# Public Key Encryption

Göran Andersson

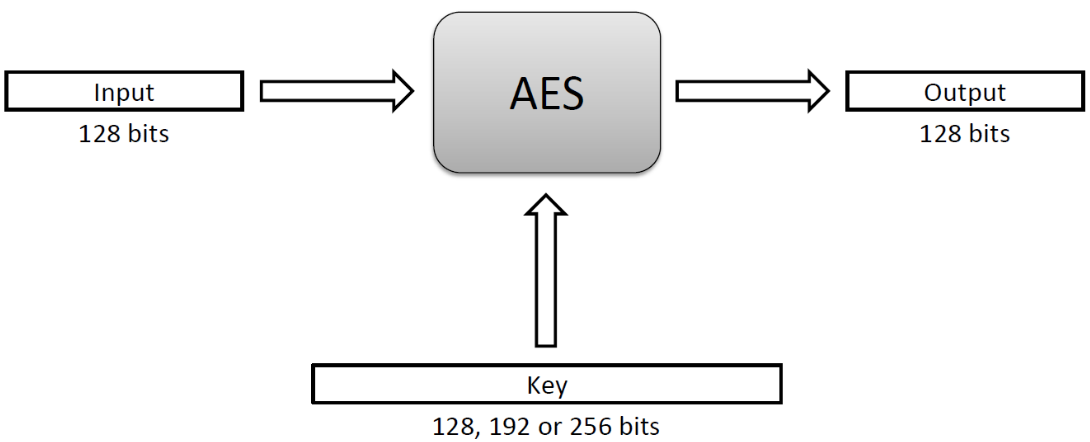
goeran@kth.se

# Symmetric Key Encryption

Cont'd

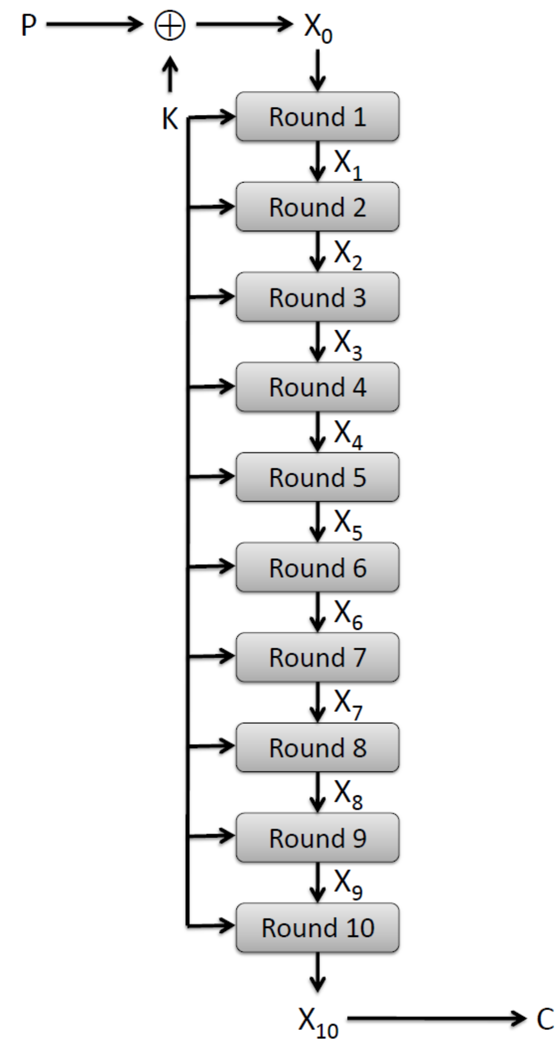
# The Advanced Encryption Standard (AES)

- In 1997, the U.S. National Institute for Standards and Technology (NIST) put out a public call for a replacement to DES.
- It narrowed down the list of submissions to five finalists, and ultimately chose an algorithm that is now known as the **Advanced Encryption Standard** (**AES**).
- AES is a block cipher that operates on 128-bit blocks. It is designed to be used with keys that are 128, 192, or 256 bits long, yielding ciphers known as AES-128, AES-192, and AES-256.



