

EE 4215

Security Technology

Lecture 5: Public-key Cryptosystems
2018/19 Semester B

Course Structure

- Week 1: Security Fundamental
- Week 2: Classical Cryptography
- Week 3: Number Theory & DES
- Week 4: Secret-key Cryptography
- Week 5: Public-key Cryptography
- Week 6: Quiz 1
- Week 6: Hashing Algorithm
- Week 7: PKI, SSL/TLS
- Week 8: Web Security
- Week 9: Wi-Fi & P2P Security
- Week 10: Cloud Computing Security
- Week 10: Quiz 2
- Week 11: Smart-card Security
- Week 12: Blockchain Security Technology
- Week 13: Secure-code, Trusted Hardware, Review

Lecture 5: Public-key Cryptosystems

- Diffie–Hellman key exchange (D-H)
- Asymmetric (Public-key) Cryptosystem
- RSA Public Key Cryptosystem
- Elliptic curve cryptography (ECC)
- Tools: SSH, PGP / GnuPG

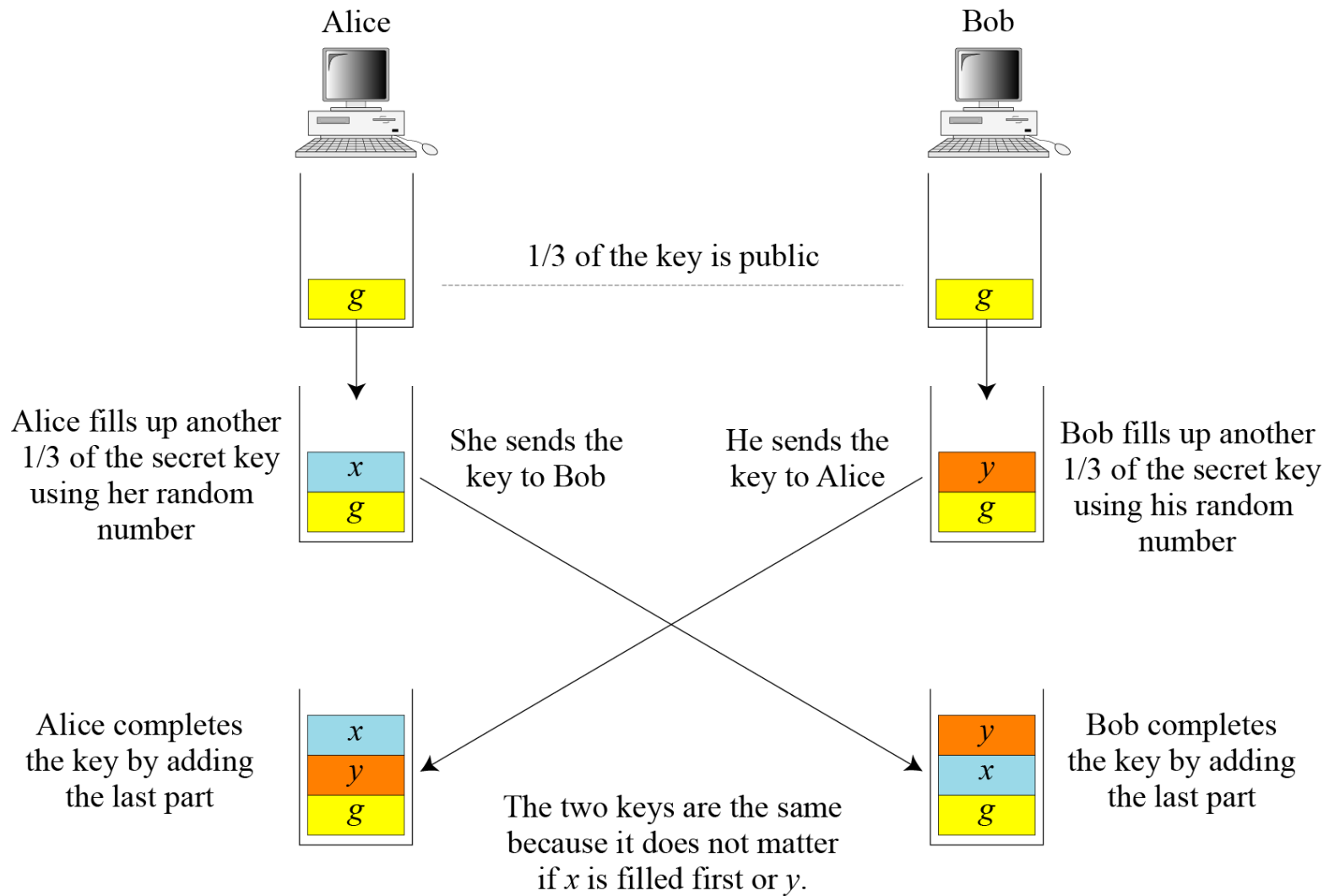
Problem Statement

- How to share secret over an unsecured communication channel so that sender and receiver can use it as a key for symmetric encryption, in such a way that an eavesdropper cannot know the key?
 - i.e. attacker is computationally infeasible to compute the key from the information overheard
- A practical solution was proposed in 1976:
 - *<http://www-ee.stanford.edu/~hellman/publications/24.pdf>*

Revision of Module Arithmetic

- Sometimes known informally as "clock arithmetic"
 - In modular arithmetic, numbers "wrap around" upon reaching a given fixed value, which is known as the "modulus"
- In modular arithmetic, numbers are never bigger than the "modulus"
 - For example, in arithmetic modulo (or mod) 12, the allowable numbers are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, and 11
 - $N \pmod{12} = \text{+ve remainder of } N \div 12$
 - Write $N = m * q + r$, then $N \pmod{m} = r$

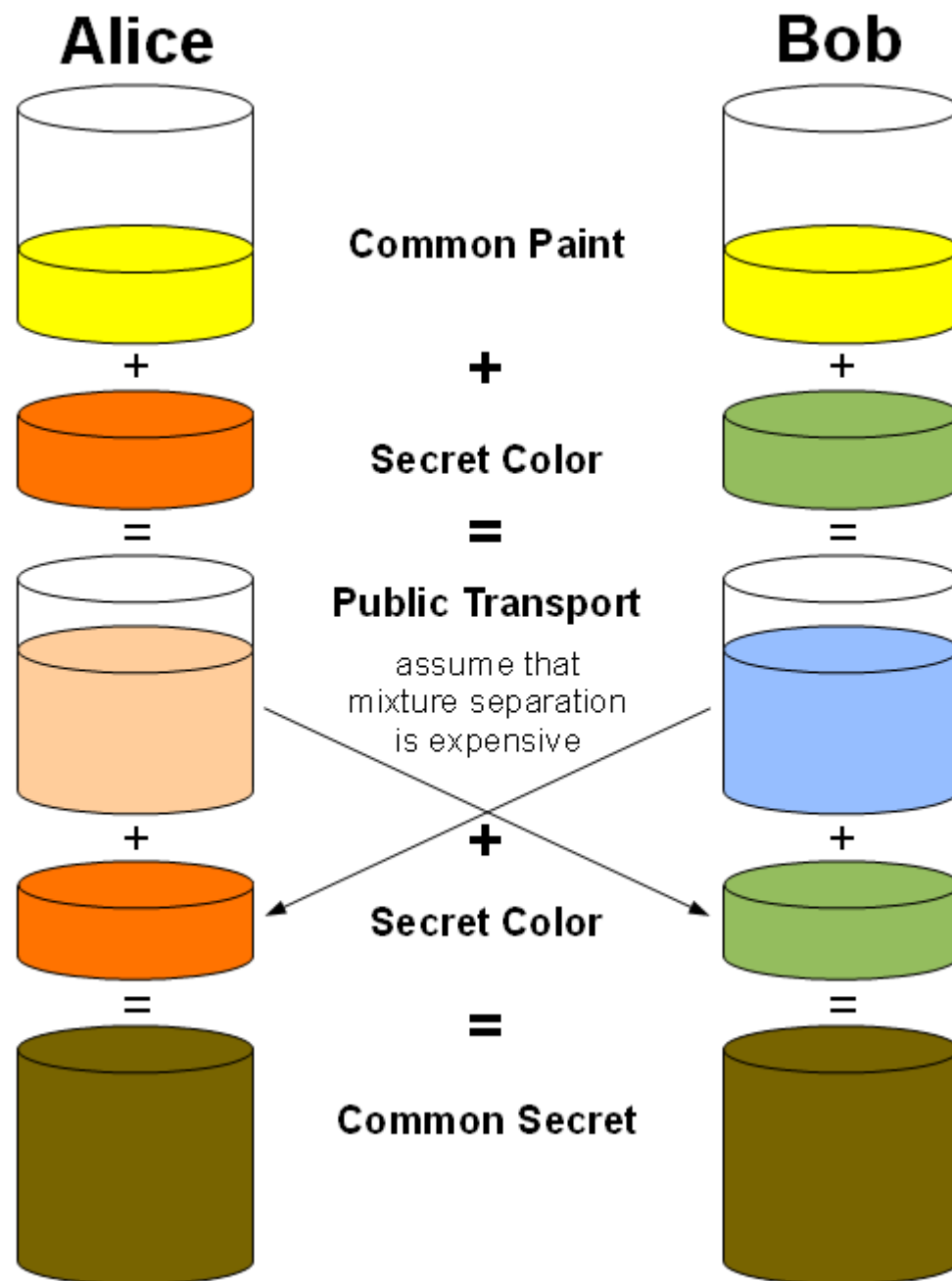
The Idea



Another Idea

Video Example:

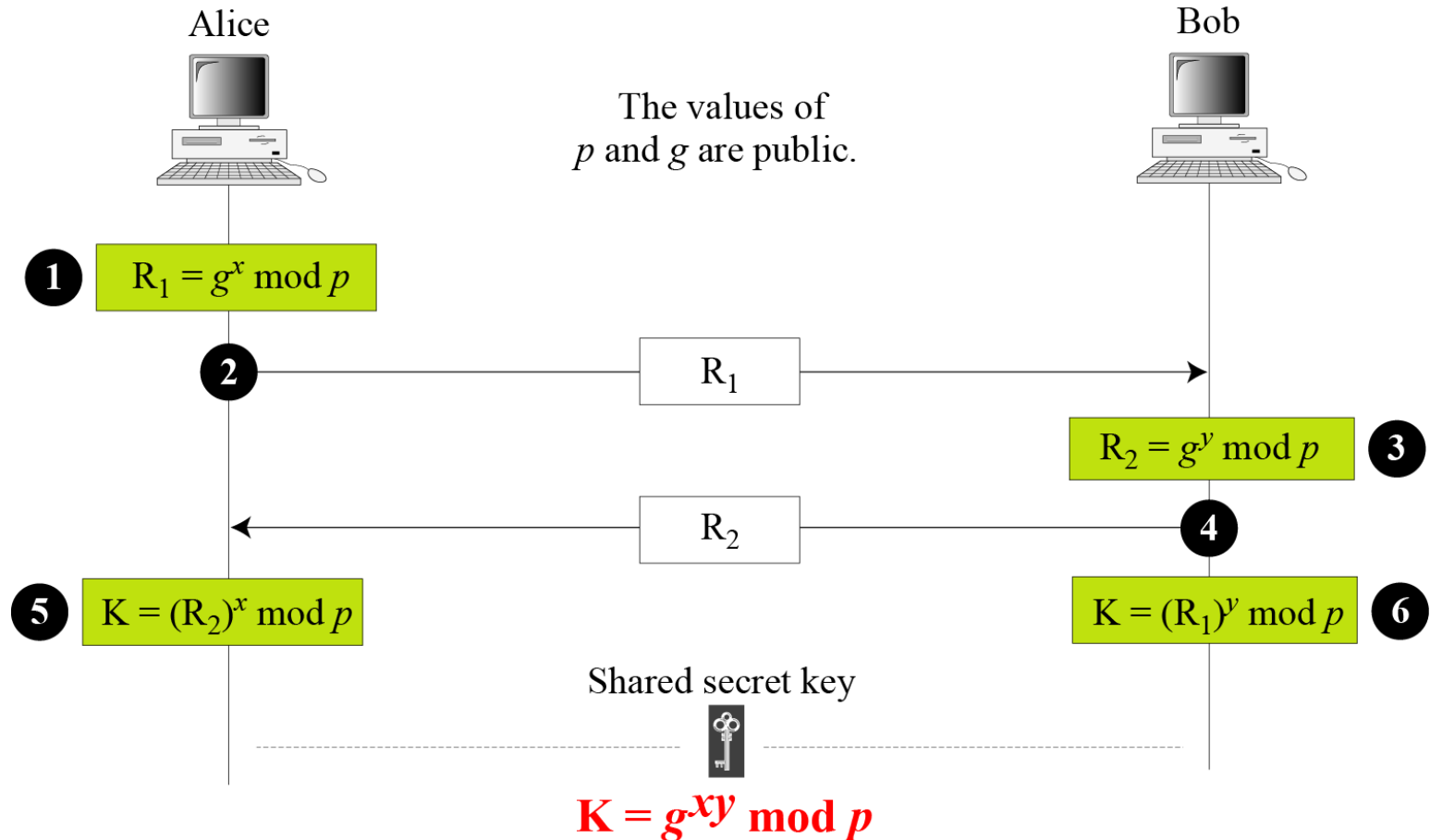
<https://www.youtube.com/watch?v=M-0qt6tdHzk>



Diffie-Hellman (D-H) Method

- Diffie-Hellman technique makes use of the apparent difficulty of computing logarithms over a finite field $\text{GF}(p)$ with a prime number p of elements
 - The idea of one-way trap-door functions
 - Let g be a fixed primitive element (or generator) of a finite field $\text{GF}(p)$
 - i.e. all non-zero elements of $\text{GF}(p)$ can be written as g^i , where $i > 0$
 - If $Y = g^X \pmod p$, for $1 \leq X \leq p - 1$
 - Then X is referred to as the logarithm of Y to the base g , mod p
 - or $X = \log_g Y \pmod p$, for $1 \leq Y \leq p - 1$
- Belief: Calculation of Y from X is easy, but computing X from Y , on the other hand can be much more difficult

