{a2 = 1, a3 = 2\}$.

The integer 91 is represented by $\{a7 = 1, a13 = 1\}$.

# Prime numbers under 2000

| 2 | 101 | 211 | 307 | 401 | 503 | 601 | 701 | 809 | 907 | 1009 | 1103 | 1201 | 1301 | 1409 | 1511 | 1601 | 1709 | 1801 | 1901 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 103 | 223 | 311 | 409 | 509 | 607 | 709 | 811 | 911 | 1013 | 1109 | 1213 | 1303 | 1423 | 1523 | 1607 | 1721 | 1811 | 1907 |
| 5 | 107 | 227 | 313 | 419 | 521 | 613 | 719 | 821 | 919 | 1019 | 1117 | 1217 | 1307 | 1427 | 1531 | 1609 | 1723 | 1823 | 1913 |
| 7 | 109 | 229 | 317 | 421 | 523 | 617 | 727 | 823 | 929 | 1021 | 1123 | 1223 | 1319 | 1429 | 1543 | 1613 | 1733 | 1831 | 1931 |
| 11 | 113 | 233 | 331 | 431 | 541 | 619 | 733 | 827 | 937 | 1031 | 1129 | 1229 | 1321 | 1433 | 1549 | 1619 | 1741 | 1847 | 1933 |
| 13 | 127 | 239 | 337 | 433 | 547 | 631 | 739 | 829 | 941 | 1033 | 1151 | 1231 | 1327 | 1439 | 1553 | 1621 | 1747 | 1861 | 1949 |
| 17 | 131 | 241 | 347 | 439 | 557 | 641 | 743 | 839 | 947 | 1039 | 1153 | 1237 | 1361 | 1447 | 1559 | 1627 | 1753 | 1867 | 1951 |
| 19 | 137 | 251 | 349 | 443 | 563 | 643 | 751 | 853 | 953 | 1049 | 1163 | 1249 | 1367 | 1451 | 1567 | 1637 | 1759 | 1871 | 1973 |
| 23 | 139 | 257 | 353 | 449 | 569 | 647 | 757 | 857 | 967 | 1051 | 1171 | 1259 | 1373 | 1453 | 1571 | 1657 | 1777 | 1873 | 1979 |
| 29 | 149 | 263 | 359 | 457 | 571 | 653 | 761 | 859 | 971 | 1061 | 1181 | 1277 | 1381 | 1459 | 1579 | 1663 | 1783 | 1877 | 1987 |
| 31 | 151 | 269 | 367 | 461 | 577 | 659 | 769 | 863 | 977 | 1063 | 1187 | 1279 | 1399 | 1471 | 1583 | 1667 | 1787 | 1879 | 1993 |
| 37 | 157 | 271 | 373 | 463 | 587 | 661 | 773 | 877 | 983 | 1069 | 1193 | 1283 |  | 1481 | 1597 | 1669 | 1789 | 1889 | 1997 |
| 41 | 163 | 277 | 379 | 467 | 593 | 673 | 787 | 881 | 991 | 1087 |  | 1289 |  | 1483 |  | 1693 |  |  | 1999 |
| 43 | 167 | 281 | 383 | 479 | 599 | 677 | 797 | 883 | 997 | 1091 |  | 1291 |  | 1487 |  | 1697 |  |  |  |
| 47 | 173 | 283 | 389 | 487 |  | 683 |  | 887 |  | 1093 |  | 1297 |  | 1489 |  | 1699 |  |  |  |
| 53 | 179 | 293 | 397 | 491 |  | 691 |  |  |  | 1097 |  |  |  | 1493 |  |  |  |  |  |
| 59 | 181 |  |  | 499 |  |  |  |  |  |  |  |  |  | 1499 |  |  |  |  |  |
| 61 | 191 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 67 | 193 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 71 | 197 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 73 | 199 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 79 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 83 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 89 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 97 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

# Relatively Prime Numbers & GCD

- Two numbers a, b are relatively prime if have no common divisors apart from 1.
  - eg. 8 & 15 are relatively prime since factors of 8 are 1,2,4,8 and of 15 are 1,3,5,15 and 1 is the only common factor.
- Can determine the greatest common divisor (GCD) by comparing their prime factorizations and using least powers.
  - eg. $300 = 2^1 \times 3^1 \times 5^2$ $18 = 2^1 \times 3^2$ hence

    $GCD(18,300) = 2^1 \times 3^1 \times 5^0 = 6$

# Fermat's Theorem

- Two theorems that play important role in public-key cryptography:
  1. Fermat’s theorem and
  2. Euler’s theorem.
- **Fermat's Theorem**
- $a^{p-1} = 1 \pmod{p}$

  where p is prime and gcd(a,p)=1

$$
\begin{aligned}
&a = 7, p = 19 \\
&7^2 = 49 \equiv 11 \pmod{19} \\
&7^4 \equiv 121 \equiv 7 \pmod{19} \\
&7^8 \equiv 49 \equiv 11 \pmod{19} \\
&7^{16} \equiv 121 \equiv 7 \pmod{19} \\
&a^{p-1} = 7^{18} = 7^{16} \times 7^2 \equiv 7 \times 11 \equiv 1 \pmod{19}
\end{aligned}
$$

# Fermat's Theorem

- Alternative form of Fermat's theorem is also useful:
  - If is prime and is a positive integer, then
  - $a^p = a \pmod p$

- The first form of the theorem requires that **a** be relatively prime to **p**, but this form does not.

$$p = 5, a = 3 \quad a^p = 3^5 = 243 \equiv 3 \pmod 5 = a \pmod p$$
$$p = 5, a = 10 \quad a^p = 10^5 = 100000 \equiv 10 \pmod 5 \equiv 0 \pmod 5 = a \pmod p$$

# Euler's Totient Function (ϕ(n))

- The number of positive integers less than n and relatively prime to n.
- ϕ(1) = 1.

DETERMINE $\phi(37)$ AND $\phi(35)$.

Because 37 is prime, all of the positive integers from 1 through 36 are relatively prime to 37. Thus $\phi(37) = 36$.

To determine $\phi(35)$, we list all of the positive integers less than 35 that are relatively prime to it:

1, 2, 3, 4, 6, 8, 9, 11, 12, 13, 16, 17, 18
19, 22, 23, 24, 26, 27, 29, 31, 32, 33, 34

There are 24 numbers on the list, so $\phi(35) = 24$.

# Euler's Totient Function (ϕ(n))

- for a prime number p,

$$\Phi(p) = p - 1$$

- Suppose that we have two prime numbers **p** and **q** with **p ≠ q**.
  - for n = pq,
  - $\Phi(n) = \Phi(pq) = \Phi(p) \times \Phi(q) = (p - 1) \times (q - 1)$
- To see that, Φ(n) = Φ(p) x Φ(q)
- Φ(n) = (pq - 1) - [(q - 1) + (p - 1)

  = pq - (p + q) + 1

  = (p - 1) x (q - 1)

  = Φ(p) x Φ(q)

$\phi(21) = \phi(3) \times \phi(7) = (3 - 1) \times (7 - 1) = 2 \times 6 = 12$
where the 12 integers are {1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20}.