# AES Round Structure

- The 128-bit version of the AES encryption algorithm proceeds in ten rounds (10, 12 or 14).
- Each round performs an invertible transformation on a 128-bit array, called **state**.
- The initial state $X_0$ is the XOR of the plaintext P with the key K:
- $X_0$ = P XOR K.
- Round i (i = 1, …, 10) receives state $X_{i-1}$ as input and produces state $X_i$.
- The ciphertext C is the output of the final round: C = $X_{10}$.



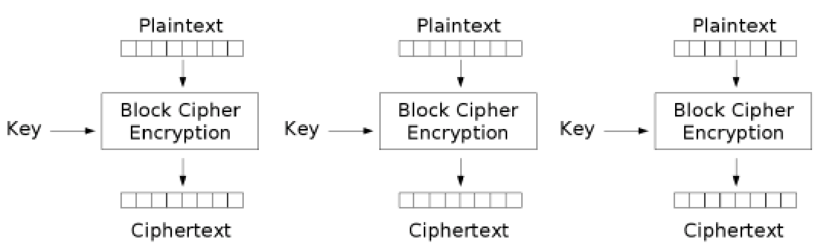
# AES Rounds

- Each round is built from four basic steps:

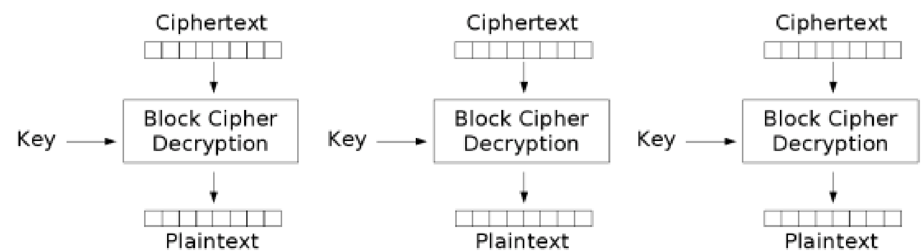
1. **SubBytes step**: an S-box substitution step
2. **ShiftRows step**: a permutation step
3. **MixColumns step**: a matrix multiplication step
4. **AddRoundKey step**: an XOR step with a **round key** derived from the 128-bit encryption key

# Block Cipher Modes

- A block cipher mode describes the way a block cipher encrypts and decrypts a sequence of message blocks.
- Electronic Code Book (ECB) Mode (is the simplest):
  - Block P[i] encrypted into ciphertext block C[i] = $E_K$(P[i])
  - Block C[i] decrypted into plaintext block M[i] = $D_K$(C[i])




Electronic Codebook (ECB) mode encryption




Electronic Codebook (ECB) mode decryption

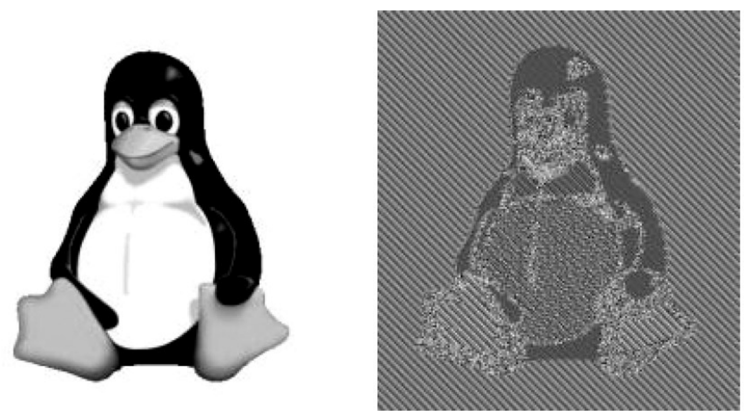


Public domain images from http://en.wikipedia.org/wiki/File:Ecb_encryption.png and http://en.wikipedia.org/wiki/File:Ecb_decryption.png

# Strengths and Weaknesses of ECB

- Strengths:
  - Is very simple
  - Allows for parallel encryptions of the blocks of a plaintext
  - Can tolerate the loss or damage of a block

- Weakness:
  - Documents and images are not suitable for ECB encryption since patters in the plaintext are repeated in the ciphertext:

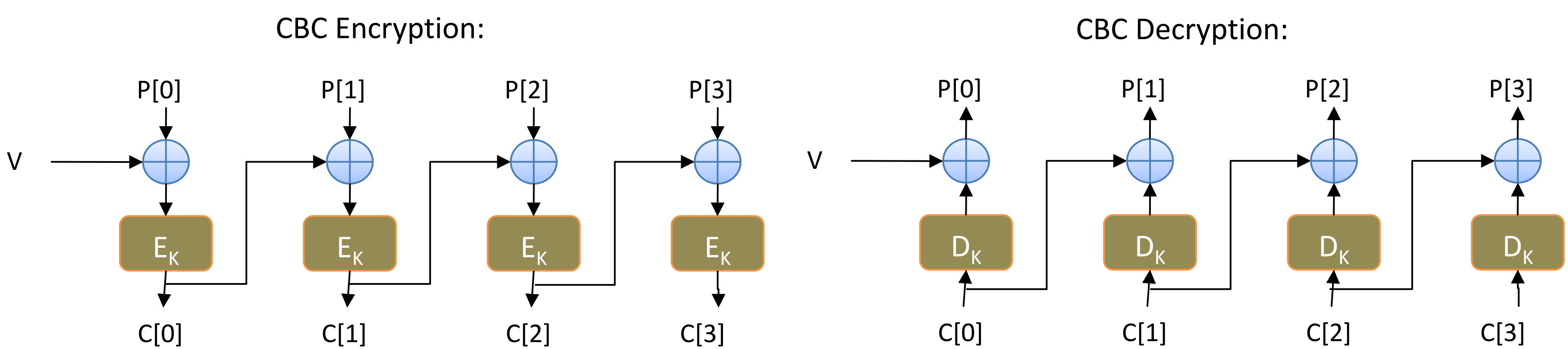


(a) (b)

**Figure 8.6:** How ECB mode can leave identifiable patterns in a sequence of blocks: (a) An image of Tux the penguin, the Linux mascot. (b) An encryption of the Tux image using ECB mode. (The image in (a) is by Larry Ewing, lewing@isc.tamu.edu, using The Gimp; the image in (b) is by Dr. Juzam. Both are used with permission via attribution.)

# Cipher Block Chaining (CBC) Mode

- In Cipher Block Chaining (CBC) Mode
  - The previous ciphertext block is combined with the current plaintext block $C[i] = E_K(C[i-1] \oplus P[i])$
  - $C[-1] = V$, a random block separately transmitted encrypted (known as the initialization vector)
  - Decryption: $P[i] = C[i-1] \oplus D_K(C[i])$



# Strengths and Weaknesses of CBC

- Strengths:
  - Doesn’t show patterns in the plaintext
  - Is the most common mode
  - Is fast and relatively simple
- Weaknesses:
  - CBC requires the reliable transmission of all the blocks sequentially
  - CBC is not suitable for applications that allow packet losses (e.g., music and video streaming)

# Java AES Encryption Example

- Source

  http://docs.oracle.com/javase/8/docs/api/javax/crypto/package-summary.html

Generate an AES key

```
KeyGenerator keygen = KeyGenerator.getInstance("AES");
SecretKey aesKey = keygen.generateKey();
```

- Create a cipher object for AES in ECB mode and PKCS5 padding

```
Cipher aesCipher;
aesCipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
```

- Encrypt

```
aesCipher.init(Cipher.ENCRYPT_MODE, aesKey);
byte[] plaintext = "My secret message".getBytes();
byte[] ciphertext = aesCipher.doFinal(plaintext);
```

- Decrypt

```
aesCipher.init(Cipher.DECRYPT_MODE, aesKey);
byte[] plaintext1 = aesCipher.doFinal(ciphertext);
```