Diffie-Hellman (D-H) Method



Diffie-Hellman (D-H) Method

- Using 1024 bits or 2048 bits numbers, with values g and p as public information
- Then the symmetric (shared) key using Diffie-Hellman method will be

$$K = g^{xy} \bmod p$$

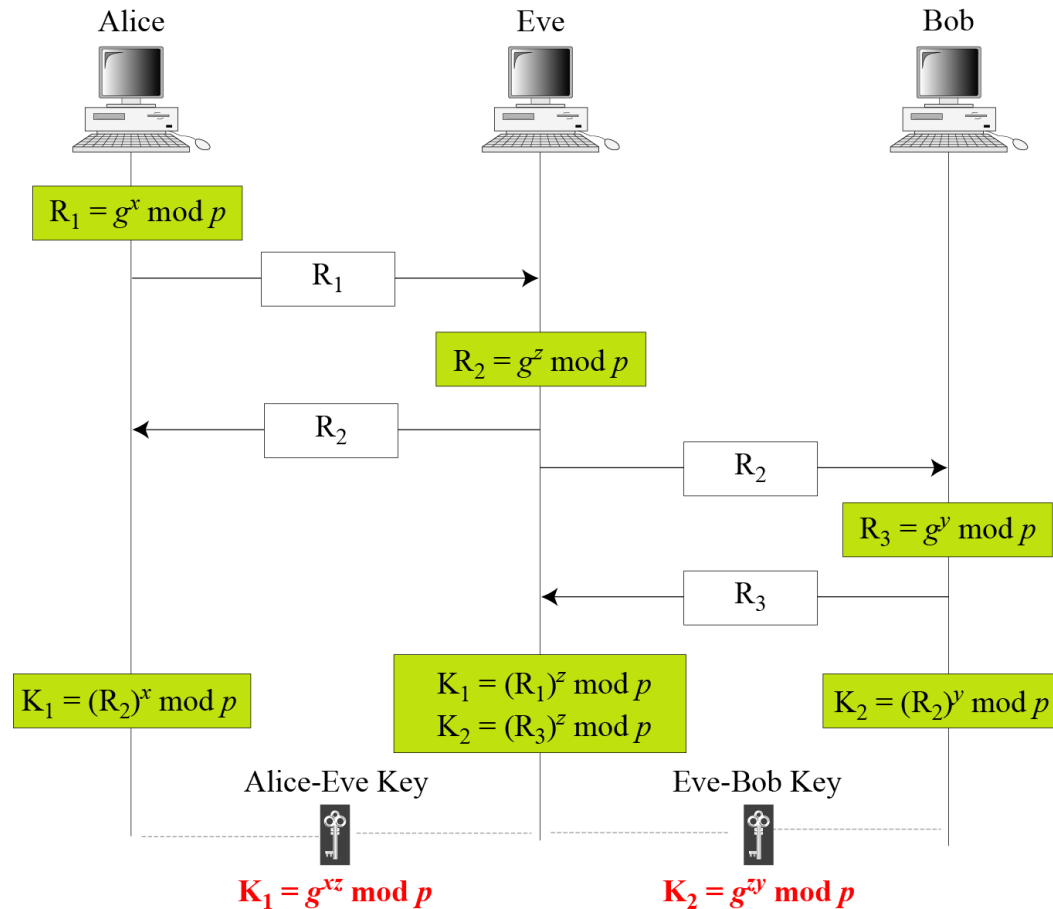
Example

- Assume that prime $p = 23$, select $g = 7$
- Alice chooses $x = 3$ and calculates
 - $R_1 = g^x \pmod{23} = 7^3 \pmod{23} = 343 \pmod{23} = 21$
- Alice sends the number 21 to Bob
- Bob chooses $y = 6$ and calculates
 - $R_2 = g^y \pmod{23} = 7^6 \pmod{23} = (7^2)^3 \pmod{23} = 3*3*3 \pmod{23} = 4$
- Bob sends the number 4 to Alice
- Alice calculates the symmetric key $K = (\text{number received from Bob})^{(\text{Alice's own chosen number})} = 4^x \pmod{23} = 4^3 \pmod{23} = 18$
- Bob calculates the symmetric key $K = 21^y \pmod{23} = 21^6 \pmod{23} = 18$

Attack

- Discrete Logarithm Attack
 - Any computationally efficient algorithm to x from $R_1 = g^x \pmod{p}$?
- Man-in-the middle Attack
 - **Requirement:** Alice and Bob were tricked to believe that the “man-in-the-middle” called Eve is the trusted partner

Man-in-the-middle Attack



Implementing D-H

- need random numbers
- may also need to find prime and generator (use known ones)
- want **big** prime (e.g.1024 bits)
- need multi-precision integer library (UNIX mp, GNU gmp) or perl BigInt, or C++ Integer, or Java BigInteger

Random Numbers

- Why?
 - Good cryptography requires good random numbers.
- Generating (pseudo) random numbers from a random seed
 - predictable = vulnerable
 - **equally likely** to choose any element (uniform distribution)
 - **independence** – cannot infer one value in the sequence from others
- Sources of random bits

Crypto Toolkit

secret-key crypto
public-key crypto
big-number math
random numbers ✓
prime numbers
hash functions

Pseudorandom numbers (PRNG)

- Computer is deterministic
- `rand()` statistically OK, but cryptographically weak
 - based on linear congruential method
 - $R_{i+1} = (aR_i + b) \bmod n$
 - random numbers between 0 and $n-1$
 - eventually repeats
 - even if good seed, R_0 , can solve for a , b , n if you can capture three R 's
- Repeatability good for testing (games/simulation), bad for crypto
- Predictable ☹️

Almost random sources

- Source for a random seed
 - keystroke timing
 - mouse tracking
 - frame buffer
 - disk times
 - /dev/random
 - system status (netstat, ps, iostat)
 - ...
- Early versions of Netscape's Secure Socket Layer (SSL) encryption protocol was broken due to their weakness of random number generation

True random numbers

- Sources for a random seed
 - coin toss, dice
 - radioactive source
 - noisy diode
 - unplugged microphone (/dev/audio)
 - lava lamps (video, SHA, BBS)
 - a few hardware sources for computers (QNG), Intel i810, AMD768
 - thermal noise (resistor)
 - sampling oscillators
 - Serial port devices
 - ...
- NIST SP800-90A "Recommendation for Random Number Generation Using Deterministic Random Bit Generators"
 - <http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf>

Best Practices

- Avoid math library PRNGs
- Avoid linear-congruential PRNGs
- Make sure your hardware sources are working (FIPS 140-2)
- Use / mix several sources
- ...

Prime Numbers

- Many cryptographic operations (e.g. key generation) need large primes
- Usually, need to do only once
- Usually part of crypto library
- The question "**Is n prime?**" is easier than "What are the factors of n ?"

Prime Number Search

- <https://www.mersenne.org/primes/>



2^p-1
May Be Prime!

Great Internet Mersenne Prime Search

GIMPS

Finding World Record Primes Since 1996

Username
Password
Log In [Forgot password?](#)

[Home](#)
[Get Started](#)
[Current Progress](#)
[Account/Team Info](#)
[Reports](#)
[Manual Testing](#)
[More Information / Help](#)
[Donate](#)
Make a donation

List of Known Mersenne Prime Numbers						
#	2^p-1	Digits	Date Discovered	Discovered By	Method / Hardware	Perfect Number
1	2^2-1	1	c. 500 BCE	Ancient Greek mathematicians		$2^1 \cdot (2^2-1)$
2	2^3-1	1	c. 500 BCE	Ancient Greek mathematicians		$2^2 \cdot (2^3-1)$
3	2^5-1	2	c. 275 BCE	Ancient Greek mathematicians		$2^4 \cdot (2^5-1)$
4	2^7-1	3	c. 275 BCE	Ancient Greek mathematicians		$2^6 \cdot (2^7-1)$
5	$2^{13}-1$	4	1456	Anonymous	trial division	$2^{12} \cdot (2^{13}-1)$
6	$2^{17}-1$	6	1588	Pietro Cataldi	trial division	$2^{16} \cdot (2^{17}-1)$
7	$2^{19}-1$	6	1588	Pietro Cataldi	trial division	$2^{18} \cdot (2^{19}-1)$
8	$2^{31}-1$	10	1772	Leonhard Euler	Enhanced trial division	$2^{30} \cdot (2^{31}-1)$
9	$2^{61}-1$	19	1883	Ivan Mikheevich Pervushin	Lucas sequences	$2^{60} \cdot (2^{61}-1)$
10	$2^{89}-1$	27	1911 Jun	R. E. Powers	Lucas sequences	$2^{88} \cdot (2^{89}-1)$
11	$2^{107}-1$	33	1914 Jun 11	R. E. Powers	Lucas sequences	$2^{106} \cdot (2^{107}-1)$
12	$2^{127}-1$	39	1876 Jan 10	Édouard Lucas	Lucas sequences	$2^{126} \cdot (2^{127}-1)$
13	$2^{521}-1$	157	1952 Jan 30	Raphael M. Robinson	L-L / SWAC	$2^{520} \cdot (2^{521}-1)$
14	$2^{607}-1$	183	1952 Jan 30	Raphael M. Robinson	L-L / SWAC	$2^{606} \cdot (2^{607}-1)$

Primality Testing

Is p prime?

- Infeasible to check all factors for really BIG integer
- Inefficient method
 - p is prime if and only if $(p - 1)! \equiv -1 \pmod{p}$
- Probabilistic methods
 - Can't determine absolutely if big number is prime, but can test for “highly probable”
 - **Fermat primality test:**
 - Given p , choose a ($< p$) that is co-prime to p (i.e. the only positive integer that can divide both a and p is 1) and calculate $a^{p-1} \pmod{p}$
 - If the result is different from 1, then p is composite.
 - If it is 1, then p may or may not be prime

Finding a prime

1. generate random n-bit number p
 2. set hi-bit to 1, low-bit to 1
 3. verify p is not divisible by first 2048 primes
 4. perform primality test for some random a. If p passes, generate another a, and repeat test (5 times?). If it fails, generate a new p and go back to step 1.
- The above is roughly what OpenSSL library does in its *BN_generate_prime()* function

BIG integer software

UNIX `bc` command

perl

```
use Math::BigInt;  
  
a = Math::BigInt->new("4324567832342");  
c = a +1;  
print c;
```

C++

```
#include <Integer.h>  
  
Integer bigi, bigj, bigmod;  
bigi = (bigj*bigi)%bigmod +55;  
cout << bigi;
```

Java

```
import java.math.BigInteger;  
BigInteger includes mod exponentiation
```

DHTest.java

```
import java.io.*;
import java.security.*;
import java.math.BigInteger;

public class DHTest {
    public static void main(String[] args) throws IOException {
        BigInteger prime, generator, sharedsecret, mysecret,
        mypublic, hispublic, hissecret;

        generator = BigInteger.valueOf(3L);

        prime = new
        BigInteger("10600793837847095502437519478711009735280641408663083351958564847666239803449310577396474102435
        91579519017104566785358230810944217204201923626592702404169103524977904602117530815851356942508431726596951
        58837903510942872312899367927109380933811222002873447475498685844537943400748484227570055775161033105076319
        ");

        mysecret = new BigInteger(1024, new SecureRandom());
        mysecret = mysecret.mod(prime);
        mypublic = generator.modPow(myscret, prime);

        hissecret = new BigInteger(1024, new SecureRandom());
        hissecret = hissecret.mod(prime);
        hispublic = generator.modPow(hissecret, prime);

        sharedsecret = hispublic.modPow(myscret, prime);
        System.out.println("secret " + sharedsecret.toString());
        sharedsecret = mypublic.modPow(hissecret, prime);
        System.out.println("secret " + sharedsecret.toString());
    }
}
```

BIG integer software (C)

GNU's MP library

```
#include "gmp.h"

mpz_t x, y, z;

void mpz_init (mpz_t integer)
void mpz_set_ui (mpz_t rop, unsigned long int op)
int mpz_set_str (mpz_t rop, char *str, int base)
void mpz_powm (mpz_t rop, mpz_t base, mpz_t exp,
               mpz_t mod)
void mpz_random (mpz_t rop, mp_size_t max_size)
char * mpz_get_str (char *str, int base, mpz_t op)
```

link with *libgmp.a*

Lecture 5: Public-key Cryptosystems

- Diffie–Hellman key exchange (D-H)
- Asymmetric (Public-key) Cryptosystem
- RSA Public Key Cryptosystem
- Elliptic curve cryptography (ECC)
- Tools: SSH, PGP / GnuPG

Problem

- Symmetric key crypto (DES, AES) needs pre-shared key
- How do Alice and Bob get their key?
- For N people, need $O(N^2)$ sets of shared keys
- Mathematics will give us some other choices
 - Diffie-Hellman will allow Alice and Bob to establish a shared key
 - Public key crypto (RSA, ECC, ...) will allow each user to have two keys, publishing the public key while keeping the private key secret to himself ONLY

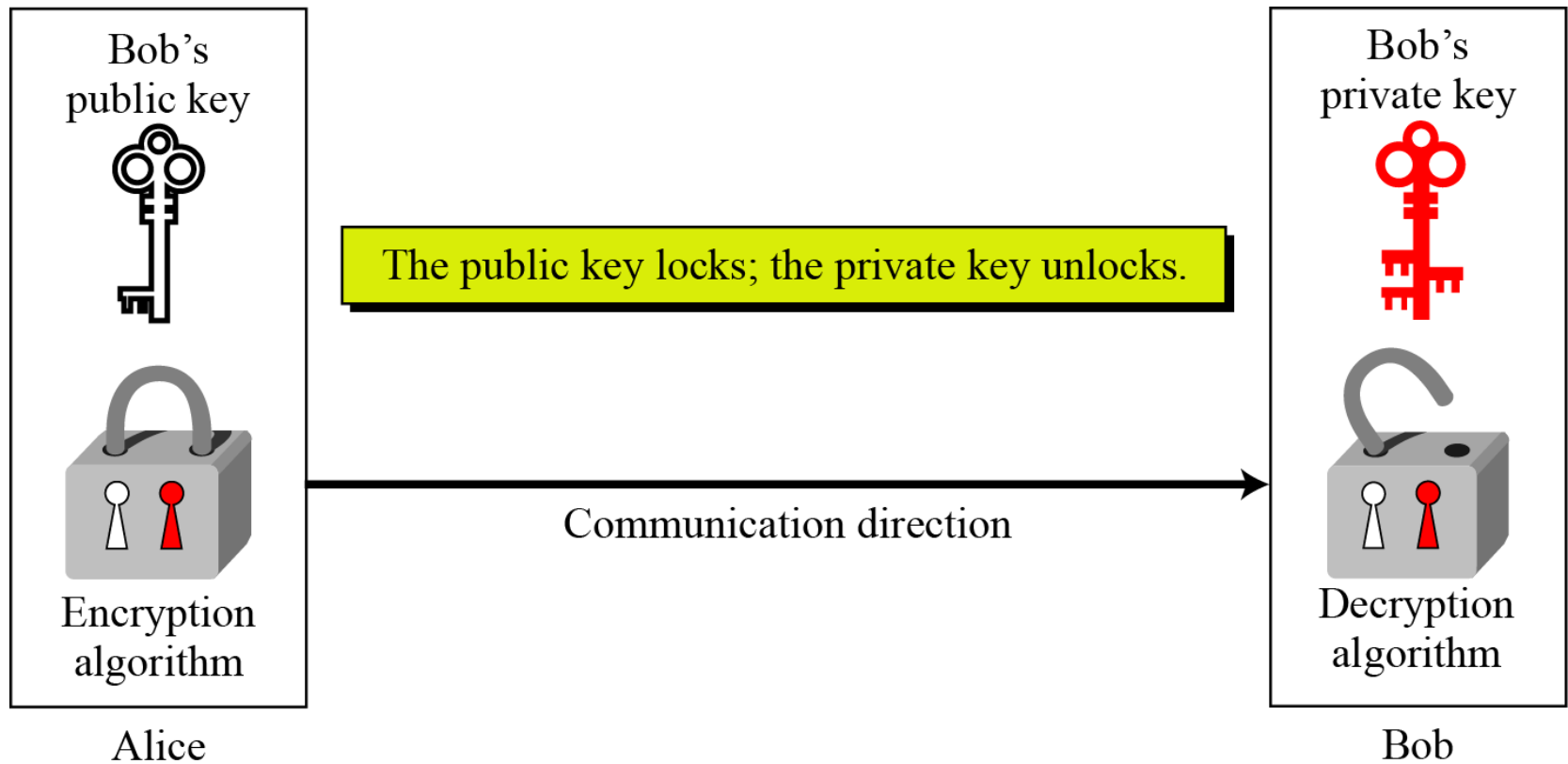
Remarks

- We are not starting to say that public key (or asymmetric) cryptosystem is better than secret key cryptosystem
- Both symmetric and asymmetric-key cryptography will exist in parallel
 - they are complements of each other
 - the advantages of one can compensate for the disadvantages of the other.
- Symmetric encryption algorithm is usually
 - faster
 - more suitable for hardware implementation
 - have short key lengththan asymmetric encryption algorithms

Comparison

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

The Idea



The 4 Keys

- There are 4 keys involved in two-party communications:
 - Bob's public key
 - Bob's private key
 - Alice's public key
 - Alice's private key
- **Encryption / Secrecy:** ALL in the world can encrypt an message with Alice's public key → only Alice can decrypt with her secret key
- **Digital Signature:** Bob encrypts an message with his secret key (only Bob can do it), ALL in the world can decrypt (or verify) the message using Bob's public key to verify that the message was actually encrypted / signed by Bob
- It makes no sense for Bob to encrypt a message with his public key

Terminology Related to Asymmetric Encryption

Asymmetric Keys

Two related keys, a public key and a private key that are used to perform complementary operations, such as encryption and decryption or signature generation and signature verification.