# Euler's Theorem

- A generalization of Fermat's Theorem
- $\mathbf{a^{\Phi(n)} = 1 \pmod{n}}$

  for any a, n where gcd(a, n)=1
- eg. **a = 3 ; n = 10 ; ϕ(10)=4;**

  hence $3^4 = 81 = 1 \bmod 10$

  **a=2 ; n=11 ; ϕ(11)=10;**

  hence $2^{10} = 1024 = 1 \bmod 11$
- also have: $\mathbf{a^{\Phi(n)+1} = a \pmod{n}}$

# Primality Testing

- often need to find large prime numbers
- traditionally sieve using trial division
  - ie. divide by all numbers (primes) in turn less than the square root of the number
  - only works for small numbers
- alternatively can use statistical primality tests based on properties of primes
  - for which all primes numbers satisfy property
  - but some composite numbers, called pseudo-primes, also satisfy the property
- can use a slower deterministic primality test

# The Chinese Remainder Theorem

- The Chinese remainder theorem (CRT) is used to solve a set of congruent equations with one variable but different moduli, which are relatively prime, as shown below:

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\cdots \\ x &\equiv a_k \pmod{m_k} \end{aligned}$$

- Suppose $m_1, \ldots, m_k$ are pairwise relatively prime positive integers, and suppose $a_1, \ldots, a_k$ are integers.
- Then the system of k congruences $x = a_i \pmod{m_i}$ $(1 \leq i \leq k)$ has a unique solution modulo $M = m_1 \times \ldots \times m_k$, which is given by

$$x = \sum_{i=1}^{k} a_i c_i \bmod M,$$

where $c_i = M_i (M_i^{-1} \bmod m_i)$ and $M_i = M / m_i$, for $1 \leq i \leq k$.

# The Chinese Remainder Theorem

- **Steps to follow:**
  1. Find M = m1 × m2 × … × mk. This is the common modulus.
  2. Find M1 = M/m1, M2 = M/m2, …, Mk = M/mk.
  3. Find the multiplicative inverse of M1, M2, …, Mk using the corresponding moduli (m1, m2, …, mk). Call the inverses M1−1, M2−1, …, Mk −1.
  4. The solution to the simultaneous equations is

$$x = (a_1 \times M_1 \times M_1^{-1} + a_2 \times M_2 \times M_2^{-1} + \cdots + a_k \times M_k \times M_k^{-1}) \bmod M$$

# The Chinese Remainder Theorem (Example)

- Find the solution to the simultaneous equations:

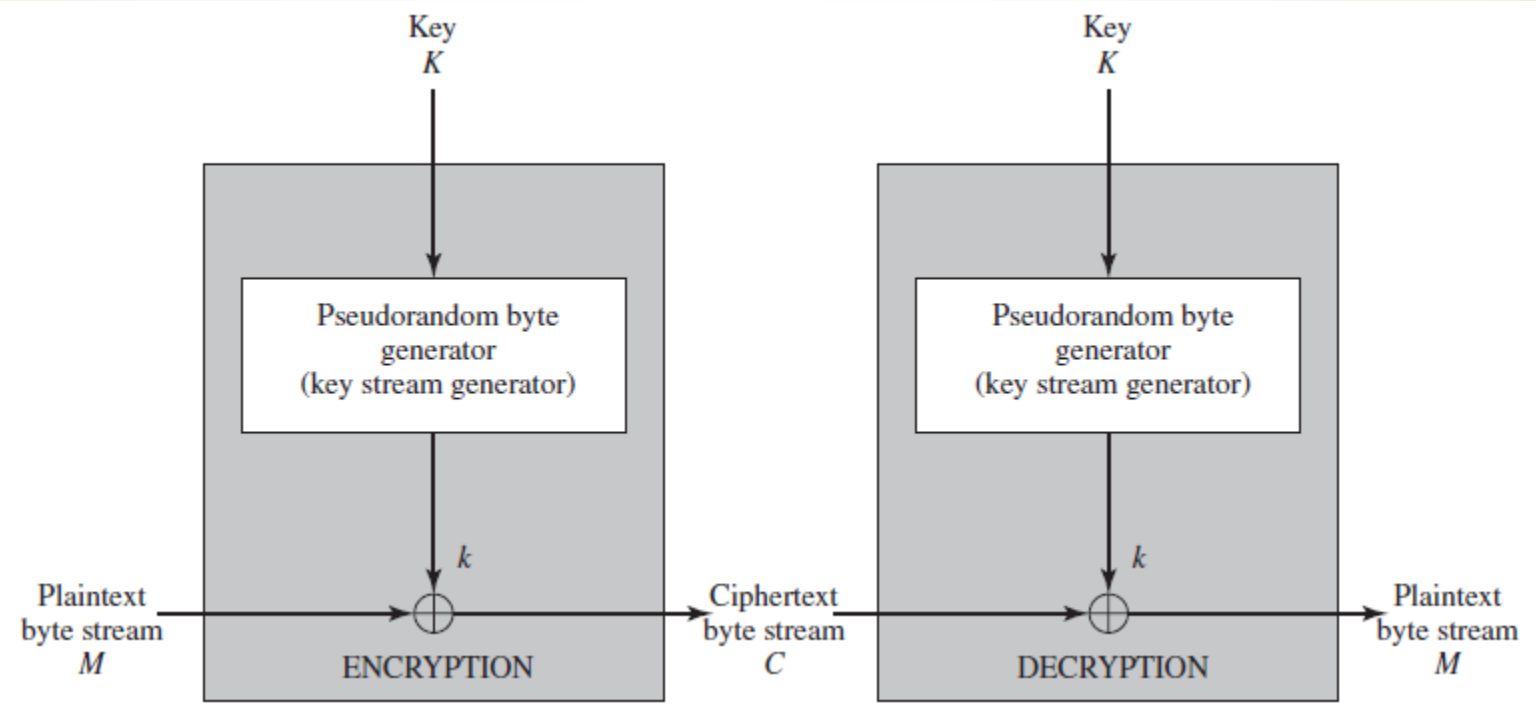
$$x \equiv 2 \pmod 3$$
$$x \equiv 3 \pmod 5$$
$$x \equiv 2 \pmod 7$$

- **Solution-**
- We follow the four steps.

1. $M = 3 \times 5 \times 7 = 105$
2. $M_1 = 105 / 3 = 35$, $M_2 = 105 / 5 = 21$, $M_3 = 105 / 7 = 15$
3. The inverses are $M_1^{-1} = 2$, $M_2^{-1} = 1$, $M_3^{-1} = 1$
4. $x = (2 \times 35 \times 2 + 3 \times 21 \times 1 + 2 \times 15 \times 1) \bmod 105 = 23 \bmod 105$

# Stream Ciphers

- Stream cipher encrypts plaintext one bit or one byte at a time.



# Stream Ciphers

- A key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers.
- The output of the generator, called a **keystream**, is combined one byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation.

```
  11001100  plaintext
⊕ 01101100  key stream
  10100000  ciphertext
```

```
  10100000  ciphertext
⊕ 01101100  key stream
  11001100  plaintext
```

- The stream cipher is similar to the one-time pad.
- The **difference** is that a **one-time pad** uses a genuine random number stream, whereas a **stream cipher** uses a pseudorandom number stream.