# Mathematica AES Example

```
In[1]:= msg = "This is a secret that cannot be revealed!";

In[2]:= keyAES = GenerateSymmetricKey[
          Method → <|"Cipher" → "AES256", "BlockMode" → "CBC"|>]

Out[2]= SymmetricKey[ cipher: AES256
                      block mode: CBC
                      key length: 256 bits ]

In[3]:= cipherAES = Encrypt[keyAES, msg]

Out[3]= EncryptedObject[ data length: 48 bytes
                         IV length: 128 bits
                         original form: String ]

In[4]:= Decrypt[keyAES, cipherAES]

Out[4]= This is a secret that cannot be revealed!
```

# Mathematica AES Example

The information can be extracted by

```
In[5]:= Normal[keyAES]

Out[5]= SymmetricKey[<|Cipher → AES256, BlockMode → CBC,
    Key → ByteArray[ 32 bytes ], InitializationVector → None|>]

In[6]:= keyBytesAES = Normal[keyAES["Key"]]

Out[6]= {232, 52, 238, 192, 18, 48, 133, 184, 61, 168,
  24, 182, 179, 182, 92, 29, 56, 52, 100, 192, 168,
  241, 3, 142, 35, 129, 185, 162, 31, 38, 100, 139}
```

The cipher can be transformed into bytes by

```
In[7]:= cipherBytesAES = Normal[cipherAES["Data"]]

Out[7]= {213, 71, 155, 182, 109, 226, 117, 251, 120, 170, 154, 68,
  185, 221, 185, 69, 230, 59, 194, 44, 222, 88, 73, 201,
  163, 32, 53, 146, 3, 135, 94, 114, 217, 147, 225, 85,
  130, 192, 52, 100, 95, 150, 129, 91, 239, 38, 116, 242}
```