Public Key Certificate

A digital document issued and digitally signed by the private key of a Certification Authority that binds the name of a subscriber to a public key. The certificate indicates that the subscriber identified in the certificate has sole control and access to the corresponding private key.

Public Key (Asymmetric) Cryptographic Algorithm

A cryptographic algorithm that uses two related keys, a public key and a private key. The two keys have the property that deriving the private key from the public key is computationally infeasible.

Public Key Infrastructure (PKI)

A set of policies, processes, server platforms, software and workstations used for the purpose of administering certificates and public-private key pairs, including the ability to issue, maintain, and revoke public key certificates.

Principles of Public-Key Cryptosystems

- The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption:

Key distribution

- **How to have secure communications in general without having to trust a KDC with your key**

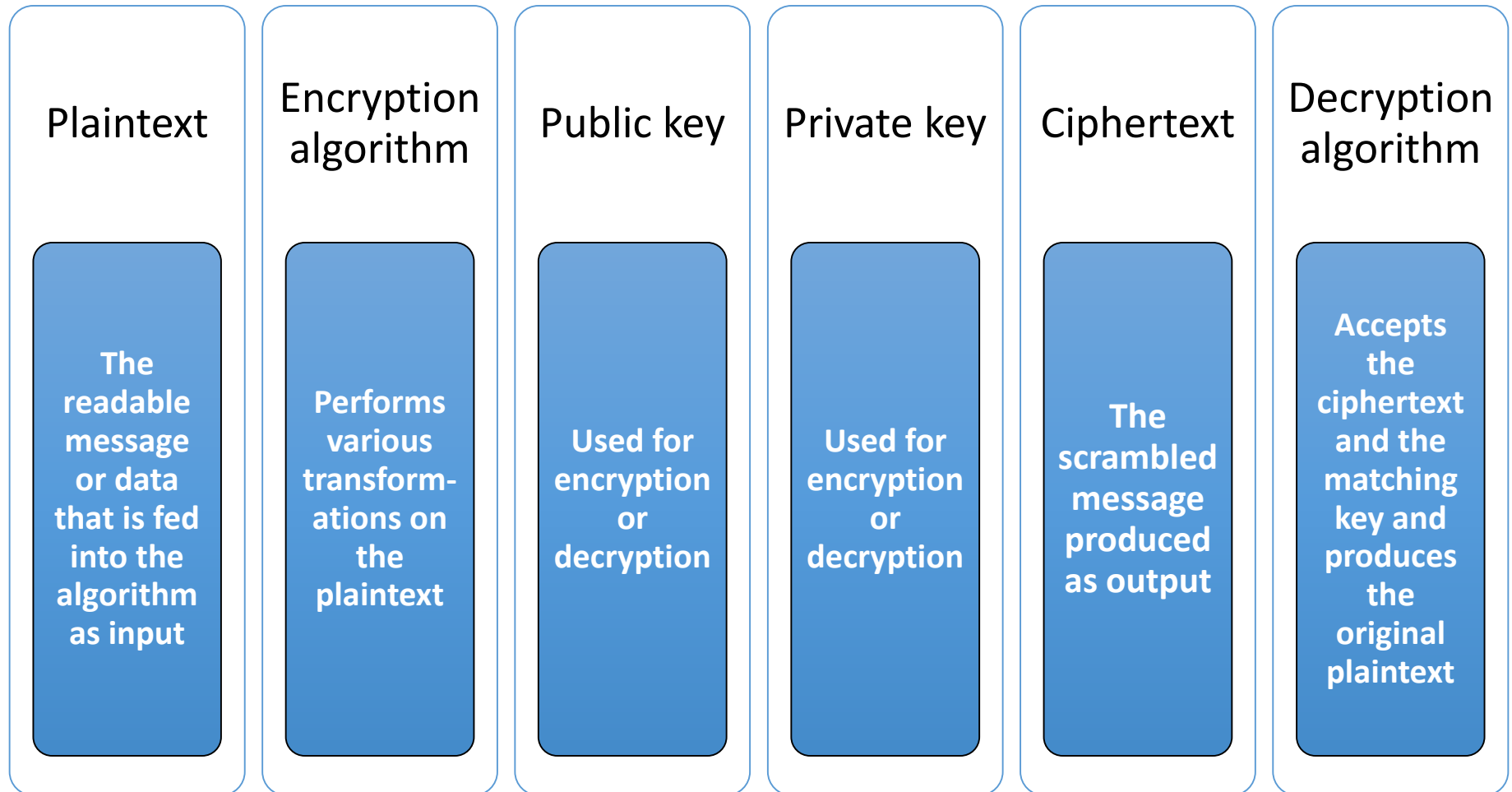
Digital signatures

- **How to verify that a message comes intact from the claimed sender**

- Whitfield Diffie and Martin Hellman from Stanford University achieved a breakthrough in 1976 by coming up with a method that addressed both problems and was radically different from all previous approaches to cryptography

Public-Key Cryptosystems

- A public-key encryption scheme has six ingredients:



Public-Key Cryptography

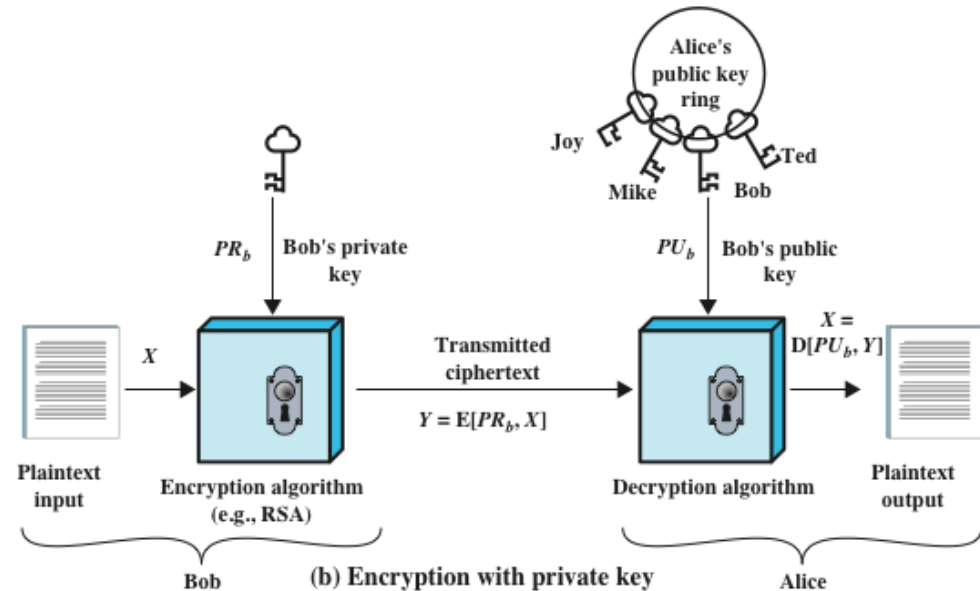
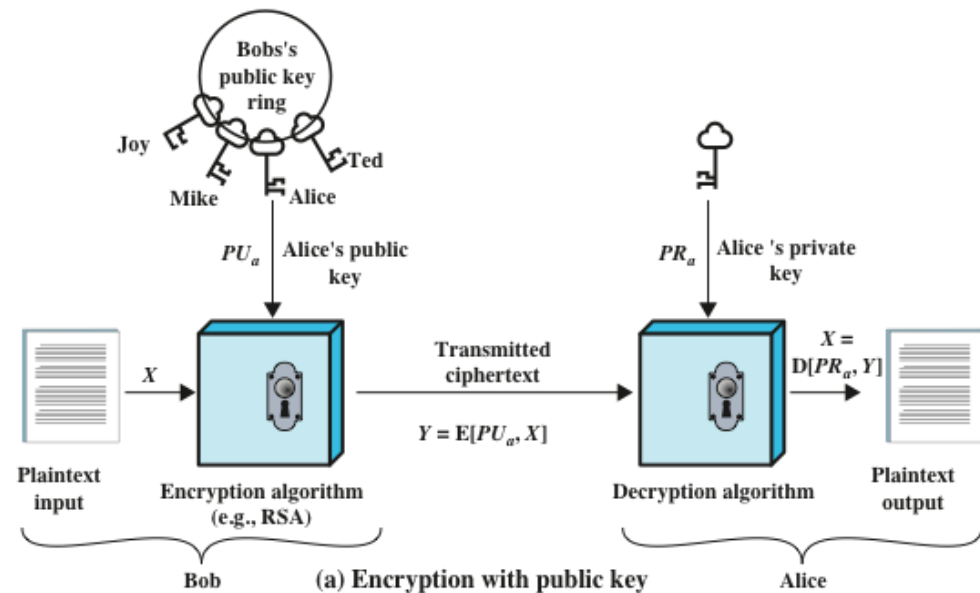
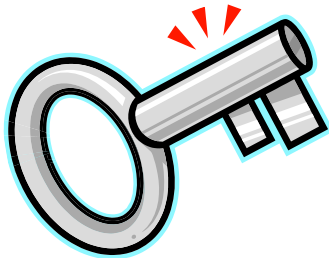