# RC4

- RC4 is a stream cipher designed in 1987 by Ron Rivest for RSA Security.
  - variable key size, byte-oriented stream cipher
  - widely used (web SSL/TLS, wireless WEP/WPA)
  - key forms random permutation of all 8-bit values
  - uses that permutation to scramble input info processed a byte at a time
- **Initialization of S**

  **for** i = 0 **to** 255 **do**

  S[i] = i;

  T[i] = K[i **mod** keylen];

# RC4

- **Use T to produce the initial permutation of S**

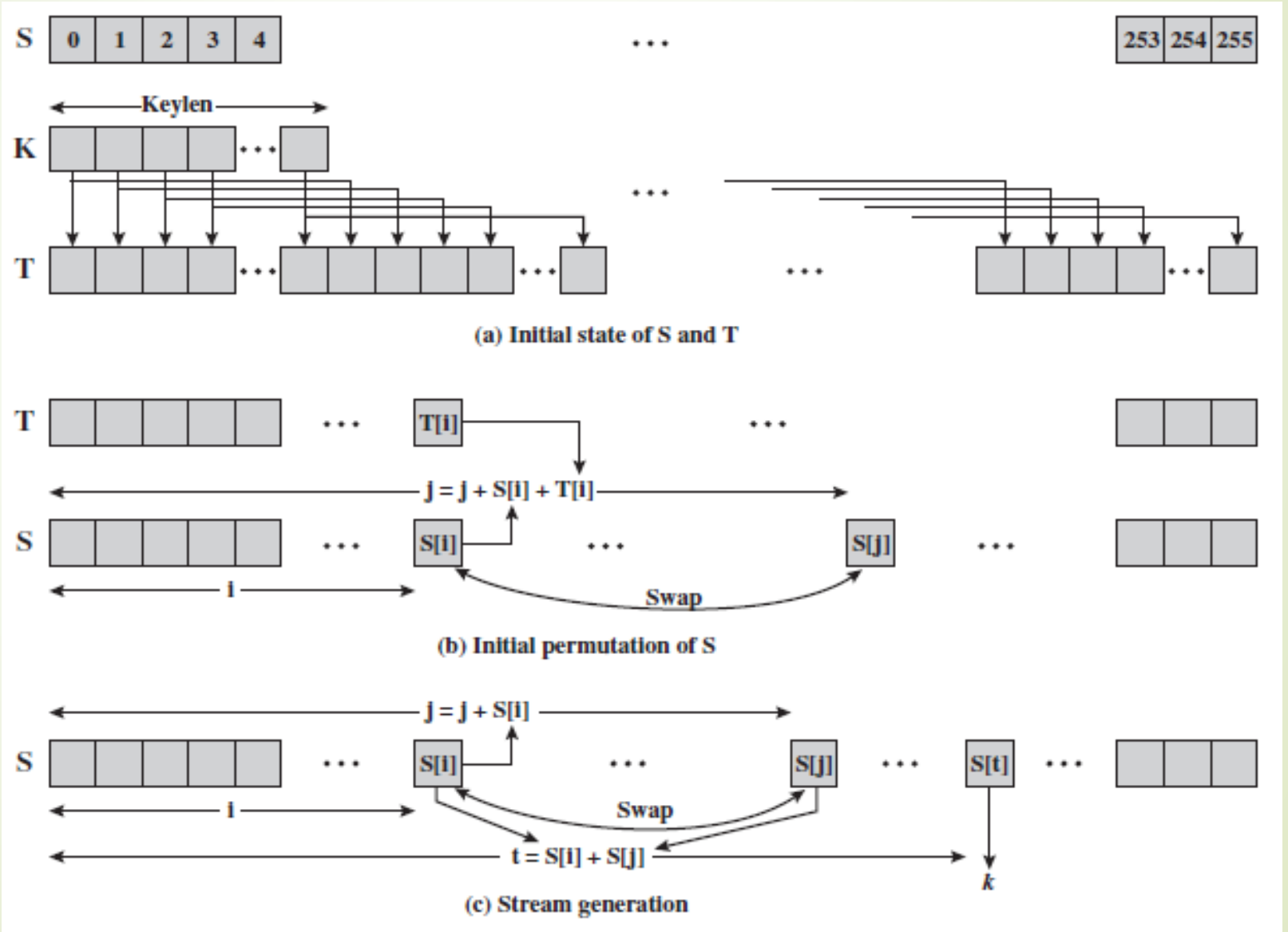
```
j = 0;
for i = 0 to 255 do
    j = (j + S[i] + T[i]) mod 256;
Swap (S[i], S[j]);
```

- **Stream Generation**

```
i, j = 0;
while (true)
    i = (i + 1) mod 256;
    j = (j + S[i]) mod 256;
Swap (S[i], S[j]);
t = (S[i] + S[j]) mod 256;
k = S[t];
```

- To encrypt, XOR the value with the next byte of plaintext. To decrypt, XOR the value with the next byte of ciphertext.

# RC4


S
0
1
2
3
4
253
254
255
Keylen
K
T
(a) Initial state of S and T
T
T[i]
j = j + S[i] + T[i]
S
S[i]
S[j]
i
Swap
(b) Initial permutation of S
j = j + S[i]
S
S[i]
S[j]
S[t]
i
Swap
t = S[i] + S[j]
k
(c) Stream generation

# Strength of RC4

- Claimed secure against known attacks
  - However, have some analyses, some on the verge of practical
  - first 256 bytes have bias
  - multiple keys/same plaintext attack
- result is very non-linear
- since RC4 is a stream cipher, must never reuse a key
- have a concern with WEP, but due to key handling rather than RC4 itself .

# PUBLIC KEY CRYPTOGRAPHY

# Principles of Public Key cryptosystems

- **Private-Key Cryptography**
  - traditional private/secret/single key cryptography uses one key
  - shared by both sender and receiver
  - if this key is disclosed communications are compromised
  - also is symmetric, parties are equal
  - hence does not protect sender from receiver forging a message & claiming is sent by sender.