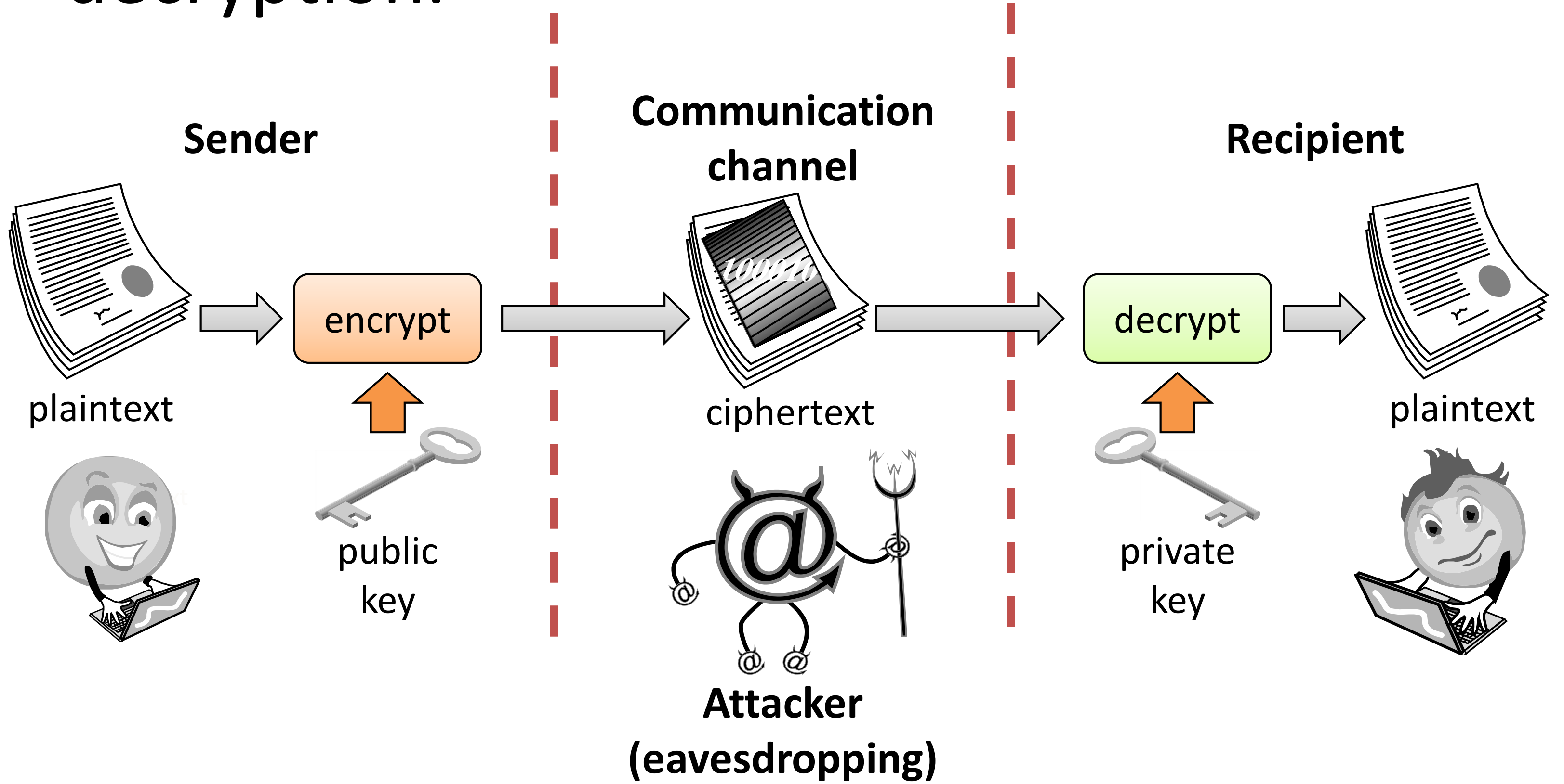
# Public Key Encryption

# Public-Key Cryptography

- Bob has two keys: a **private key,** $S_B$, which Bob keeps **s**ecret, and a **public key,** $P_B$, which Bob broadcasts widely.
- Alice encrypts using Bob's public key, $P_B$,
- $C = E_{P_B}(M)$
- Bob then uses his private key to decrypt the message
- $M = D_{S_B}(C)$.