Figure 9.1 Public-Key Cryptography

Public-Key Cryptosystem: Secrecy

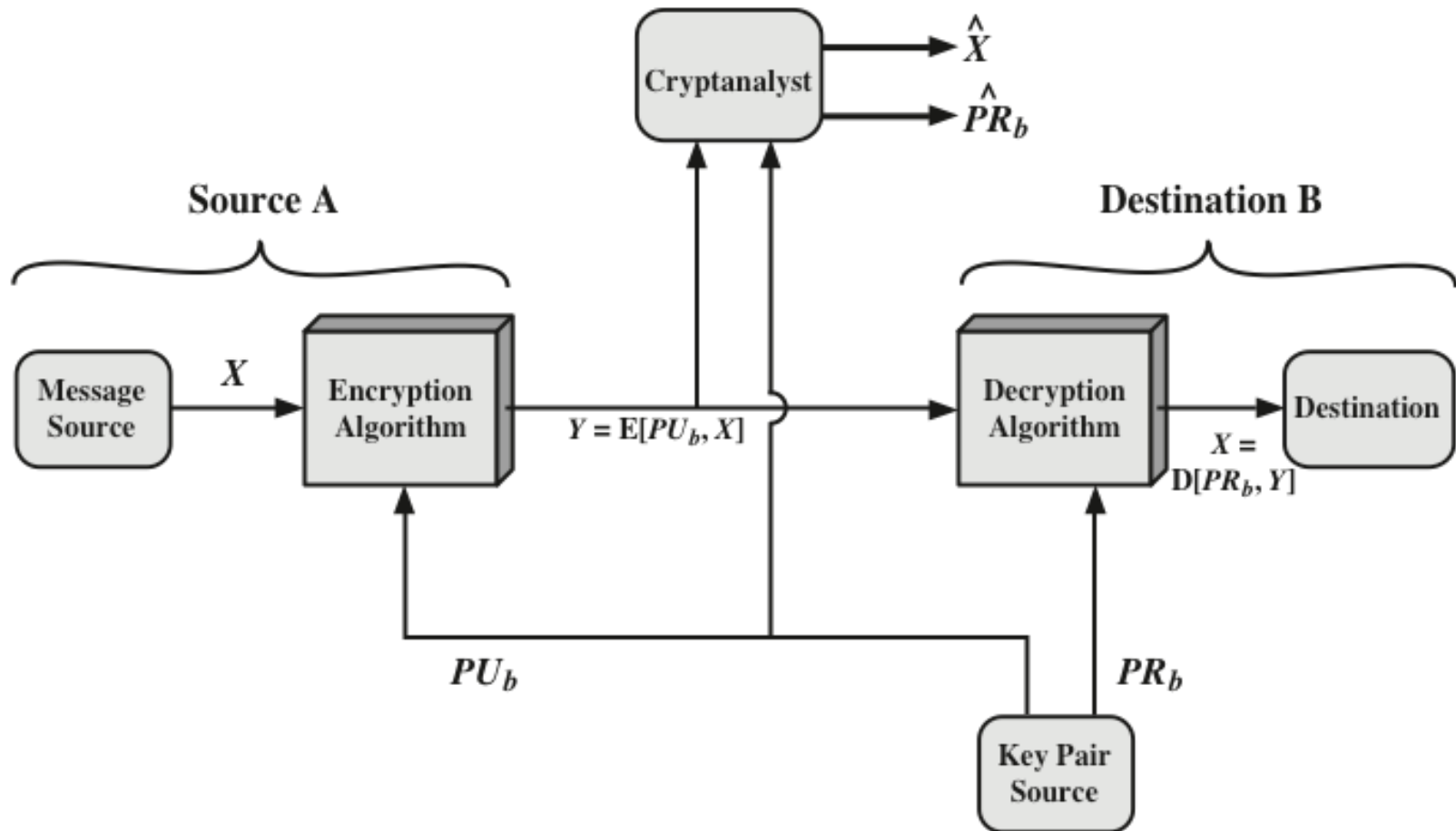


Figure 9.2 Public-Key Cryptosystem: Secrecy

Public-Key Cryptosystem: Authentication

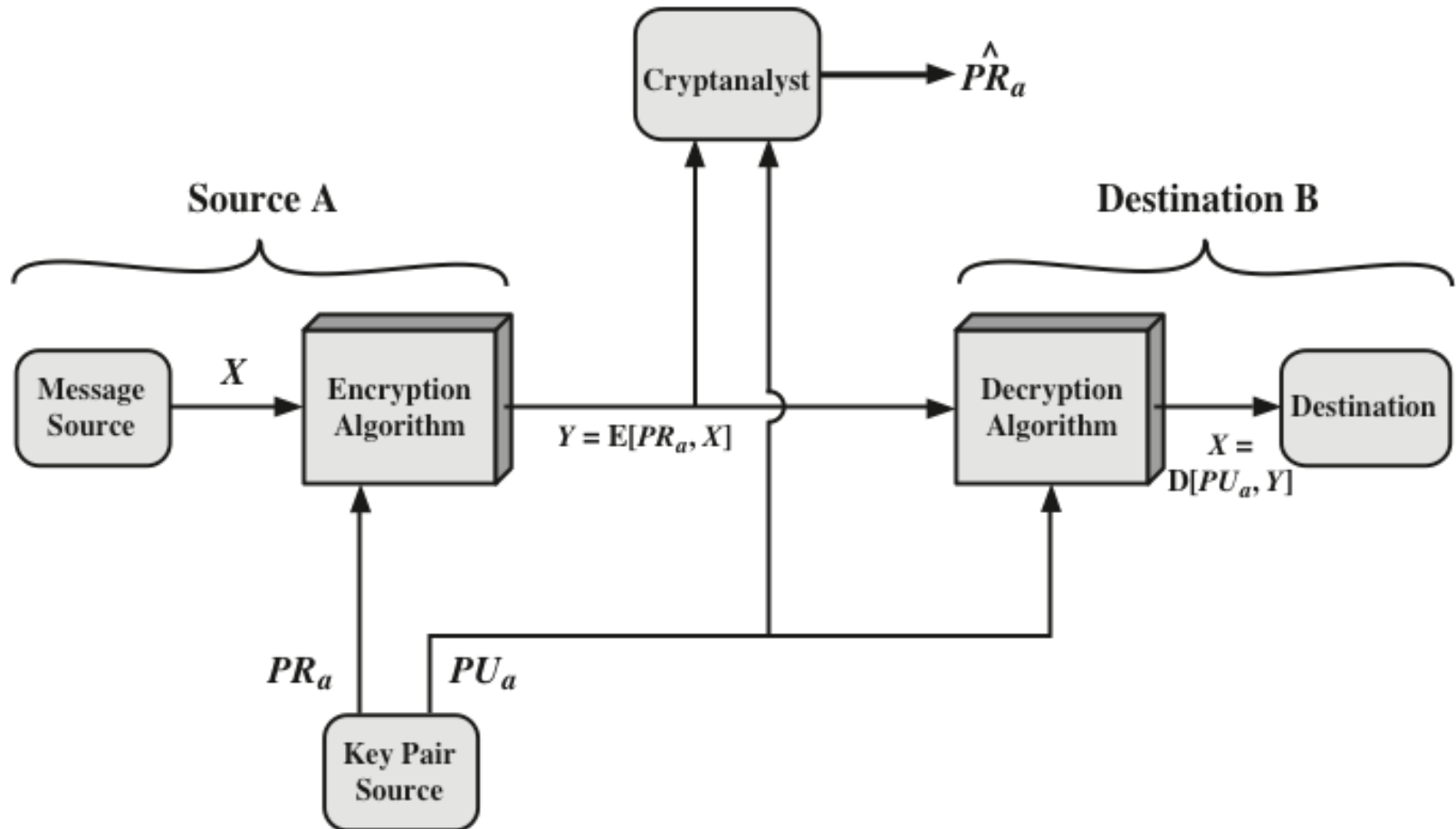


Figure 9.3 Public-Key Cryptosystem: Authentication

Public-Key Cryptosystem: Authentication and Secrecy

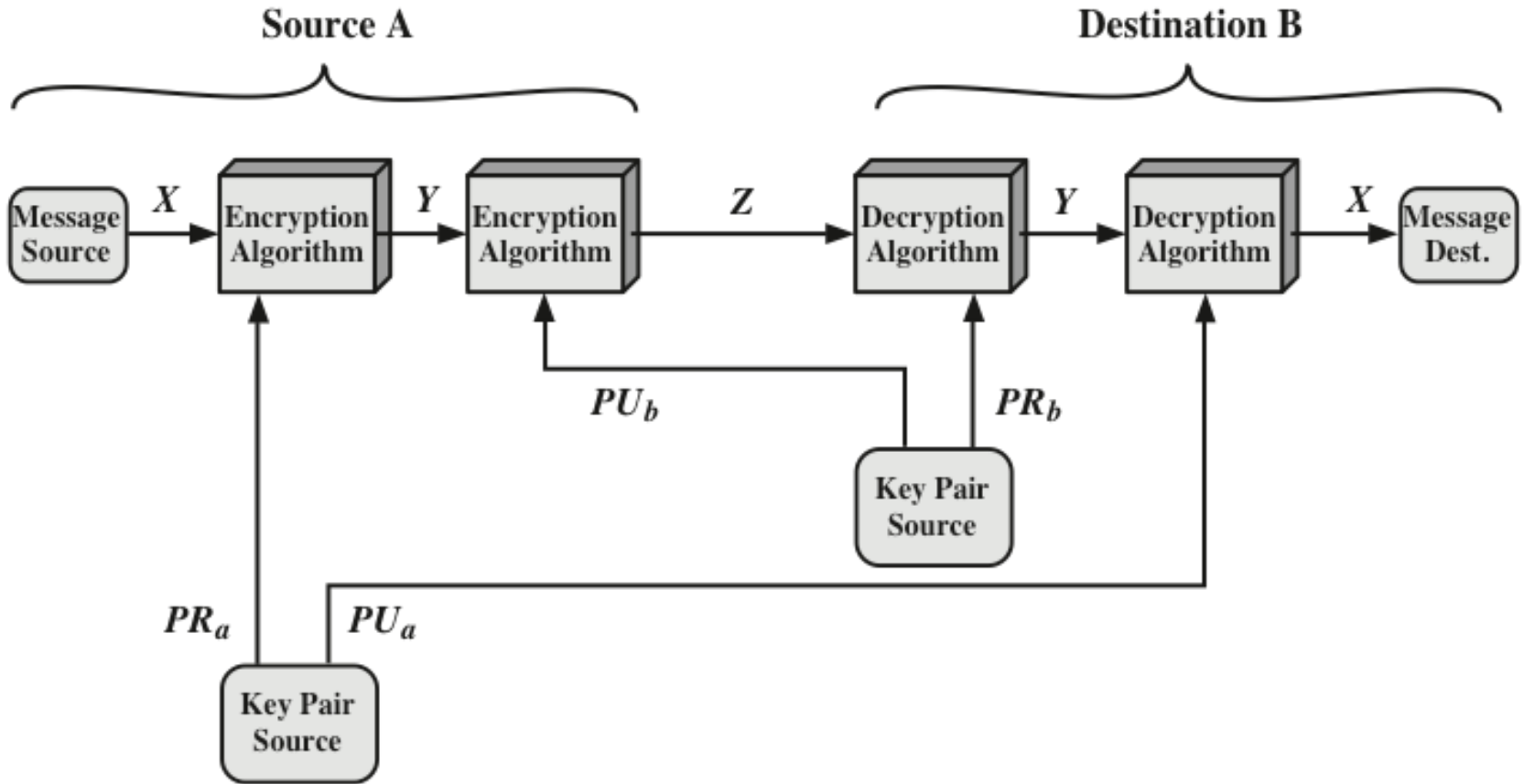
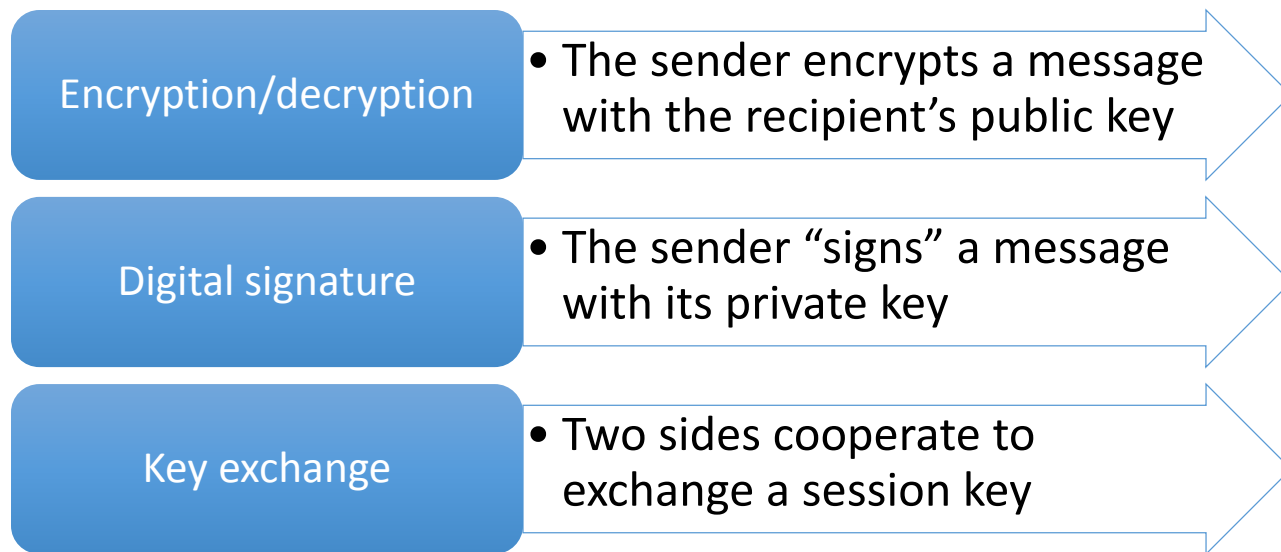


Figure 9.4 Public-Key Cryptosystem: Authentication and Secrecy

Applications for Public-Key Cryptosystems

- Public-key cryptosystems can be classified into three categories:



- Some algorithms are suitable for all three applications, whereas others can be used only for one or two

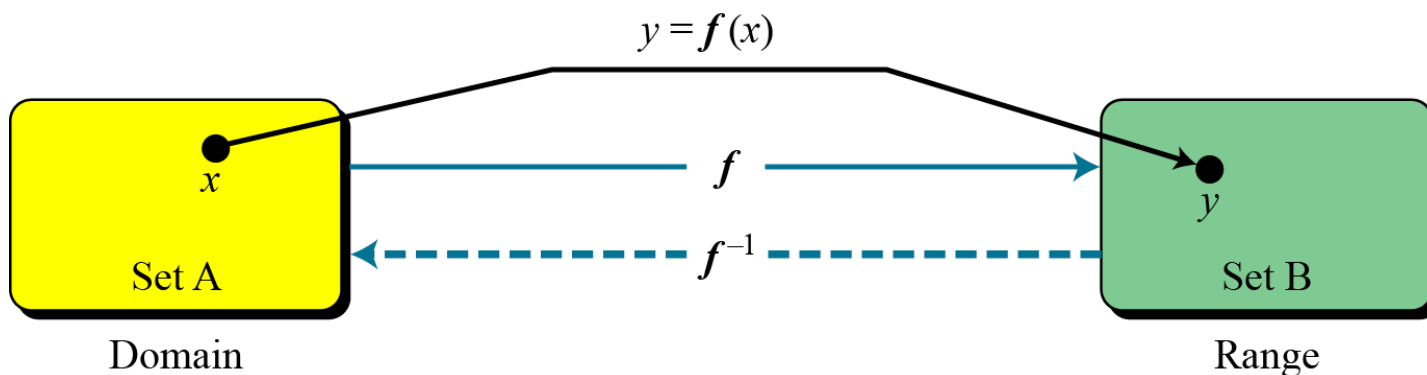
Applications for Public-Key Cryptosystems

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

Applications for Public-Key Cryptosystems

Magic: Trapdoor One-Way Function

- The main idea behind asymmetric-key cryptography is the concept of the trapdoor one-way function
 - *f is easy to compute*
 - *f^{-1} is difficult to compute*
 - *Given y and a trapdoor information, x can be computed easily.*



Lecture 5: Public-key Cryptosystems

- Diffie–Hellman key exchange (D-H)
- Asymmetric (Public-key) Cryptosystem
- RSA Public Key Cryptosystem
- Elliptic curve cryptography (ECC)
- Tool: SSH, PGP / GnuPG

Misconceptions Concerning Public-Key Encryption

- Public-key encryption is more secure from cryptanalysis than symmetric encryption
- Public-key encryption is a general-purpose technique that has made symmetric encryption obsolete
- There is a feeling that key distribution is trivial when using public-key encryption, compared to the cumbersome handshaking involved with key distribution centers for symmetric encryption



RSA (Rivest, Shamir, and Adleman)

- Named after its inventors (Rivest, Shamir, and Adleman)
 - Developed in 1977 at MIT by Ron Rivest, Adi Shamir & Len Adleman
- Based on the presumed difficulty of factoring large integers (the factoring problem)
- The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and $n-1$ for some n
 - A typical size for n is 1024 bits, now recommended to be 2048 bits
 - 1024 bits, or 309 decimal digits
- Original paper: “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”
 - <http://people.csail.mit.edu/rivest/Rsapaper.pdf>

RSA Algorithm

- RSA makes use of an expression with exponentials
- Plaintext is encrypted in blocks with each block having a binary value less than some number n
- Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C

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

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

- Both sender and receiver must know the value of n
- The sender knows the value of e , and only the receiver knows the value of d
- This is a public-key encryption algorithm with a public key of $PU=\{e,n\}$ and a private key of $PR=\{d,n\}$

Video Example

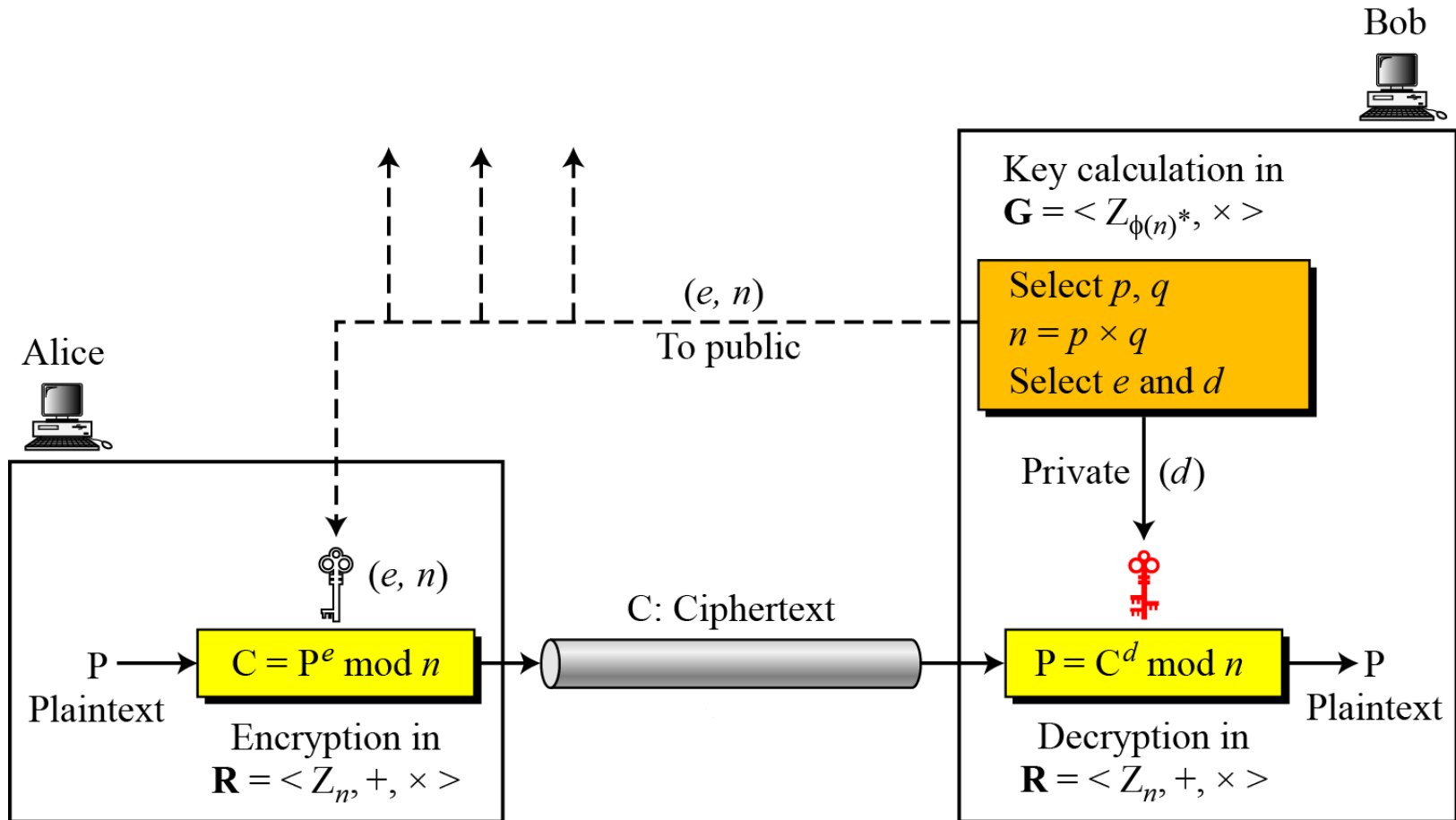
- <https://www.khanacademy.org/computing/computer-science/cryptography/modern-crypt/v/intro-to-rsa-encryption>