# Principles of Public Key cryptosystems

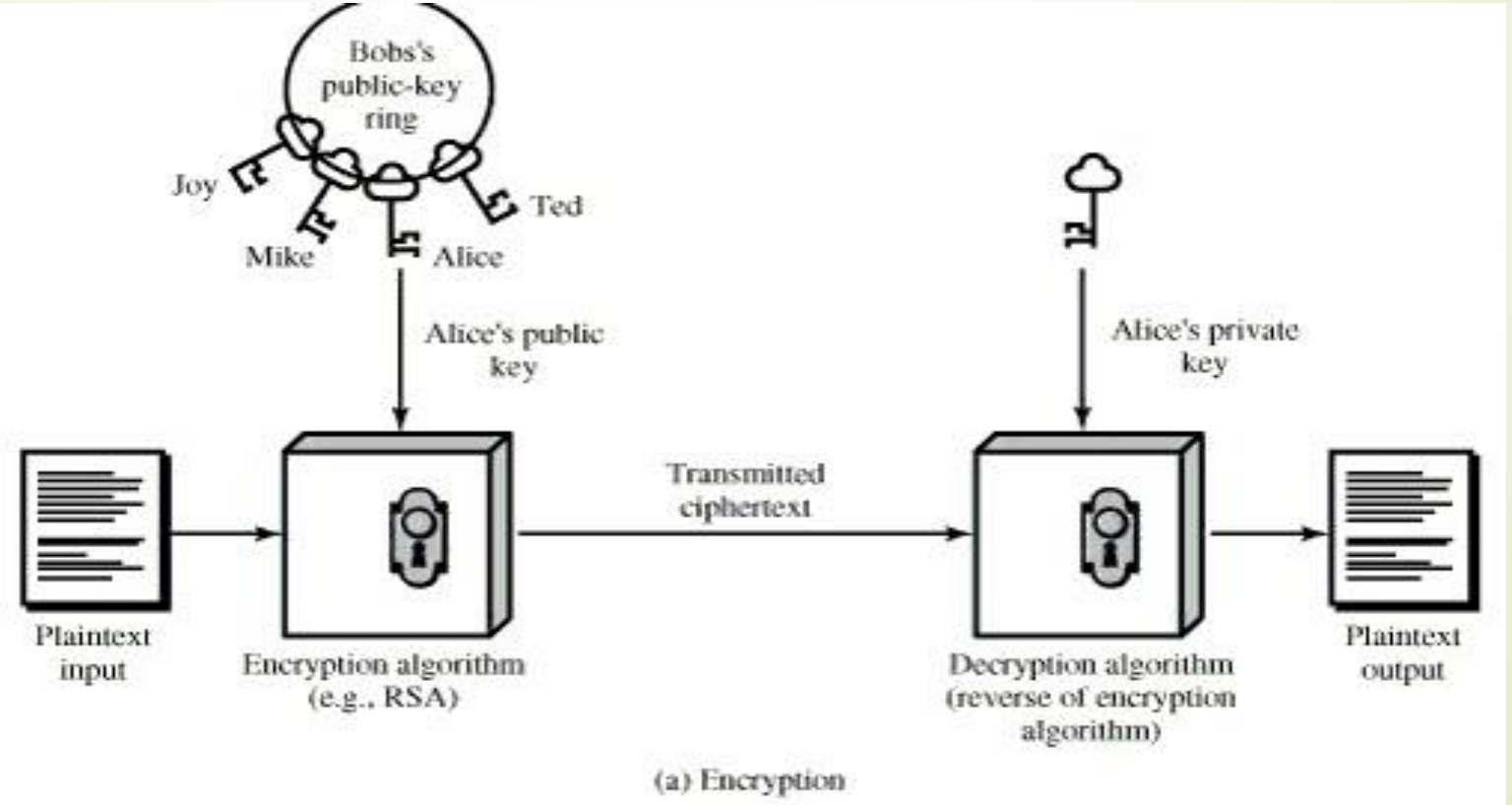
- **Public-Key Cryptography**
  - probably most significant advance in the 3000 year history of cryptography
  - uses two keys – a public & a private key
  - asymmetric since parties are not equal
  - uses clever application of number theoretic concepts to function
  - complements rather than replaces private key crypto.

# Principles of Public Key cryptosystems

- **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.

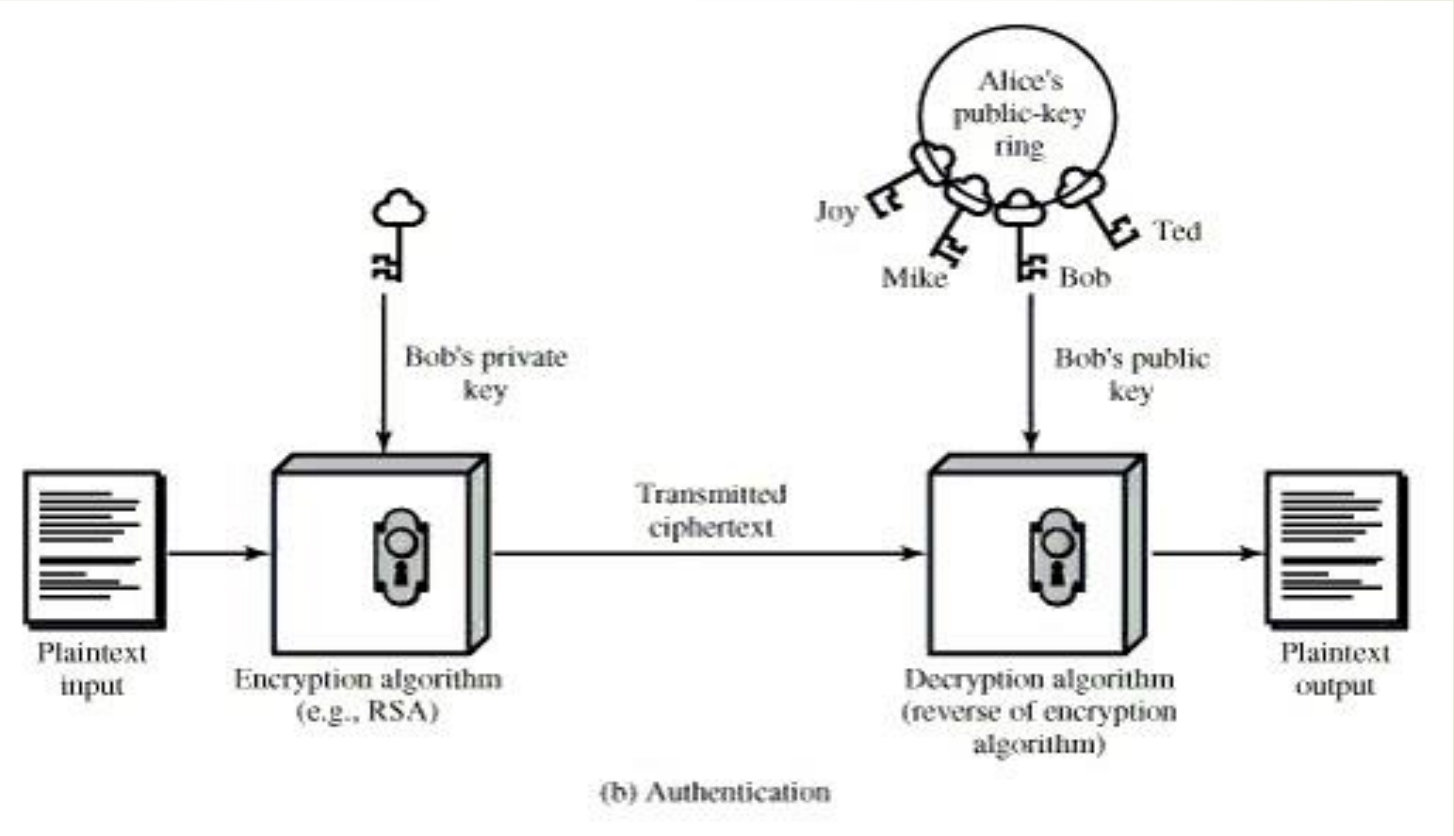
# Principles of Public Key cryptosystems