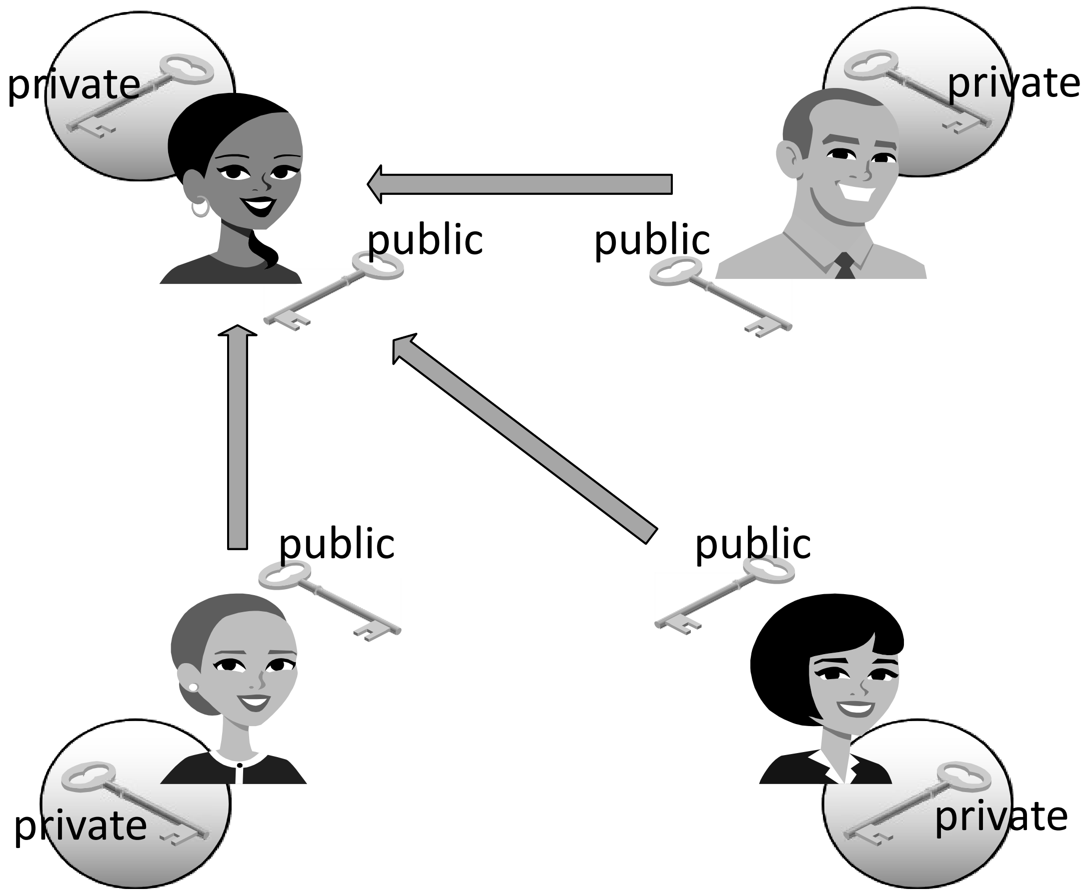
# Public-Key Cryptography

- Separate keys are used for encryption and decryption.



# Public Key Distribution

- Only one key pair is needed for each participant




$$2n \quad \text{keys}$$

$$n > 5 \Rightarrow 2n < \binom{n}{2}$$

# Math Concepts (Review)

## Fundamental theorem of arithmetic

- Every integer $n > 1$ has a unique decomposition of prime factors

$$n = \prod_{i=1}^{r} p_i^{e_i}$$

$$\text{Example}: 13\,276\,725 = 3 \times 5^2 \times 7 \times 11^3 \times 19$$

## Coprime

- Two integers $n_1$ and $n_2$ are relatively prime or coprimes iff

$$\gcd(n_1, n_2) = 1$$

$$\text{Example}: \gcd(15, 16) = \gcd\left(3 \times 5, 2^4\right) = 1$$

$$\text{Example}: \gcd(700, 392) = \gcd\left(2^2\, 5^2\, 7, 2^3\, 7^2\right)$$

$$= \gcd\left(\mathbf{2^2}\, 5^2\, \mathbf{7^1}, \mathbf{2^2}\, 2^1\, \mathbf{7^1}\, 7^1\right) = \mathbf{2^2}\, \mathbf{7^1} = 28 \neq 1$$

# Math Concepts (Review)

**Inverse**

- In $\mathbf{Z}_n$ $i = a^{-1}$ is the (multiplicative) inverse to $a$ iff
  $a\ i = 1 \bmod n$

$$a \in \mathbf{Z}_n,\ : \gcd(a, n) = 1 \Leftrightarrow a^{-1} \in \mathbf{Z}_n$$

- This means that if $a$ is a coprime to $n$ then $a^{-1}$ exists

**Euler's totient function**

- In $\mathbf{Z}_n$ $\phi(n)$ gives the number of coprimes to $n$

$$\phi(n) = n \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right)$$

# Problem

- How many invertible elements are there in $\mathbf{Z}_{19}$?
- in $\mathbf{Z}_{63}$?

# Math Concepts (Review)

## Euler's theorem

$$a \in \mathbf{Z}_n,\ \gcd(a, n) = 1 \Rightarrow a^{\phi(n)} \equiv 1 \bmod (n)$$

Example: What's $5^{10200} \bmod 10403$?

- Consider $\mathbf{Z}_n$, $n = 10403 = 101 \times 103$.
- We have $\phi(10403) = (101-1)(103-1) = 10200$.
- Then since $n$ and $5$ are coprimes we have:
  $5^{10200} = 5^{\phi(10403)} = 1 \bmod 10403$

# Problem

- Compute $10^{842}$ in $\mathbf{Z}_{147}$

# RSA

- RSA encryption was introduced in 1977 by
  Ron **R**ivest, Adi **S**hamir and Len **A**dleman
- Its security is based on the fact that it is time-consuming to factorize large numbers
- The encryption method makes use of Euler's theorem

# RSA Cryptosystem

- Setup:
  - $n = pq$, with $p$ and $q$ primes
  - $e$ relatively prime to $\phi(n) = (p-1)(q-1)$
  - $d$ inverse of $e$ in $Z_{\phi(n)}$
- Keys:
  - Public key: $K_E = (n, e)$
  - Private key: $K_D = d$
- Encryption:
  - Plaintext $M$ in $Z_n$
  - $C = M^e \bmod n$
- Decryption:
  - $M = C^d \bmod n$