Some Mathematics

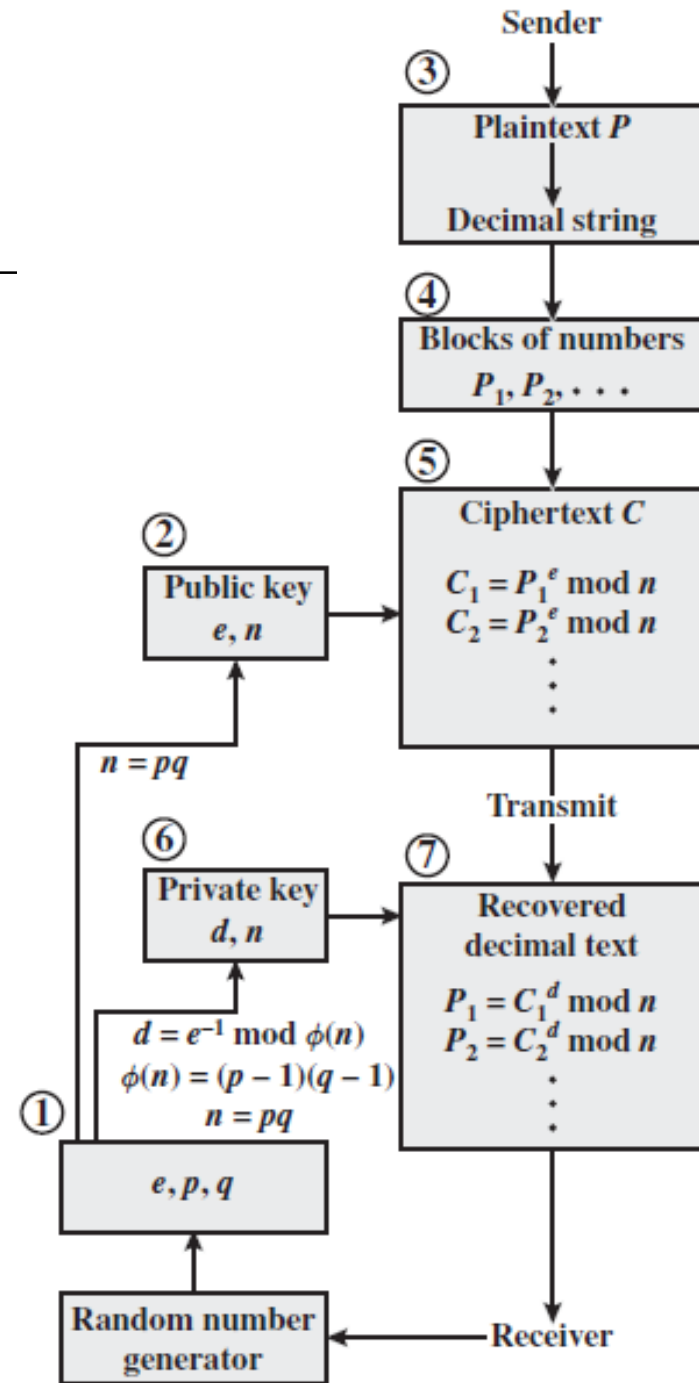
- Euler totient function $\phi(n)$ = the number of integers k in the range $1 \leq k < n$ for which $\gcd(n, k) = 1$
- $\phi(p) = p - 1$ if p is prime
- $\phi(pq) = (p - 1)(q - 1)$ if p and q are primes
- Euler Theorem:
 - if n and M are co-prime positive integers, then $M^{\phi(n)} \equiv 1 \pmod{n}$

General Idea

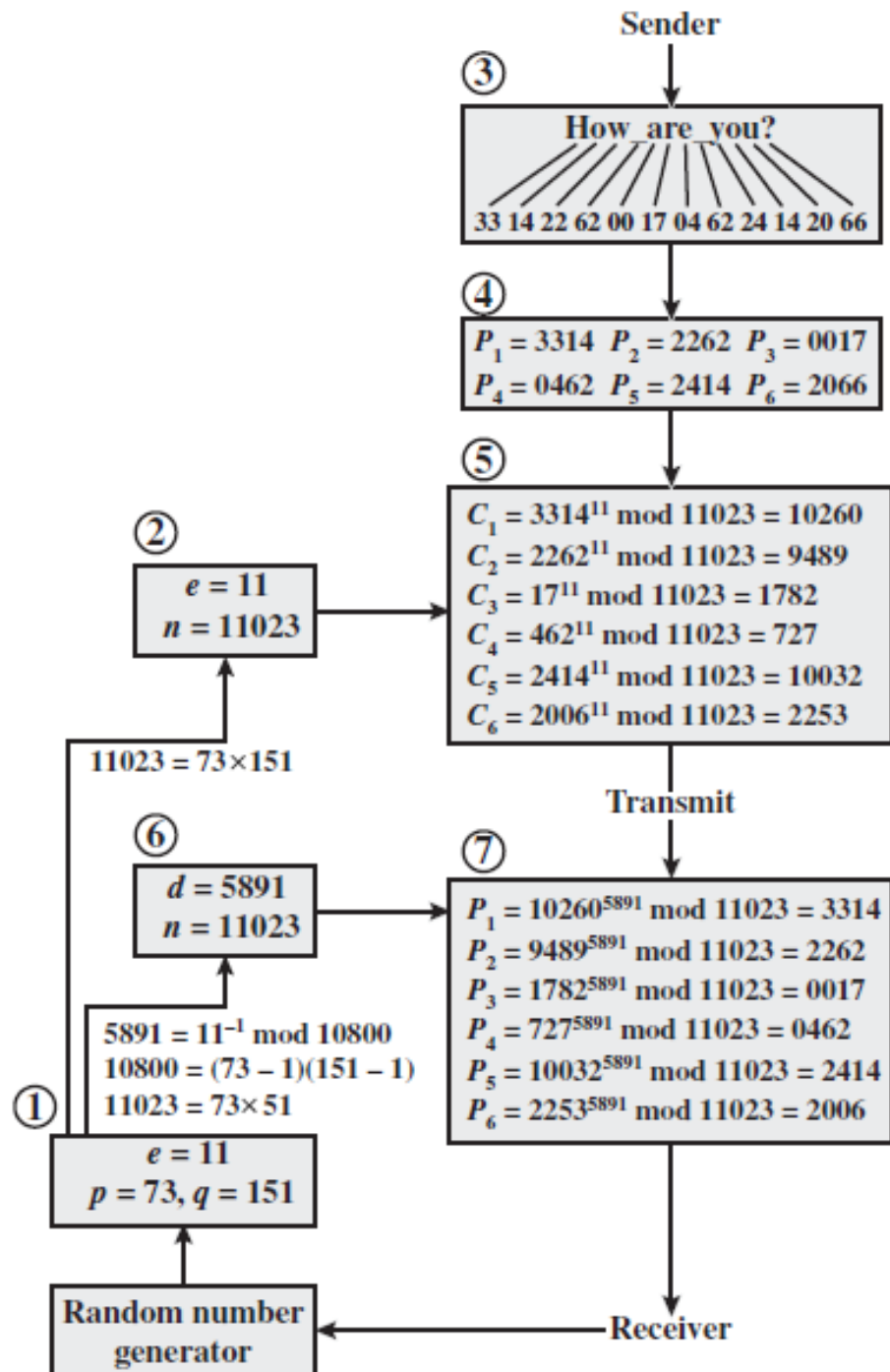


Steps

- Public key: e, n
 - It is infeasible to determine d given e and n
- The **factors** of n (i.e. p and q) must be kept secret
- d must also be kept secret
 - $ed = 1 \bmod (\phi(n))$
 - or $ed = k\phi(n) + 1$



Detailed Example



Key Generation

Key Generation Alice

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

$PR = \{d, n\}$

Message Encryption and Decryption

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: $M = C^d \bmod n$

Numerical Example

- **Key Generation**

1. Select two prime numbers, $p = 17$ and $q = 11$.
2. Calculate $n = pq = 17 \times 11 = 187$.
3. Calculate $\phi(n) = (p - 1)(q - 1) = 16 \times 10 = 160$.
4. Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$; we choose $e = 7$.
5. Determine d such that $de \equiv 1 \pmod{160}$ and $d < 160$. The correct value is $d = 23$, because $23 \times 7 = 161 = (1 \times 160) + 1$; d can be calculated using the extended Euclid's algorithm (Chapter 4).
 - Public key: $(e, n) = (7, 187)$
 - Private key: $d = 23$

Numerical Example

- **Encryption** for the message $M = 88$
- $C = M^e = 88^7 \pmod{187}$

$$88^7 \pmod{187} = [(88^4 \pmod{187}) \times (88^2 \pmod{187}) \times (88^1 \pmod{187})] \pmod{187}$$

$$88^1 \pmod{187} = 88$$

$$88^2 \pmod{187} = 7744 \pmod{187} = 77$$

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

$$88^7 \pmod{187} = (88 \times 77 \times 132) \pmod{187} = 894,432 \pmod{187} = 11$$

Numerical Example

- **Decryption:** calculate $M = C^d = 11^{23} \pmod{187}$

$$11^{23} \pmod{187} = [(11^1 \pmod{187}) \times (11^2 \pmod{187}) \times (11^4 \pmod{187}) \\ \times (11^8 \pmod{187}) \times (11^8 \pmod{187})] \pmod{187}$$

$$11^1 \pmod{187} = 11$$

$$11^2 \pmod{187} = 121$$

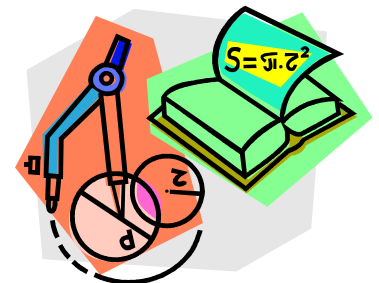
$$11^4 \pmod{187} = 14,641 \pmod{187} = 55$$

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

$$11^{23} \pmod{187} = (11 \times 121 \times 55 \times 33 \times 33) \pmod{187} = 79,720,245 \pmod{187} = 88$$

Algorithm Requirements

- For this algorithm to be satisfactory for public-key encryption, the following requirements must be met:
 1. It is possible to find values of e, d, n such that $M^{ed} \bmod n = M$ for all $M < n$
 2. It is relatively easy to calculate $M^e \bmod n$ and $C^d \bmod n$ for all values of $M < n$
 3. It is infeasible to determine d given e and n



Example of RSA Algorithm

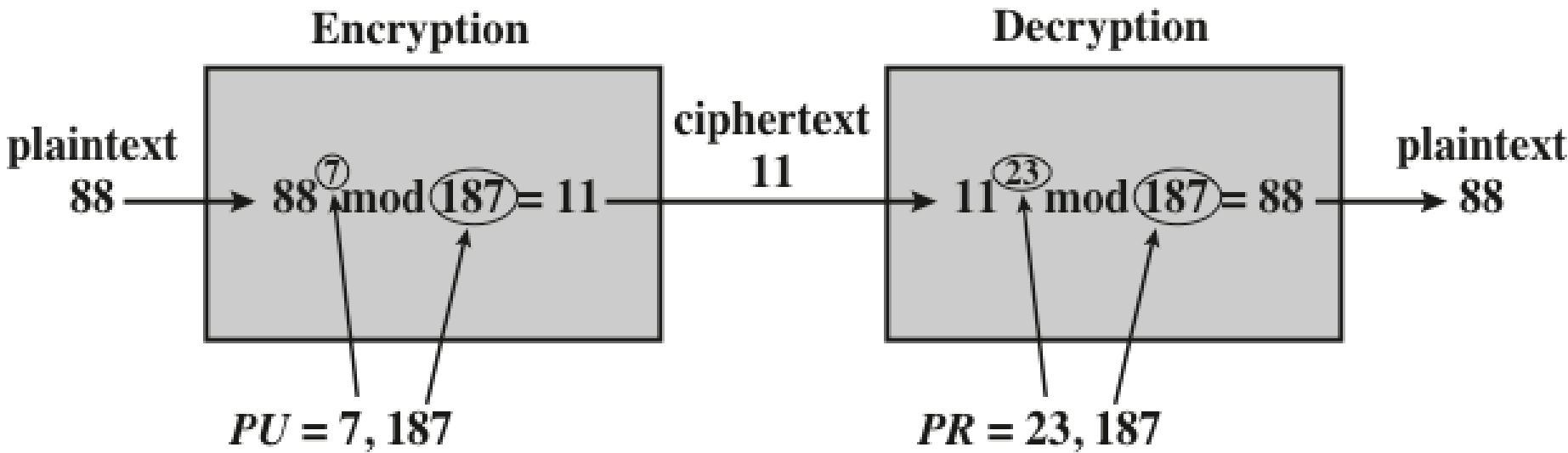


Figure 9.6 Example of RSA Algorithm

Exponentiation in Modular Arithmetic

- Both encryption and decryption in RSA involve raising an integer to an integer power, mod n
- Can make use of a property of modular arithmetic:
$$[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$$
- With RSA you are dealing with potentially large exponents so efficiency of exponentiation is a consideration

```

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

```

Note: The integer b is expressed as a binary number $b_k b_{k-1} \dots b_0$

Figure 9.8 Algorithm for Computing $a^b \bmod n$

Efficient Operation Using the Public Key

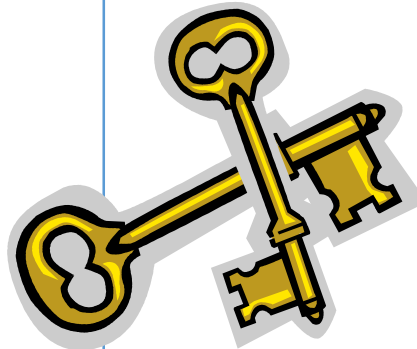
- To speed up the operation of the RSA algorithm using the public key, a specific choice of e is usually made
- The most common choice is 65537 ($2^{16} + 1$)
 - Two other popular choices are $e=3$ and $e=17$
 - Each of these choices has only two 1 bits, so the number of multiplications required to perform exponentiation is minimized
 - With a very small public key, such as $e = 3$, RSA becomes vulnerable to a simple attack