- **Public-Key Cryptography: Encryption**



# Principles of Public Key cryptosystems

- **Public-Key Cryptography: Authentication**



# Principles of Public Key cryptosystems

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

# Principles of Public Key cryptosystems

- **Conventional and Public – Key Encryption**

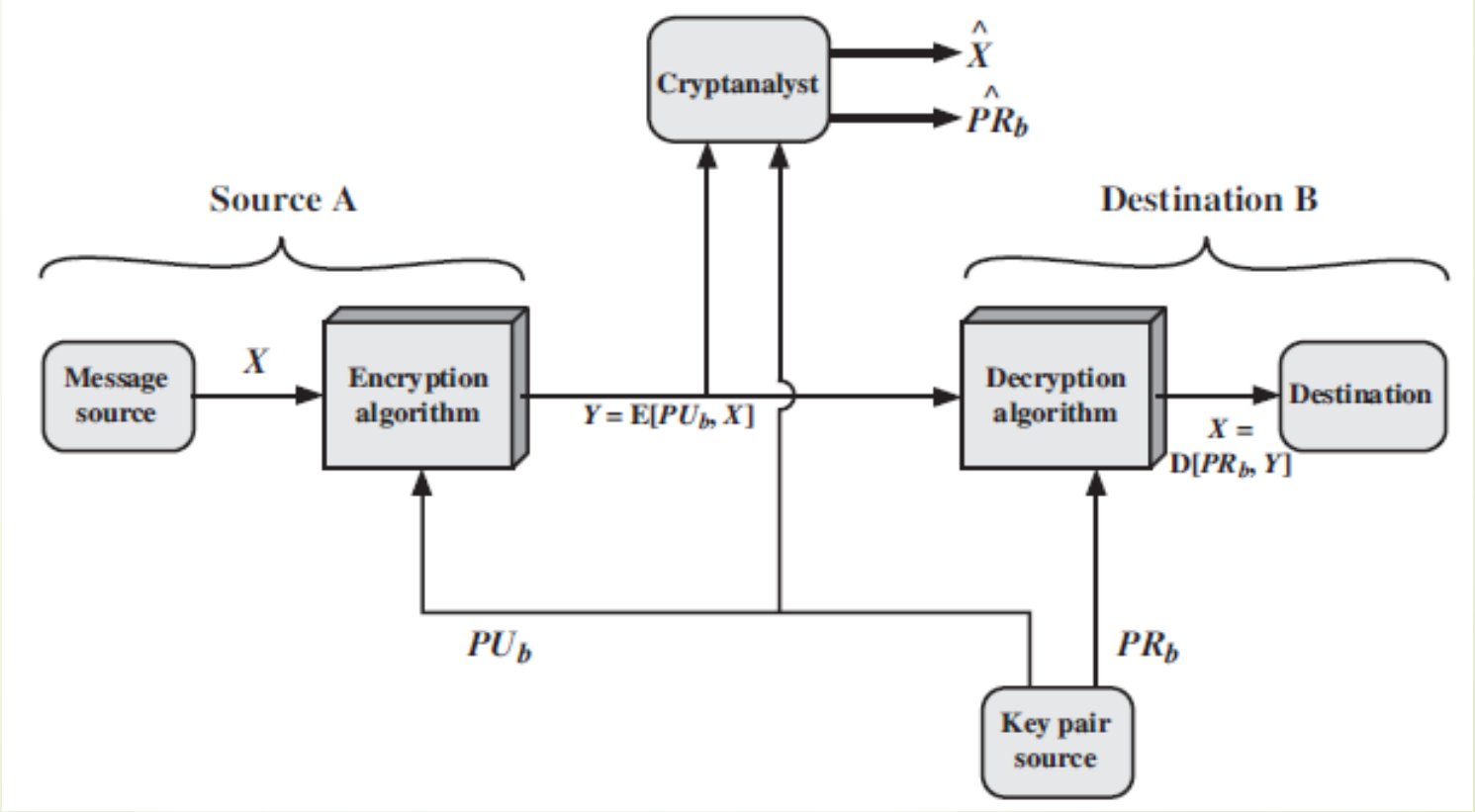
| Conventional Encryption | Public-Key Encryption |
|---|---|
| *Needed to Work:*<br>1. The same algorithm with the same key is used for encryption and decryption.<br>2. The sender and receiver must share the algorithm and the key.<br>*Needed for Security:*<br>1. The key must be kept secret.<br>2. It must be impossible or at least impractical to decipher a message if no other information is available.<br>3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key. | *Needed to Work:*<br>1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption.<br>2. The sender and receiver must each have one of the matched pair of keys (not the same one).<br>*Needed for Security:*<br>1. One of the two keys must be kept secret.<br>2. It must be impossible or at least impractical to decipher a message if no other information is available.<br>3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key. |

# Principles of Public Key cryptosystems

- **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)

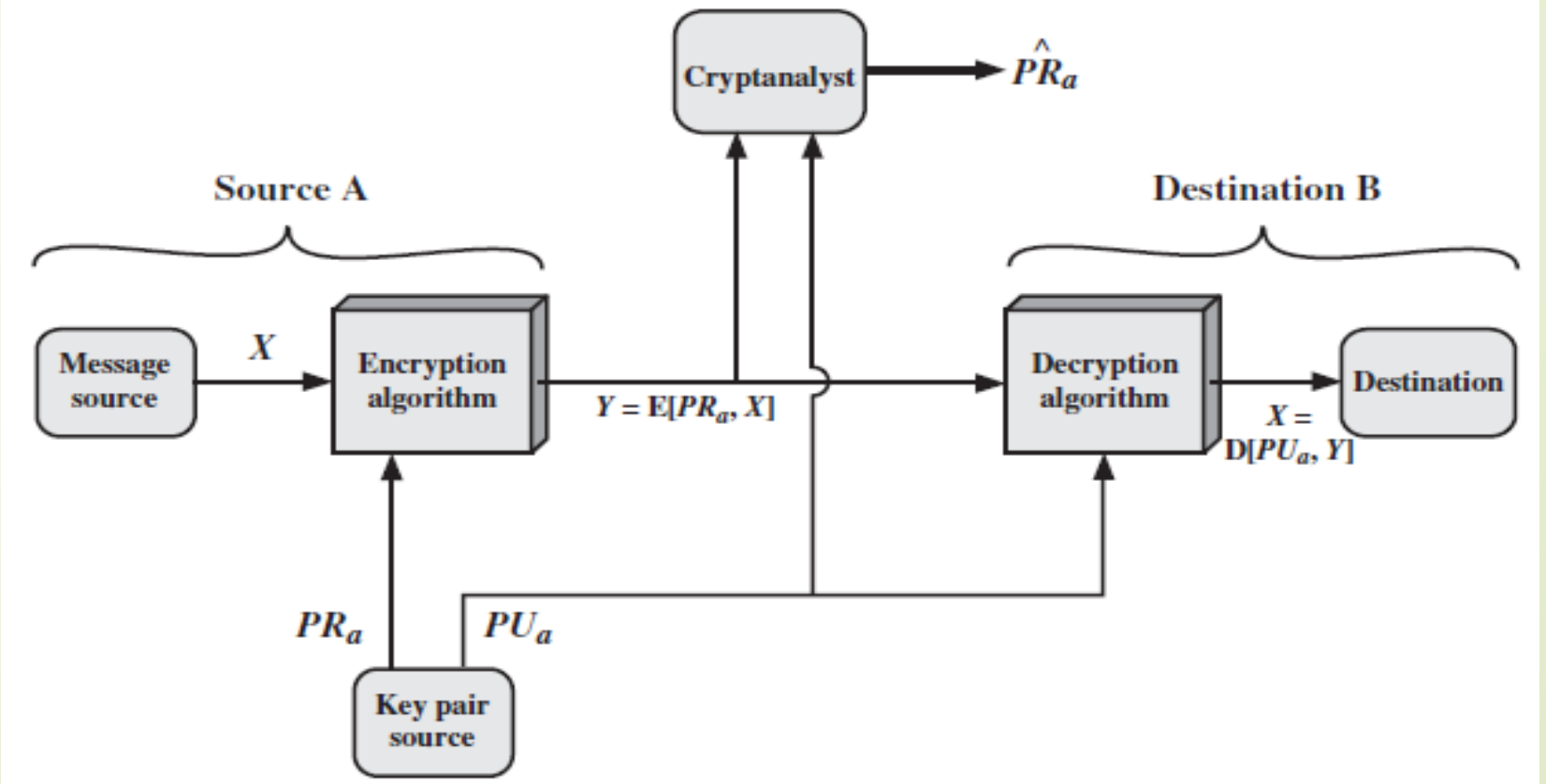
# Principles of Public Key cryptosystems