- Example
  - Setup:
    - $p = 7, \ q = 17$
    - $n = 7 \cdot 17 = 119$
    - $\phi(n) = 6 \cdot 16 = 96$
    - $e = 5$
    - $d = 77$
  - Keys:
    - public key: $(119, 5)$
    - private key: $77$
  - Encryption:
    - $M = 19$
    - $C = 19^5 = 66 \bmod 119$
  - Decryption:
    - $C = 66^{77} = 19 \bmod 119$

# Complete RSA Example

- Setup:
  - $\boldsymbol{p} = 5,\ \boldsymbol{q} = 11$
  - $\boldsymbol{n} = 5 \cdot 11 = 55$
  - $\phi(\boldsymbol{n}) = 4 \cdot 10 = 40$
  - $\boldsymbol{e} = 3$
  - $\boldsymbol{d} = 27\ (3 \cdot 27 = 81 = 2 \cdot 40 + 1)$
- Encryption
  - $\boldsymbol{C} = \boldsymbol{M}^3 \bmod 55$
- Decryption
  - $\boldsymbol{M} = \boldsymbol{C}^{27} \bmod 55$

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $M$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| $C$ | 1 | 8 | 27 | 9 | 15 | 51 | 13 | 17 | 14 | 10 | 11 | 23 | 52 | 49 | 20 | 26 | 18 | 2 |
| $M$ | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
| $C$ | 39 | 25 | 21 | 33 | 12 | 19 | 5 | 31 | 48 | 7 | 24 | 50 | 36 | 43 | 22 | 34 | 30 | 16 |
| $M$ | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 |
| $C$ | 53 | 37 | 29 | 35 | 6 | 3 | 32 | 44 | 45 | 41 | 38 | 42 | 4 | 40 | 46 | 28 | 47 | 54 |

# Problem

- With $n = 323$ and $e = 5$ encrypt the message $m = 4$

# Security

- Security of RSA based on difficulty of factoring
  - Widely believed
  - Best known algorithm takes exponential time
- RSA Security factoring challenge (discontinued)
- In 1999, 512-bit challenge factored in 4 months using 35.7 CPU-years
  - 160 175-400 MHz SGI and Sun
  - 8 250 MHz SGI Origin
  - 120 300-450 MHz Pentium II
  - 4 500 MHz Digital/Compaq
- In 2005, a team of researchers factored the RSA-640 challenge number using 30 2.2GHz CPU years
- In 2004, the prize for factoring RSA-2048 was $200,000
- Current practice is 2,048-bit keys
- Estimated resources needed to factor a number within one year

| Length (bits) | PCs | Memory |
|---|---|---|
| 430 | 1 | 128MB |
| 760 | 215,000 | 4GB |
| 1,020 | $342\times10^6$ | 170GB |
| 1,620 | $1.6\times10^{15}$ | 120TB |

# Correctness

- We show the correctness of the RSA cryptosystem for the case when the plaintext $M$ does not divide $n$
- Namely, we show that

  $$(M^e)^d \bmod n = M$$

- Since $ed \bmod \phi(n) = 1$, there is an integer $k$ such that

  $$ed = k\phi(n) + 1$$

- Since $M$ does not divide $n$, by Euler's theorem we have

  $$M^{\phi(n)} \bmod n = 1$$

- Thus, we obtain

  $$\begin{aligned}
  &(M^e)^d \bmod n = \\
  &\quad M^{ed} \bmod n = \\
  &\quad M^{k\phi(n)+1} \bmod n = \\
  &\quad MM^{k\phi(n)} \bmod n = \\
  &\quad M\,(M^{\phi(n)})^k \bmod n = \\
  &\quad M\,(M^{\phi(n)} \bmod n)^k \bmod n = \\
  &\quad M\,(1)^k \bmod n = \\
  &\quad M \bmod n = \\
  &\quad M
  \end{aligned}$$

- Proof of correctness can be extended to the case when the plaintext $M$ divides $n$