Efficient Operation Using the Private Key

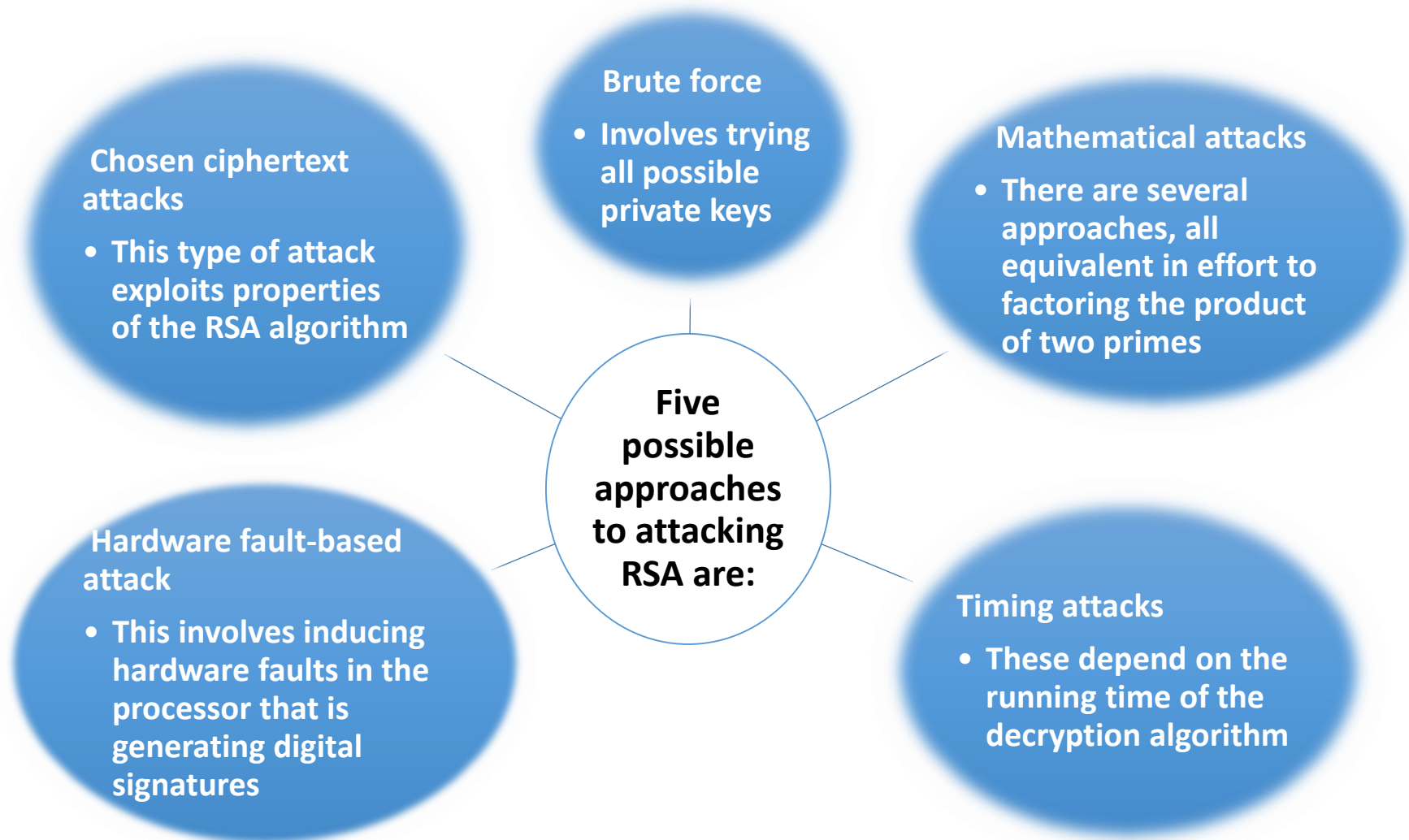
- Decryption uses exponentiation to power d
 - A small value of d is vulnerable to a brute-force attack and to other forms of cryptanalysis
- Can use the Chinese Remainder Theorem (CRT) to speed up computation
 - The quantities $d \bmod (p - 1)$ and $d \bmod (q - 1)$ can be precalculated
 - End result is that the calculation is approximately four times as fast as evaluating $M = C^d \bmod n$ directly

Key Generation

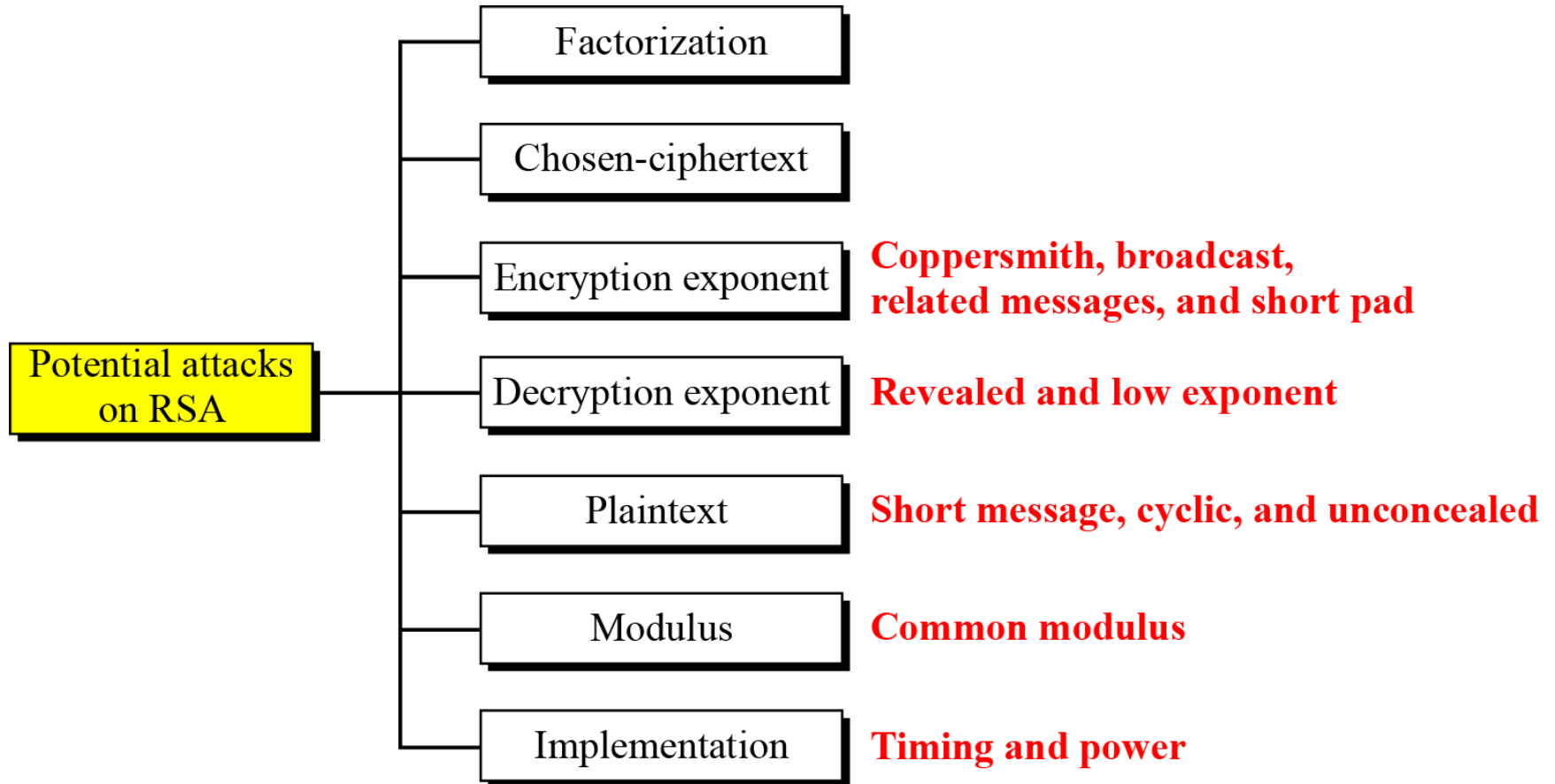
- Before the application of the public-key cryptosystem each participant must generate a pair of keys:
 - Determine two prime numbers p and q
 - Select either e or d and calculate the other
- Because the value of $n = pq$ will be known to any potential adversary, primes must be chosen from a sufficiently large set
 - The method used for finding large primes must be reasonably efficient



The Security of RSA



Potential Attacks on RSA



Factoring Problem

- Attacking RSA mathematically:
 - Factor n into its two prime factors. This enables calculation of $\phi(n) = (p - 1) \times (q - 1)$, which in turn enables determination of $d = e^{-1} \pmod{\phi(n)}$

RSA number	Decimal digits	Binary digits	Cash prize offered	Factored on	Factored by
RSA-100	100	330	US\$1,000 ^[4]	April 1, 1991 ^[5]	Arjen K. Lenstra
RSA-110	110	364	US\$4,429 ^[4]	April 14, 1992 ^[5]	Arjen K. Lenstra and M.S. Manasse
RSA-120	120	397	\$5,898 ^[4]	July 9, 1993 ^[6]	T. Denny et al.
RSA-129 ^[**]	129	426	US\$100	April 26, 1994 ^[5]	Arjen K. Lenstra et al.
RSA-130	130	430	US\$14,527 ^[4]	April 10, 1996	Arjen K. Lenstra et al.
RSA-140	140	463	US\$17,226	February 2, 1999	Herman te Riele et al.
RSA-150	150	496		April 16, 2004	Kazumaro Aoki et al.
RSA-155	155	512	\$9,383 ^[4]	August 22, 1999	Herman te Riele et al.
RSA-160	160	530		April 1, 2003	Jens Franke et al. , University of Bonn
RSA-170 ^[*]	170	563		December 29, 2009	D. Bonenberger and M. Krone ^[***]
RSA-576	174	576	US\$10,000	December 3, 2003	Jens Franke et al. , University of Bonn

Number of Decimal Digits	Number of Bits	Date Achieved
100	332	April 1991
110	365	April 1992
120	398	June 1993
129	428	April 1994
130	431	April 1996
140	465	February 1999
155	512	August 1999
160	530	April 2003
174	576	December 2003
200	663	May 2005
193	640	November 2005
232	768	December 2009

Progress in RSA Factorization

MIPS-Years Needed to Factor

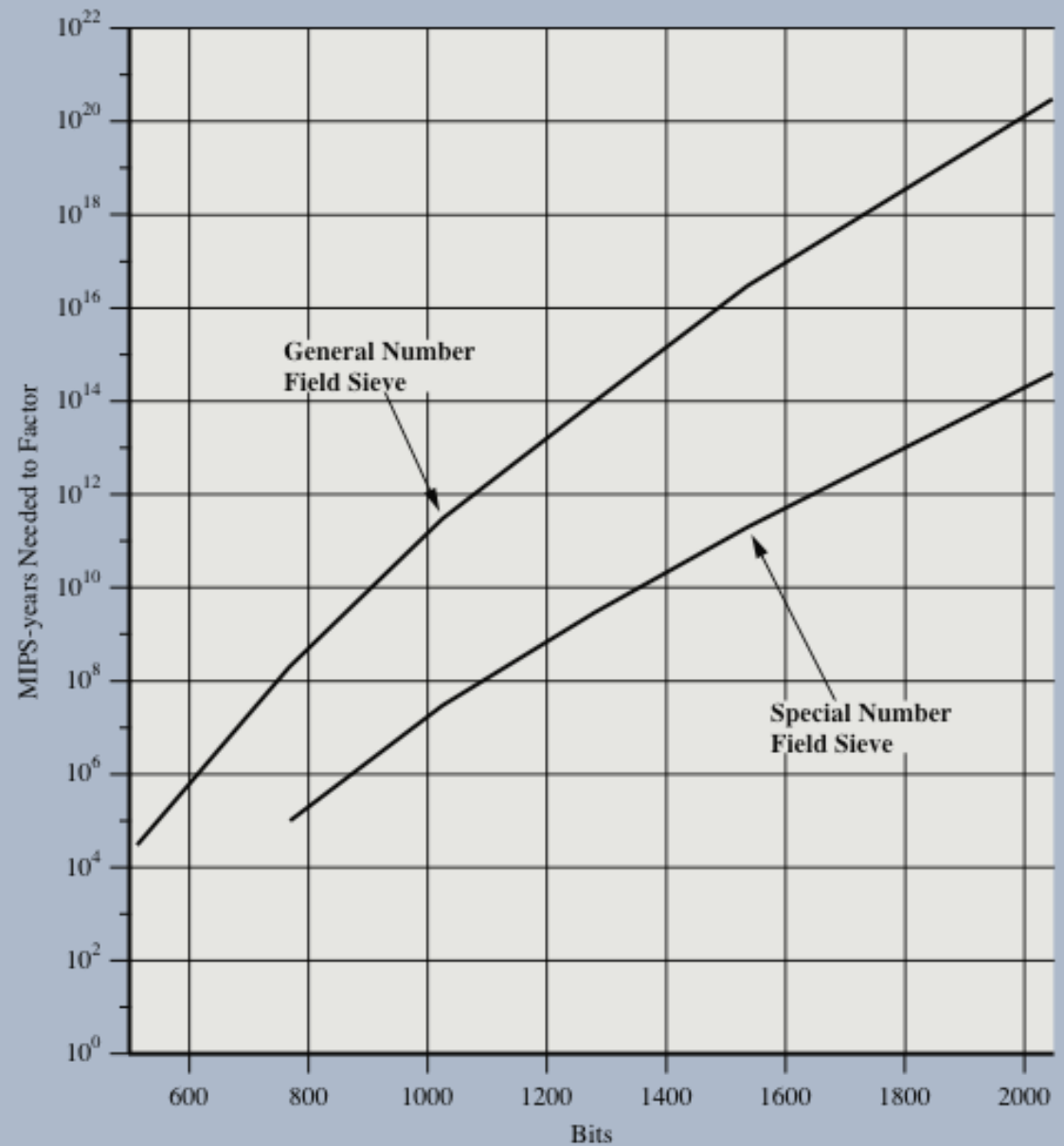


Figure 9.9 MIPS-years Needed to Factor

Timing Attacks

- Paul Kocher, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages
- Are applicable not just to RSA but to other public-key cryptography systems
- Are alarming for two reasons:
 - It comes from a completely unexpected direction
 - It is a ciphertext-only attack



Fault-Based Attack

- An attack on a processor that is generating RSA digital signatures
 - Induces faults in the signature computation by reducing the power to the processor
 - The faults cause the software to produce invalid signatures which can then be analyzed by the attacker to recover the private key
- The attack algorithm involves inducing single-bit errors and observing the results
- While worthy of consideration, this attack does not appear to be a serious threat to RSA
 - It requires that the attacker have physical access to the target machine and is able to directly control the input power to the processor

Summary

- Public-key cryptosystems
- Applications for public-key cryptosystems
- Requirements for public-key cryptography
- Public-key cryptanalysis



- The RSA algorithm
 - Description of the algorithm
 - Computational aspects
 - Security of RSA

Lecture 5: Public-key Cryptosystems

- Diffie–Hellman key exchange (D-H)
- Asymmetric (Public-key) Cryptosystem
- RSA Public Key Cryptosystem
- Elliptic curve cryptography (ECC)
- Tool: SSH, PGP / GnuPG