- **Public-Key Cryptosystems: Secrecy**



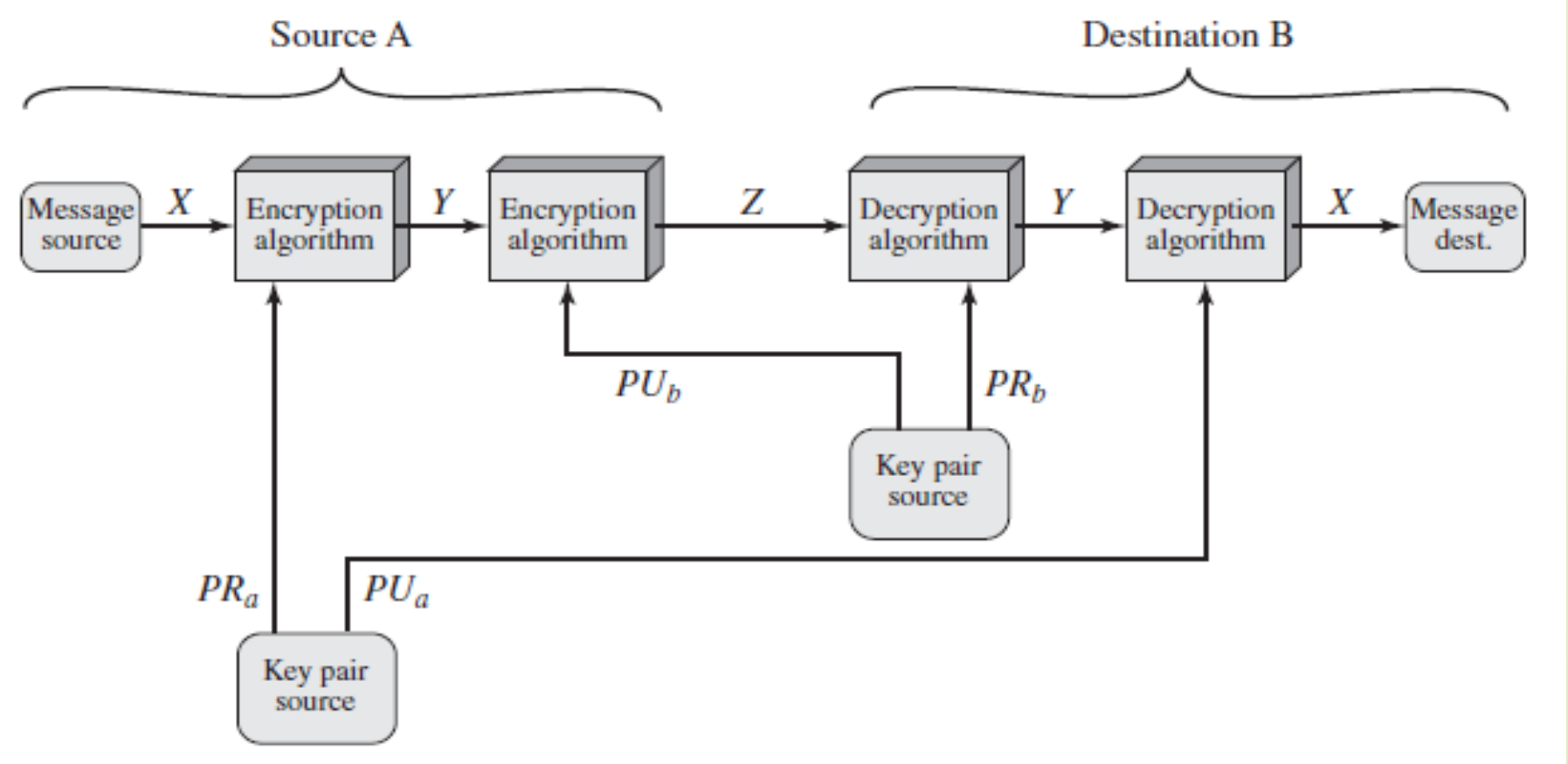
# Principles of Public Key cryptosystems

- **Public-Key Cryptosystems: Authentication**



# Principles of Public Key cryptosystems

- **Public-Key Cryptosystems: Secrecy and Authentication**



# Principles of Public Key cryptosystems

- **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.
- **Applications :**

| 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 |

# Principles of Public Key cryptosystems

- **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 (encrypt/decrypt) and hard (cryptanalyse) 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 private key schemes.

# The RSA Algorithm

- Developed by **Rivest, Shamir & Adleman** of MIT in 1977
- First published in 1978.
- best known & widely used public-key scheme
- The **RSA** scheme is a **block cipher** in which the plaintext and ciphertext are integers between 0 and *n* - 1 for some *n*.
- typical size for *n* is 1024 bits
- security due to cost of factoring large numbers

# The RSA Algorithm

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

# The RSA Algorithm

- **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: $M = C^d \bmod N$
  - the message M must be smaller than the modulus N (block if needed)

# The RSA Algorithm

- **Brief summary of RSA**

**Key Generation Alice**

| | |
|---|---|
| Select $p, q$ | $p$ and $q$ both prime, $p \neq q$ |
| *Calculate* $n = p \times q$ | |
| Calcuate $\phi(n) = (p-1)(q-1)$ | |
| 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 by Bob with Alice's Public Key**

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

**Decryption by Alice with Alice's Public Key**

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

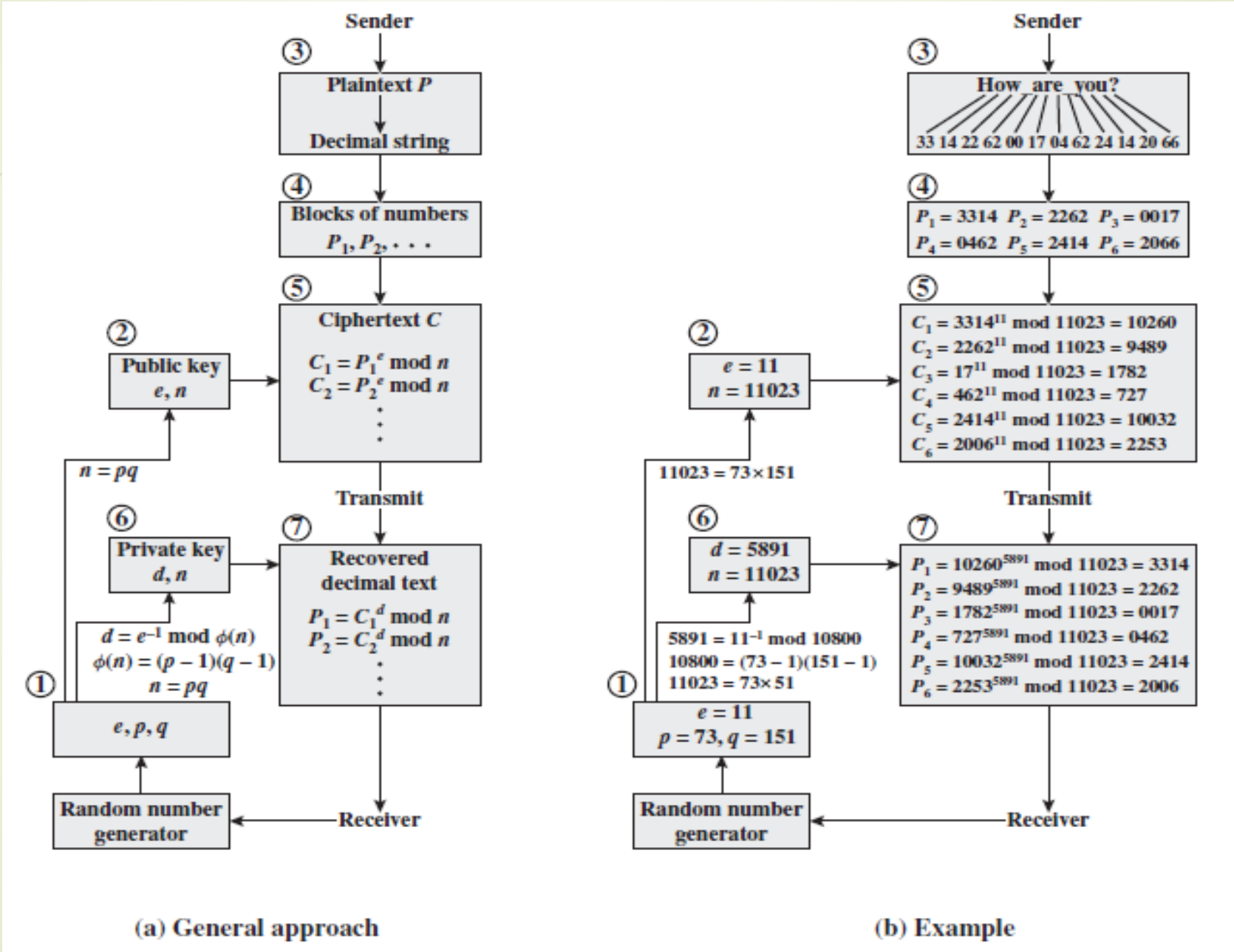
# The RSA Algorithm (Example)