# Algorithmic Issues

- The implementation of the RSA cryptosystem requires various algorithms
- Overall
  - Representation of integers of arbitrarily large size and arithmetic operations on them
- Encryption
  - Modular power
- Decryption
  - Modular power
- Setup
  - Generation of random numbers with a given number of bits (to generate candidates $p$ and $q$)
  - Primality testing (to check that candidates $p$ and $q$ are prime)
  - Computation of the GCD (to verify that $e$ and $\phi(n)$ are relatively prime)
  - Computation of the multiplicative inverse (to compute $d$ from $e$)

# Modular Power

- The repeated squaring algorithm speeds up the computation of a modular power $a^p \bmod n$
- Write the exponent $p$ in binary

  $p = p_{b-1} p_{b-2} \cdots p_1 p_0$
- Start with

  $Q_1 = a^{p_{b-1}} \bmod n$
- Repeatedly compute

  $Q_i = ((Q_{i-1})^2 \bmod n) a^{p_{b-i}} \bmod n$
- We obtain

  $Q_b = a^p \bmod n$
- The repeated squaring algorithm performs $O(\log p)$ arithmetic operations

- Example
  - $3^{18} \bmod 19$ ($18 = 10010$)
  - $Q_1 = 3^1 \bmod 19 = 3$
  - $Q_2 = (3^2 \bmod 19)3^0 \bmod 19 = 9$
  - $Q_3 = (9^2 \bmod 19)3^0 \bmod 19 =$ $81 \bmod 19 = 5$
  - $Q_4 = (5^2 \bmod 19)3^1 \bmod 19 =$ $(25 \bmod 19)3 \bmod 19 =$ $18 \bmod 19 = 18$
  - $Q_5 = (18^2 \bmod 19)3^0 \bmod 19 =$ $(324 \bmod 19) \bmod 19 =$ $17 \cdot 19 + 1 \bmod 19 = 1$

# Problem

- Compute $9^{100} \bmod 147$

# Modular Inverse

Theorem

Given positive integers $\boldsymbol{a}$ and $\boldsymbol{b}$, let $\boldsymbol{d}$ be the smallest positive integer such that

$$\boldsymbol{d} = \boldsymbol{ia} + \boldsymbol{jb}$$

for some integers $\boldsymbol{i}$ and $\boldsymbol{j}$.

We have

$$\boldsymbol{d} = \gcd(\boldsymbol{a},\boldsymbol{b})$$

- Example
  - $\boldsymbol{a} = 21$
  - $\boldsymbol{b} = 15$
  - $\boldsymbol{d} = 3$
  - $\boldsymbol{i} = 3, \boldsymbol{j} = -4$
  - $3 = 3 \cdot 21 + (-4) \cdot 15 = 63 - 60 = 3$

- Given positive integers $\boldsymbol{a}$ and $\boldsymbol{b}$, the extended Euclid's algorithm computes a triplet $(\boldsymbol{d},\boldsymbol{i},\boldsymbol{j})$ such that
  - $\boldsymbol{d} = \gcd(\boldsymbol{a},\boldsymbol{b})$
  - $\boldsymbol{d} = \boldsymbol{ia} + \boldsymbol{jb}$
- To test the existence of and compute the inverse of $\boldsymbol{x} \in \boldsymbol{Z_n}$, we execute the extended Euclid's algorithm on the input pair $(\boldsymbol{x},\boldsymbol{n})$
- Let $(\boldsymbol{d},\boldsymbol{i},\boldsymbol{j})$ be the triplet returned
  - $\boldsymbol{d} = \boldsymbol{ix} + \boldsymbol{jn}$

Case 1: $\boldsymbol{d} = 1$

$\boldsymbol{i}$ is the inverse of $\boldsymbol{x}$ in $\boldsymbol{Z_n}$

Case 2: $\boldsymbol{d} > 1$

$\boldsymbol{x}$ has no inverse in $\boldsymbol{Z_n}$

# Problem

- Compute $117^{-1}$ in $\mathbf{Z}_{337}$

# Problem

- With $n = 323$ and $e = 5$, find $d$ and decrypt the cipher $c = 300$ (home work)