Problems with RSA

- Key is very long – now recommended key size is 2048 bits (c.f. AES key size = 128 bits to 256 bits)
- Need to look for alternatives that give the same level of security with smaller key sizes (for resource limited environment)
 - In 1985, Neal Koblitz and Victor Miller independently proposed using elliptic curves for public key cryptosystems
 - In the late 1990's, ECC was standardized and it started receiving commercial acceptance (e.g. Certicom <http://www.certicom.com/>)

Elliptic Curves

- 3 types of elliptic curves
 - Elliptic curve over R (real numbers)
 - Slow, inaccurate due to rounding error
 - Not considered here
 - Prime field: Elliptic curve over F_p where p is prime
 - prime curves are best for software applications
 - Binary field: Elliptic curve over $F(2^m)$ where $m > 0$
 - binary curves are best for hardware applications
- Need to define and perform two operations:
 - addition and scalar multiplication

Notation

- Points on elliptic curve: P, Q
- Scalar multiplication
 - $kP = P + P + \dots + P$ (for k times, involving $k - 1$ additions)
 - Double-and-Add algorithm: reducing the number of additions
 - e.g. calculate $17P = 2(2(2(2P))) + P$
 - Only addition: $(17 - 1 = 16)$ Point additions
 - Double-and-add: 4 Point doublings + 1 Point addition

Elliptic Curve Discrete Logarithm Problem (ECDLP)

- Elliptic curve cryptosystems depend on the difficulty in solving the ECDLP
 - Given an elliptic curve E defined over a finite field. For points P, Q on the elliptic curve E , find the integer $d \in [0, n - 1]$ such that $Q = dP$

Key Generation

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 $P_B = n_B \times G$

ECC Diffie-Hellman Key Exchange

Calculation of Secret Key by User A

$$K = n_A \times P_B$$

Calculation of Secret Key by User B

$$K = n_B \times P_A$$

- $Key = K = n_A \times (P_B) = n_A \times (n_B \times G) = n_B \times P_A$

ECC Encryption

- All knows G
- User A (public key: P_A ; private key: n_A)
- User B (public key: P_B ; private key: n_B)
- A wants to send a message M to B
 - A chooses a random integer $k > 0$
 - A calculates the ciphertext $C = \{kG, M + kP_B\}$ and sends to B
- B decrypts for A as follow:
 - B multiplies the first point in the pair by B's secret key and subtracts the result from the second point
 - $(M + kP_B) - (n_B \times kG) = M + kP_B - k \times n_B G = M$

ECC Tutorial

- <https://www.certicom.com/content/certicom/en/20-elliptic-curve-groups-over-real-numbers.html>

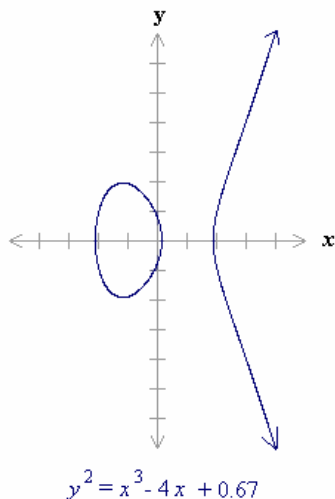
An elliptic curve over real numbers may be defined as the set of points (x,y) which satisfy an elliptic curve equation of the form:

$$y^2 = x^3 + ax + b, \text{ where } x, y, a \text{ and } b \text{ are real numbers.}$$

Each choice of the numbers a and b yields a different elliptic curve. For example, $a = -4$ and $b = 0.67$ gives the elliptic curve with equation $y^2 = x^3 - 4x + 0.67$; the graph of this curve is shown below:

If $x^3 + ax + b$ contains no repeated factors, or equivalently if $4a^3 + 27b^2$ is not 0, then the elliptic curve $y^2 = x^3 + ax + b$

can be used to form a group. An elliptic curve group over real numbers consists of the points on the corresponding elliptic curve, together with a special point O called the point at infinity.



Numerical Example

- An elliptic curve over F_p has the general form:

$$E_p(a, b): y^2 = x^3 + ax + b$$

- Take $p = 751$
- $E_p(-1, 188): y^2 = x^3 - x + 188$
- Take $G = (0, 376)$
- B's public key is $P_B = (201, 5)$
- A wants to send the message $M = (562, 201)$ to B
- A choose $k = 386$

Numerical Example

- A needs to calculate C
 - $kG = 386 \times (0, 376)$
 - $M + kP_B = (562, 201) + 386 \times (201, 5)$
- Point addition for elliptic curve over F_p :
 - For $R = P + Q$

$$x_R = (\lambda^2 - x_P - x_Q) \bmod p$$

$$y_R = (\lambda(x_P - x_R) - y_P) \bmod p$$

$$\text{where } \lambda = \frac{y_Q - y_P}{x_Q - x_P} \bmod p$$

Numerical Example

- Point doubling for elliptic curve over F_p :
 - $R = 2P$

$$x_R = (\lambda^2 - 2x_P) \bmod p$$

$$y_R = (\lambda(x_P - x_R) - y_P) \bmod p$$

$$\text{where } \lambda = \frac{3x_P^2 + a}{2y_P} \bmod p$$

Numerical Example

- Use sage to assist our computation
 - <https://cloud.sagemath.com/>
- Can use the following to compute the ciphertext {C1, C2}
 - *$F = \text{FiniteField}(751)$*
 - *$E = \text{EllipticCurve}(F, [-1, 188])$*
 - *$\text{print}(E)$*
 - *$G = E(0, 376)$*
 - *$C1 = 386 * G$*
 - *$M = E(562, 201)$*
 - *$C2 = M + 386 * E(201, 5)$*
 - *$C1 ; C2$*

Numerical Example

```
~$ sage
```

```
Sage Version 5.11, Release Date: 2013-08-13  
Enhanced for the Sagemath Cloud  
Type "help()" for help.
```

```
sage: F = FiniteField(751)  
sage: E = EllipticCurve(F, [ -1, 188 ])  
sage: print(E)  
Elliptic Curve defined by  $y^2 = x^3 + 750x + 188$  over Finite Field of size 751  
sage: G = E(0,376)  
sage: C1 = 386*G  
sage: M = E(562,201)  
sage: C2 = M + 386 * E(201,5)  
sage: C1 ; C2  
(676 : 558 : 1)  
(385 : 328 : 1)  
sage: 
```

- A sends $\{(676, 558), (385, 328)\}$ to B

How to Decrypt?

- Need private key of B: n_B
 - Brute force search again B's public key $P_B = (201, 5)$
 - $PB=E(201,5)$
 - *for i in range(751):*
if $i * G == PB$:print(i)
 - *print (58*G)*
 - *print (C2 - 58*C1)*

Elliptic Curve Arithmetic

- Most of the products and standards that use public-key cryptography for encryption and digital signatures use RSA
 - The key length for secure RSA use has increased over recent years and this has put a heavier processing load on applications using RSA
- Elliptic curve cryptography (ECC) is showing up in standardization efforts including the IEEE P1363 Standard for Public-Key Cryptography
- Principal attraction of ECC is that it appears to offer equal security for a far smaller key size
- Confidence level in ECC is not yet as high as that in RSA

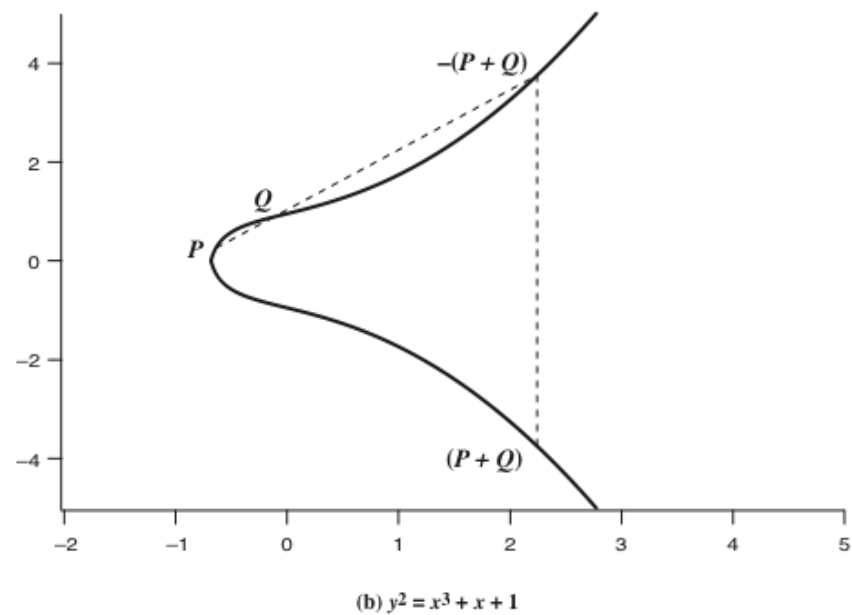
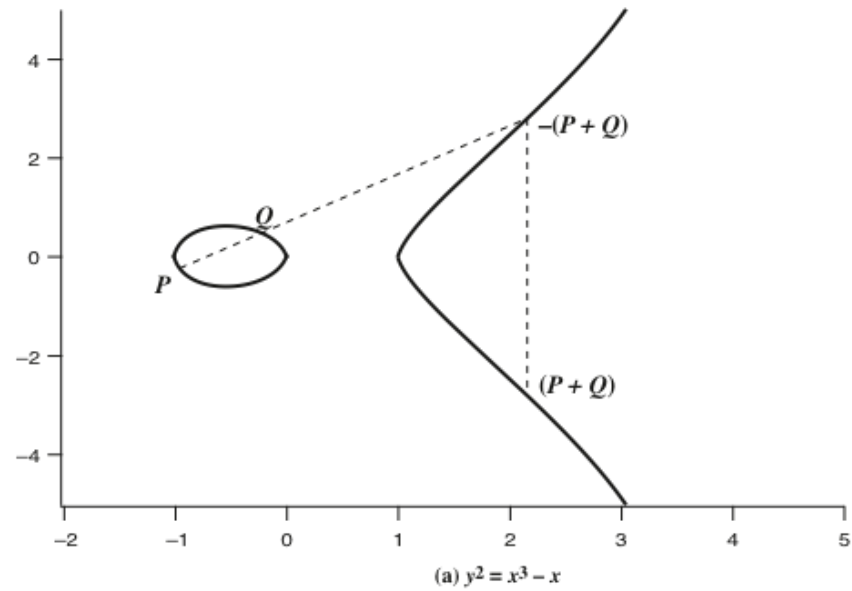
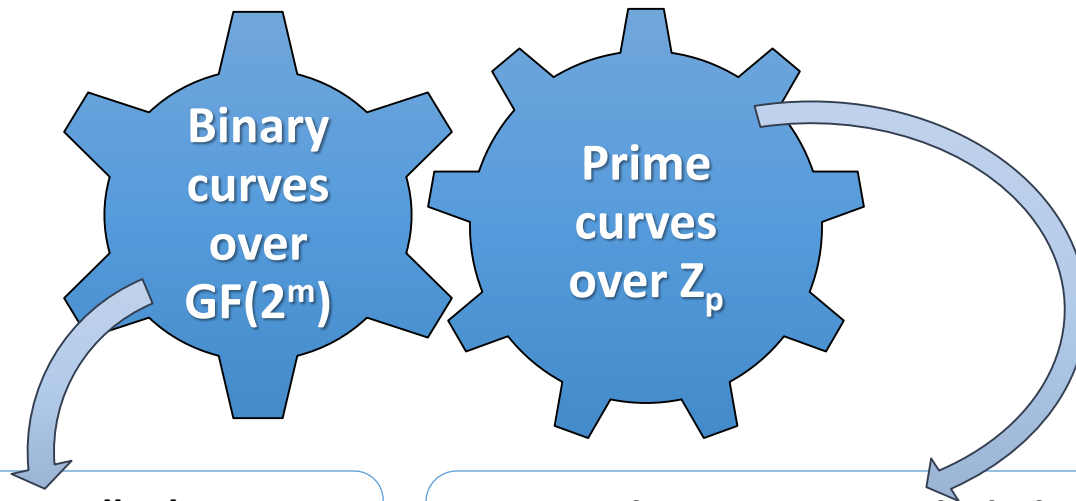


Figure 10.4 Example of Elliptic Curves

Elliptic Curves Over $\mathbb{Z}_p$

- Elliptic curve cryptography uses curves whose variables and coefficients are finite
- Two families of elliptic curves are used in cryptographic applications:

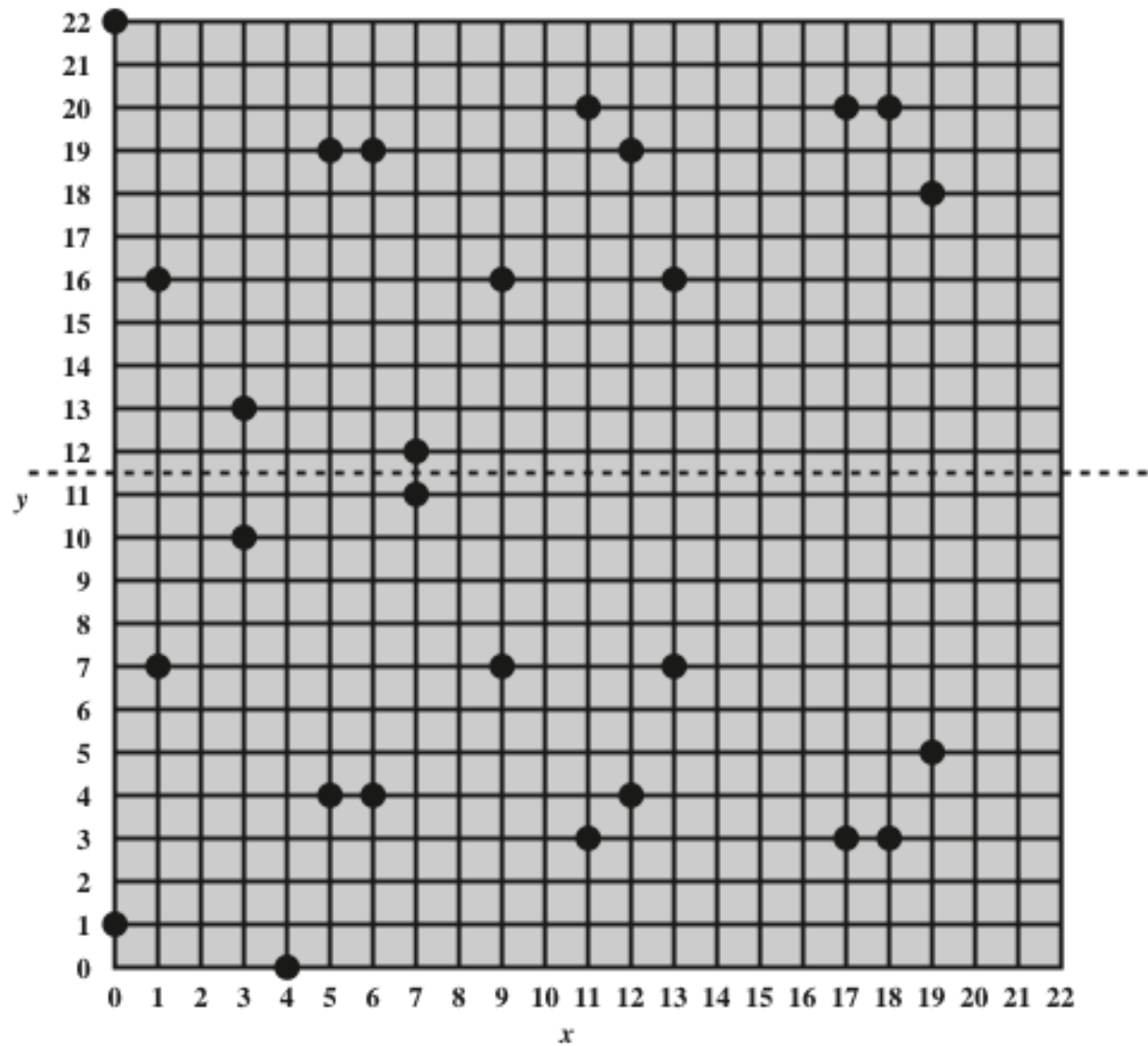


- Variables and coefficients all take on values in $\text{GF}(2^m)$ and in calculations are performed over $\text{GF}(2^m)$
- Best for hardware applications