- Select primes: p=17 & q=11
- Compute n = pq =17×11=187
- Compute ø(n)=(p–1)(q-1)=16×10=160
- Select e : gcd(e,160)=1; choose e=7
- Determine d:

  de=1 mod 160 and d < 160

  Value is d=23 since 23×7=161= 10×160+1
- Publish public key KU={7,187}
- Keep secret private key KR={23,17,11}

# RSA Processing of Multiple Blocks



# The Security of RSA

- Four approaches to attacking RSA:
  - **Brute force:** This involves trying all possible private keys.
  - **Mathematical attacks:** There are several approaches, all equivalent in effort to factoring the product of two primes.
  - **Timing attacks:** These depend on the running time of the decryption algorithm.
  - **Chosen ciphertext attacks:** This type of attack exploits properties of the RSA algorithm.

# The Security of RSA

- **Factoring Problem**
  - mathematical approach takes 3 forms:
    - factor N=p.q, hence find ø(N) and then d
    - determine ø(N) directly and find d
    - find d directly

| 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 |

# The Security of RSA

- **Timing Attacks**
  - developed in mid-1990's
  - exploit timing variations in operations
    - eg. multiplying by small vs large number
  - 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

# Diffie-Hellman Key Exchange

- First public-key type scheme.
- Proposed by Diffie & Hellman in 1976 along with the exposition of public key concepts.
- is a practical method for public exchange of a secret key.
- used in a number of commercial products.

- The **purpose of the algorithm** is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of secret values.

# Diffie-Hellman Key Exchange

- Two publicly known numbers: **a prime number q** and **an integer α** that is a primitive root of q.
- Each user (eg. A) generates their key:
  - chooses a secret key (number): $\mathbf{X_A < q}$
  - compute their public key: $\mathbf{Y_A = a^{XA} \bmod q}$
- Similarly, B also generates the keys.
- Each side keeps the **X** value private and makes the **Y** value available publicly to the other side.
- User **A** computes the key as, $\boldsymbol{K = (Y_B)^{XA} \bmod q}$
- User **B** computes the key as, $\mathbf{K = (Y_A)^{XB} \bmod q}$

# Diffie-Hellman Key Exchange

- **Key Exchange Algorithm**

| Global Public Elements | |
|---|---|
| $q$ | prime number |
| $\alpha$ | $\alpha < q$ and $\alpha$ a primitive root of $q$ |

| User A Key Generation | |
|---|---|
| Select private $X_A$ | $X_A < q$ |
| Calculate public $Y_A$ | $Y_A = \alpha^{XA} \bmod q$ |

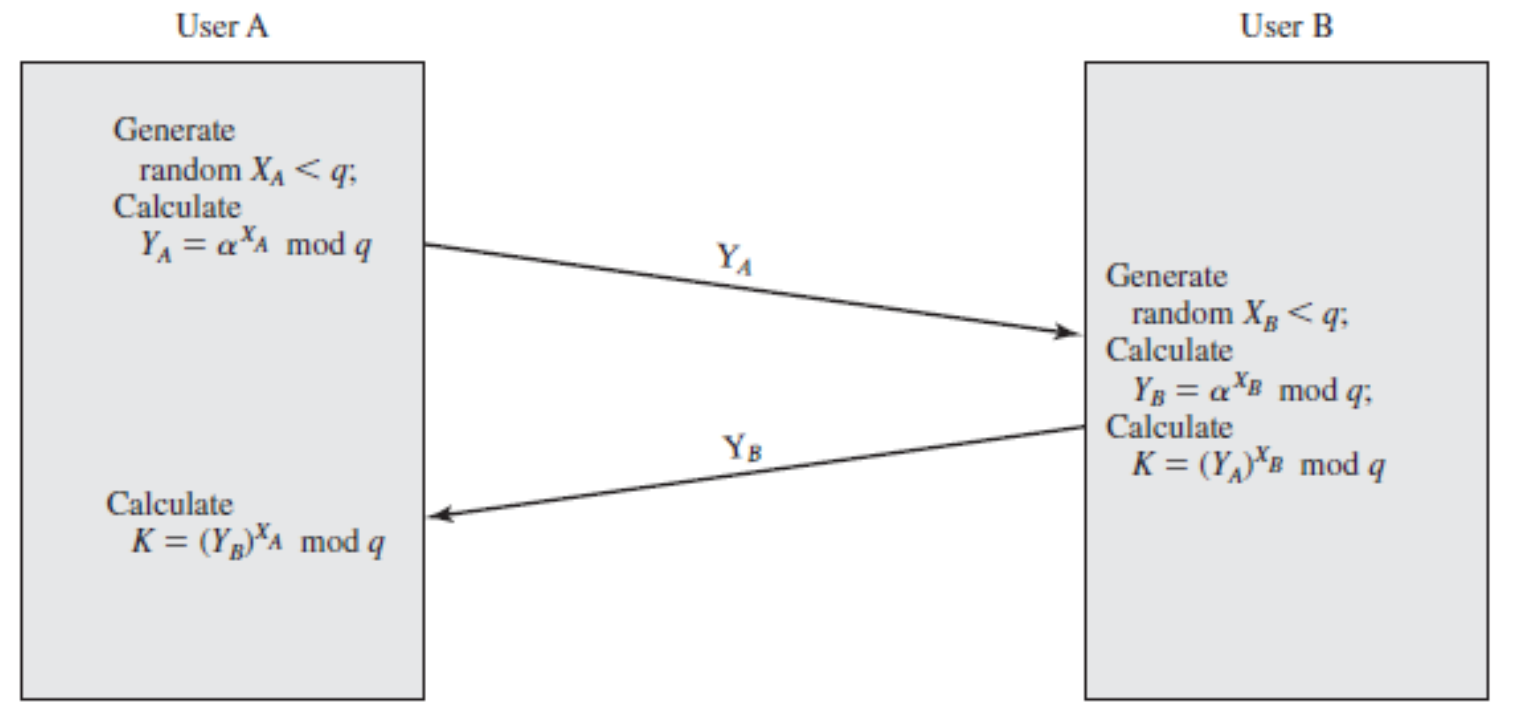
| User B Key Generation | |
|---|---|
| Select private $X_B$ | $X_B < q$ |
| Calculate public $Y_B$ | $Y_B = \alpha^{XB} \bmod q$ |

| Calculation of Secret Key by User A |
|---|
| $K = (Y_B)^{XA} \bmod q$ |

| Calculation of Secret Key by User B |
|---|
| $K = (Y_A)^{XB} \bmod q$ |

# Diffie-Hellman Key Exchange

- **Diffie-Hellman Key Exchange**



# Man-in-the-Middle Attack

- Suppose Alice and Bob wish to exchange keys, and Darth is the adversary.
- **The attack proceeds as follows:**
  - Darth 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.
  - Darth intercepts $Y_A$ and transmits $Y_{D1}$ to Bob. Darth also calculates