- Use a cubic equation in which the variables and coefficients all take on values in the set of integers from 0 through $p-1$ and in which calculations are performed modulo p
- Best for software applications

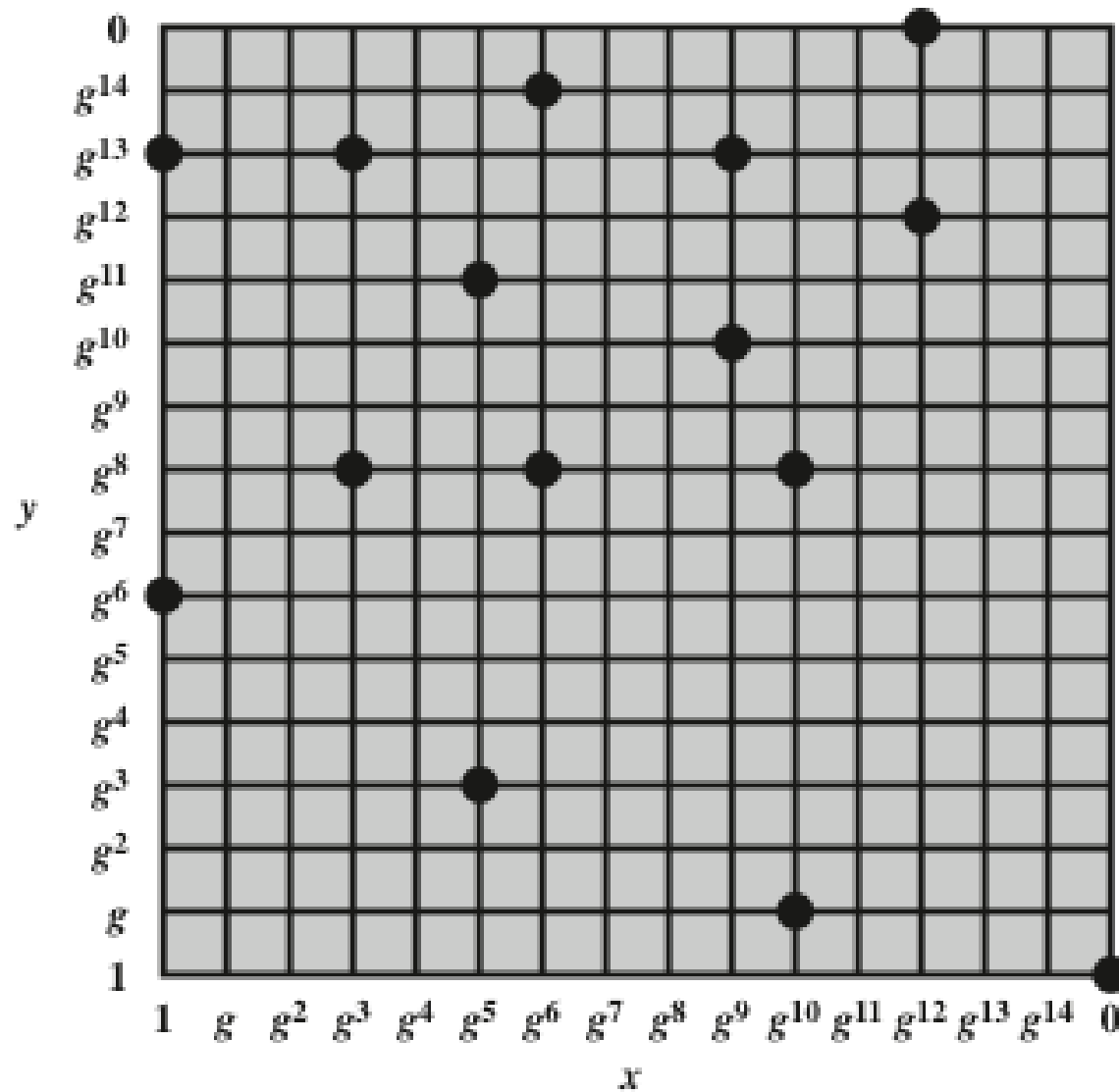
Points (other than O) on the Elliptic Curve $E_{23}(1, 1)$

(0, 1)	(6, 4)	(12, 19)
(0, 22)	(6, 19)	(13, 7)
(1, 7)	(7, 11)	(13, 16)
(1, 16)	(7, 12)	(17, 3)
(3, 10)	(9, 7)	(17, 20)
(3, 13)	(9, 16)	(18, 3)
(4, 0)	(11, 3)	(18, 20)
(5, 4)	(11, 20)	(19, 5)
(5, 19)	(12, 4)	(19, 18)



Elliptic Curves Over Prime field $\mathbb{Z}_p$

Figure 10.5 The Elliptic Curve $E_{23}(1,1)$



Elliptic Curves Over $GF(2^m)$

Figure 10.6 The Elliptic Curve $E_{2^4}(g^4, 1)$

Elliptic Curve Cryptography (ECC)

- Addition operation in ECC is the counterpart of modular multiplication in RSA
- Multiple addition is the counterpart of modular exponentiation

To form a cryptographic system using elliptic curves, we need to find a “hard problem” corresponding to factoring the product of two primes or taking the discrete logarithm

- $Q=kP$, where Q, P belong to a prime curve
- Is “easy” to compute Q given k and P
- But “hard” to find k given Q , and P
- Known as the elliptic curve logarithm problem

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	
$K = n_B \times P_A$	

Figure 10.7 ECC Diffie-Hellman Key Exchange

ECC Encryption/Decryption

- Several approaches using elliptic curves have been analyzed
- Must first encode any message m as a point on the elliptic curve P_m
- Select suitable curve and point G as in Diffie-Hellman
- Each user chooses a private key n_A and generates a public key $P_A = n_A * G$
- To encrypt and send message P_m to B, A chooses a random positive integer k and produces the ciphertext C_m *consisting of the pair of points*:

$$C_m = \{kG, P_m + kP_B\}$$

- To decrypt the ciphertext, B multiplies the first point in the pair by B's secret key and subtracts the result from the second point:

$$P_m + kP_B - n_B(kG) = P_m + k(n_B G) - n_B(kG) = P_m$$

Security of Elliptic Curve Cryptography

- Depends on the difficulty of the elliptic curve logarithm problem
- Fastest known technique is “Pollard rho method”
- Compared to factoring, can use much smaller key sizes than with RSA
- For equivalent key lengths computations are roughly equivalent
- Hence, for similar security ECC offers significant computational advantages

Comparable Key Sizes in Terms of Computational Effort for Cryptanalysis (NIST SP-800-57)

Symmetric key algorithms	Diffie-Hellman, Digital Signature Algorithm	RSA (size of n in bits)	ECC (modulus size in bits)
80	$L = 1024$ $N = 160$	1024	160–223
112	$L = 2048$ $N = 224$	2048	224–255
128	$L = 3072$ $N = 256$	3072	256–383
192	$L = 7680$ $N = 384$	7680	384–511
256	$L = 15,360$ $N = 512$	15,360	512+

Note: L = size of public key, N = size of private key

Lecture 5: Public-key Cryptosystems

- Diffie–Hellman key exchange (D-H)
- Asymmetric (Public-key) Cryptosystem
- RSA Public Key Cryptosystem
- Elliptic curve cryptography (ECC)
- Tool: SSH, PGP / GnuPG

Tools

- There are many tools making use of asymmetric cryptosystems
 - ssh / openssh allows you to perform automatic login from host A to Host B using RSA authentication
 - `ssh-keygen -t rsa`
 - For more details, refer to http://www.linuxproblem.org/art_9.html

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/wwfung/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/wwfung/.ssh/id_rsa.
Your public key has been saved in /home/wwfung/.ssh/id_rsa.pub.
The key fingerprint is:
61:86:7d:65:e8:ca:10:35:e9:7c:21:a2:17:d4:61:c9 wwfung@wwfung-PC
The key's randomart image is:
+--[ RSA 2048 ]-----+
  |      . . + = + . 0  |
  |      ++E.oo       |
  |      ..*=o..      |
  |      . ooooo      |
  |      . oSo        |
  |      o            |
+-----+-----+
```

Tools

- PGP / GnuPG relies on a special approach for its data encryption that combines the features of both secret-key and public-key cryptosystem
- Suppose that Alice wishes to send Bob a confidential email using the PGP
 - PGP first compresses Alice's plaintext document
 - After the document is compressed, the PGP software generates a session-specific key for it
 - PGP then uses this session key to encrypt the document
- Public-key encryption must now be used to securely deliver the session key to Bob

GnuPG

- <https://www.gnupg.org/>



[Home](#) [Donate](#) [Software](#) [Download](#) [Documentation](#) [Blog](#)

THE GNU PRIVACY GUARD

GnuPG is a complete and free implementation of the OpenPGP standard as defined by [RFC4880](#) (also known as *PGP*). GnuPG allows to encrypt and sign your data and communication, features a versatile key management system as well as access modules for all kinds of public key directories. GnuPG, also known as *GPG*, is a command line tool with features for easy integration with other applications. A wealth of [frontend applications](#) and [libraries](#) are available. GnuPG also provides support for S/MIME and Secure Shell (ssh).

GnuPG is [Free Software](#) (meaning that it respects your freedom). It can be freely used, modified and distributed under the terms of the [GNU General Public License](#).

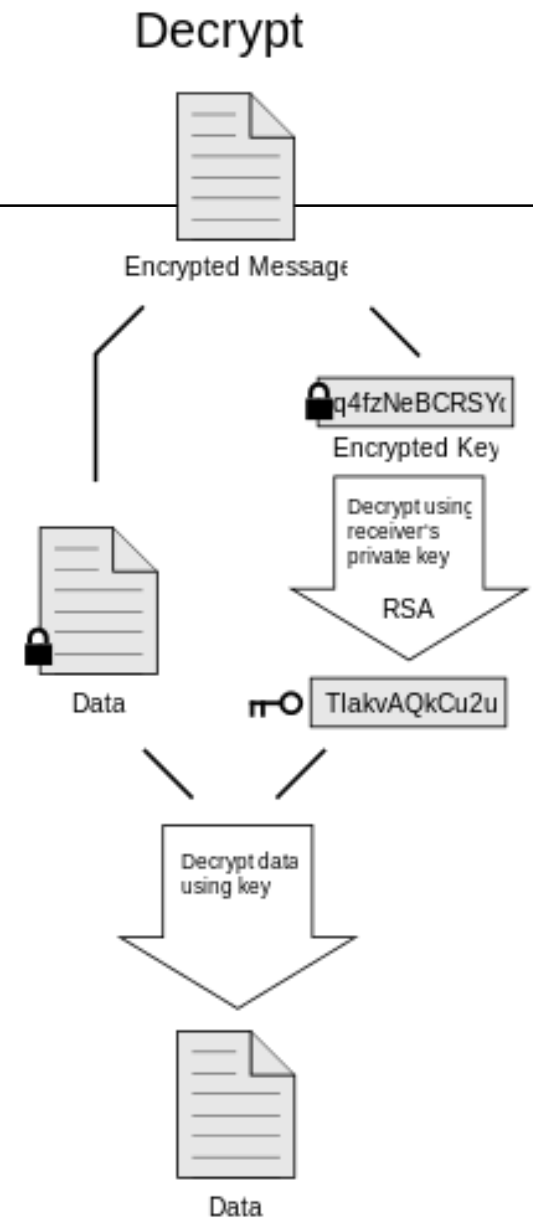
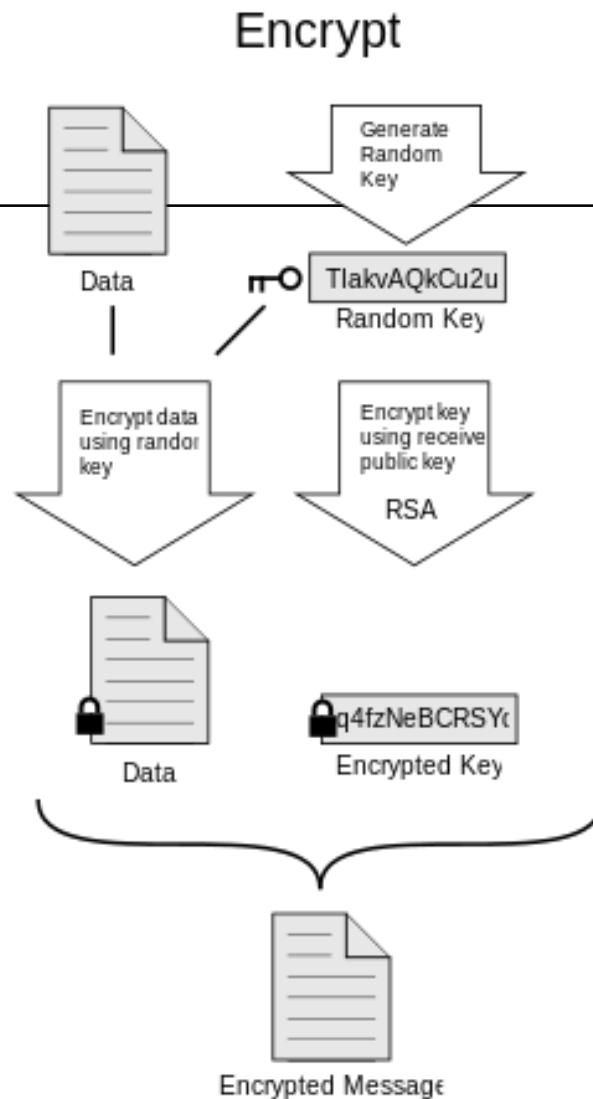
The current version of GnuPG is 2.2.1. See the [download](#) page for other maintained versions.

[Gpg4win](#) is a Windows version of GnuPG featuring a context menu tool, a crypto manager, and an Outlook plugin to send and receive standard PGP/MIME mails. The current version of Gpg4win is 3.0.0.

Tools

- PGP encrypts the randomly-generated session key with Bob's public key
 - then appends the RSA-encrypted session key to the beginning of Alice's session-key-encrypted document
- The document and session key are then sent together to Bob.
- To decrypt Alice's document
 - Bob first uses his private key to decrypt the session key
 - Then uses the session key to decrypt the document itself

Hybrid Approach



Lecture 5: What you have learnt?

- Diffie–Hellman key exchange (D-H)
- Asymmetric (Public-key) Cryptosystem
- RSA Public Key Cryptosystem
- Elliptic curve cryptography (ECC)
- Tool: SSH, PGP / GnuPG

Your mission



- Understand DH and public key crypto
- Master the tools for public-key crypto