    $K2 = (Y_A)^{XD2} \bmod q$
  - Bob receives $Y_{D1}$ and calculates $K1 = (Y_{D1})^{XB} \bmod q$
  - Bob transmits $Y_B$ to Alice
  - Darth intercepts $Y_B$ and transmits $Y_{D2}$ to Alice. Darth calculates

    $K1 = (Y_B)^{XD1} \bmod q$
  - Alice receives $Y_{D2}$ and calculates $K2 = (Y_{D2})^{XA} \bmod q$

# Man-in-the-Middle Attack

- Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key **K1** and Alice and Darth 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)
  - Darth intercepts the encrypted message and decrypts it to recover
  - Darth sends Bob $E(K1, M)$ or E($K1$, $M$'), where M' is any message.
    - In the first case, Darth simply wants to eavesdrop on the communication without altering it.
    - In the second case, Darth 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.

# Elliptic Curve Cryptography

- Majority of public-key crypto (RSA, Diffie-Hellman) use either integer or polynomial arithmetic with very large numbers/polynomials
- imposes a significant load in storing and processing keys and messages
- an alternative is to use elliptic curves
- offers same security with smaller bit sizes

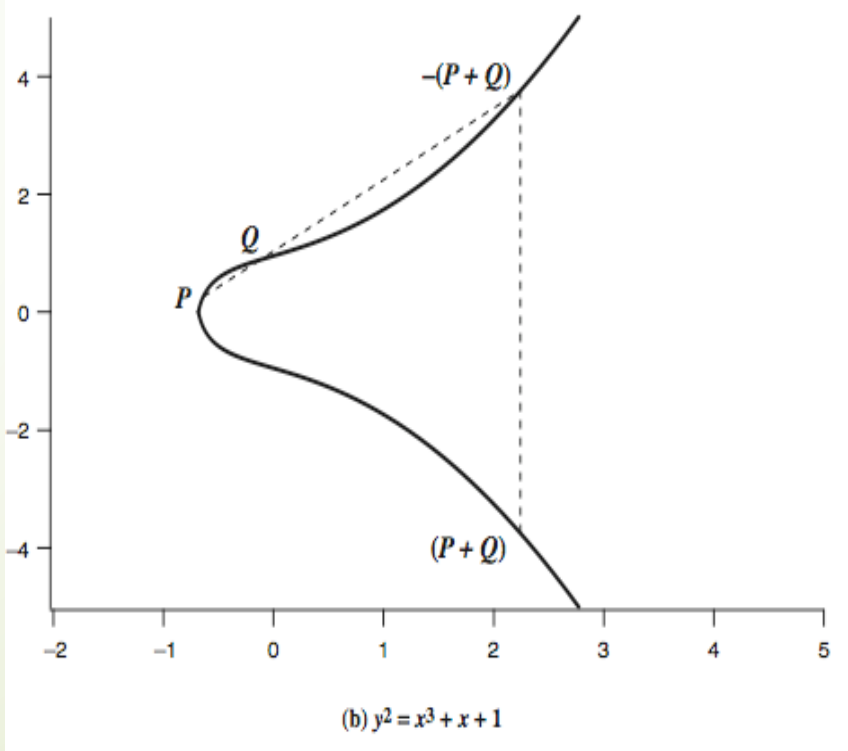
# Real Elliptic Curves

- an elliptic curve is defined by an equation in two variables x & y, with coefficients
- consider a cubic elliptic curve of form

  y2 = x3 + ax + b

  where x, y, a, b are all real numbers

  also define zero point O
- consider set of points E(a,b) that satisfy
- have addition operation for elliptic curve
  - geometrically sum of P+Q is reflection of the intersection R



(b) $y^2 = x^3 + x + 1$

# Finite Elliptic Curves

- Elliptic curve cryptography uses curves whose variables & coefficients are finite
- have two families commonly used:
  - prime curves $E_p(a,b)$ defined over $Z_p$
  - use integers modulo a prime
  - best in software
  - binary curves $E_{2m}(a,b)$ defined over $GF(2^n)$
  - use polynomials with binary coefficients
  - best in hardware

# Elliptic Curve Cryptography

- ECC addition is analog of modulo multiply
- ECC repeated addition is analog of modulo exponentiation
- need “hard” problem equiv to discrete log
  - Q=kP, where Q,P belong to a prime curve
  - is “easy” to compute Q given k,P
  - but “hard” to find k given Q,P
  - known as the elliptic curve logarithm problem
- Certicom example: E23(9,17)

# ECC Diffie-Hellman Key Exchange

- Can do key exchange analogous to D-H
- users select a suitable curve $E_q(a,b)$
- select base point $G=(x_1,y_1)$
- A & B select private keys $n_A < n$, $n_B < n$
- compute public keys: $P_A=n_AG$, $P_B=n_BG$
- compute shared key: $K=n_AP_B$, $K=n_BP_A$
  - same since $K=n_An_BG$
- attacker would need to find k, hard

**Global Public Elements**

| | |
|---|---|
| $E_q(a, b)$ | elliptic curve with parameters $a$, $b$, and $q$, where $q$ is a prime or an integer of the form $2^m$ |
| $G$ | point on elliptic curve whose order is large value $n$ |

**User A Key Generation**

| | |
|---|---|
| Select private $n_A$ | $n_A < n$ |
| Calculate public $P_A$ | $P_A = n_A \times G$ |

**User B Key Generation**

| | |
|---|---|
| Select private $n_B$ | $n_B < n$ |
| Calculate public $P_B$ | $P_B = n_B \times G$ |

**Calculation of Secret Key by User A**

$K = n_A \times P_B$

**Calculation of Secret Key by User B**

$K = n_B \times P_A$

# ECC Encryption/Decryption

- several alternatives, will consider simplest
- must first encode any message M as a point on the elliptic curve $P_m$
- select suitable curve & point G as in D-H
- each user chooses private key $n_A<n$
- and computes public key $P_A=n_AG$
- to encrypt $P_m$ : $C_m=\{kG, P_m+kP_b\}$, k random
- decrypt $C_m$ compute:

$$P_m+kP_b-n_B(kG) = P_m+k(n_BG)-n_B(kG) = P_m$$

# ECC Security

- relies on elliptic curve logarithm problem.
- fastest method is “Pollard rho method”.
- compared to factoring, can use much smaller key sizes than with RSA etc.
- for equivalent key lengths computations are roughly equivalent.
- hence for similar security ECC offers significant computational advantages.


Thank You