Chapter 10

Asymmetric-Key Cryptography
10.1.1 Keys

Asymmetric key cryptography uses two separate keys: one private and one public.

Figure 10.1 Locking and unlocking in asymmetric-key cryptosystem
10.1.2 General Idea

Figure 10.2 General idea of asymmetric-key cryptosystem
Plaintext/Ciphertext
Unlike in symmetric-key cryptography, plaintext and ciphertext are treated as integers in asymmetric-key cryptography.

Encryption/Decryption

\[ C = f(K_{public}, P) \quad P = g(K_{private}, C) \]
10.1.3 Need for Both

There is a very important fact that is sometimes misunderstood: The advent of asymmetric-key cryptography does not eliminate the need for symmetric-key cryptography.
The main idea behind asymmetric-key cryptography is the concept of the trapdoor one-way function.
One-Way Function (OWF)

1. \( f \) is easy to compute.
2. \( f^{-1} \) is difficult to compute.

Trapdoor One-Way Function (TOWF)

3. Given \( y \) and a trapdoor, \( x \) can be computed easily.
Example 10. 1

When \( n \) is large, \( n = p \times q \) is a one-way function. Given \( p \) and \( q \), it is always easy to calculate \( n \); given \( n \), it is very difficult to compute \( p \) and \( q \). This is the factorization problem.

Example 10. 2

When \( n \) is large, the function \( y = x^k \mod n \) is a trapdoor one-way function. Given \( x \), \( k \), and \( n \), it is easy to calculate \( y \). Given \( y \), \( k \), and \( n \), it is very difficult to calculate \( x \). This is the discrete logarithm problem. However, if we know the trapdoor, \( k' \) such that \( k \times k' = 1 \mod \phi(n) \), we can use \( x = y^{k'} \mod n \) to find \( x \).
10.1.5 Knapsack Cryptosystem

**Definition**

\[ a = [a_1, a_2, \ldots, a_k] \text{ and } x = [x_1, x_2, \ldots, x_k]. \]

\[ s = \text{knapsackSum} (a, x) = x_1a_1 + x_2a_2 + \cdots + x_ka_k \]

Given \( a \) and \( x \), it is easy to calculate \( s \). However, given \( s \) and \( a \) it is difficult to find \( x \).

**Superincreasing Tuple**

\[ a_i \geq a_1 + a_2 + \cdots + a_{i-1} \]
**Algorithm 10.1**  \textit{knap sackSum} and \textit{inv_knapsackSum} for a superincreasing k-tuple

\begin{tabular}{|l|}
\hline
\textbf{knapsackSum} (x [1 \ldots k], a [1 \ldots k])  \\
\{  
\hspace{1em} s \leftarrow 0  
\hspace{1em} for (i = 1 \ to \ k)  
\hspace{1em} \{  
\hspace{2em} s \leftarrow s + a_i \times x_i  
\hspace{1em} \}  
\hspace{1em} return s  
\}  \\
\hline
\end{tabular}

\begin{tabular}{|l|}
\hline
\textbf{inv_knapsackSum} (s, a [1 \ldots k])  \\
\{  
\hspace{1em} for (i = k \ down \ to \ 1)  
\hspace{1em} \{  
\hspace{2em} if s \geq a_i  
\hspace{2em} \{  
\hspace{3em} x_i \leftarrow 1  
\hspace{3em} s \leftarrow s - a_i  
\hspace{3em} \}  
\hspace{2em} else x_i \leftarrow 0  
\hspace{1em} \}  
\hspace{1em} return x [1 \ldots k]  
\}  \\
\hline
\end{tabular}
Example 10.3

As a very trivial example, assume that \( a = [17, 25, 46, 94, 201, 400] \) and \( s = 272 \) are given. Table 10.1 shows how the tuple \( x \) is found using inv_knapsackSum routine in Algorithm 10.1. In this case \( x = [0, 1, 1, 0, 1, 0] \), which means that 25, 46, and 201 are in the knapsack.

<table>
<thead>
<tr>
<th>( i )</th>
<th>( a_i )</th>
<th>( s )</th>
<th>( s \geq a_i )</th>
<th>( x_i )</th>
<th>( s \leftarrow s - a_i \times x_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>400</td>
<td>272</td>
<td>false</td>
<td>( x_6 = 0 )</td>
<td>272</td>
</tr>
<tr>
<td>5</td>
<td>201</td>
<td>272</td>
<td>true</td>
<td>( x_5 = 1 )</td>
<td>71</td>
</tr>
<tr>
<td>4</td>
<td>94</td>
<td>71</td>
<td>false</td>
<td>( x_4 = 0 )</td>
<td>71</td>
</tr>
<tr>
<td>3</td>
<td>46</td>
<td>71</td>
<td>true</td>
<td>( x_3 = 1 )</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>25</td>
<td>true</td>
<td>( x_2 = 1 )</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>0</td>
<td>false</td>
<td>( x_1 = 0 )</td>
<td>0</td>
</tr>
</tbody>
</table>
Secret Communication with Knapsacks.

Figure 10.4 Secret communication with knapsack cryptosystem

Key generation
Select $b = [b_1, b_2, \ldots, b_k]$  
Select modulus $n$ and $r$  
Calculate $a = [a_1, a_2, \ldots, a_k]$

To public

Private

Bob

Alice

Encryption
$s = \text{knapsackSum} (x, a)$

Decryption

$s' = r^{-1} \times s \mod n$

$x' = \text{inv_knapsackSum} (s', b)$

$x = \text{permute} (x')$

Plaintext

$x$

$x$

Plaintext

$x$

Plaintext

$x$

Plaintext
This is a trivial (very insecure) example just to show the procedure.

1. Key generation:
   a. Bob creates the superincreasing tuple \( b = [7, 11, 19, 39, 79, 157, 313] \).
   b. Bob chooses the modulus \( n = 900 \) and \( r = 37 \), and \([4 2 5 3 1 7 6]\) as permutation table.
   c. Bob now calculates the tuple \( t = [259, 407, 703, 543, 223, 409, 781] \).
   d. Bob calculates the tuple \( a = \text{permute} (t) = [543, 407, 223, 703, 259, 781, 409] \).
   e. Bob publicly announces \( a \); he keeps \( n, r, \) and \( b \) secret.

2. Suppose Alice wants to send a single character “g” to Bob.
   a. She uses the 7-bit ASCII representation of “g”, \((1100111)_{2}\), and creates the tuple \( x = [1, 1, 0, 0, 1, 1, 1] \). This is the plaintext.
   b. Alice calculates \( s = \text{knapsackSum} (a, x) = 2165 \). This is the ciphertext sent to Bob.

3. Bob can decrypt the ciphertext, \( s = 2165 \).
   a. Bob calculates \( s' = s \times r^{-1} \text{ mod } n = 2165 \times 37^{-1} \text{ mod } 900 = 527 \).
   b. Bob calculates \( x' = \text{Inv_knapsackSum} (s', b) = [1, 1, 0, 1, 0, 1, 1] \).
   c. Bob calculates \( x = \text{permute} (x') = [1, 1, 0, 0, 1, 1, 1] \). He interprets the string \((1100111)_{2}\)
      as the character “g”.

10.13
The most common public-key algorithm is the RSA cryptosystem, named for its inventors (Rivest, Shamir, and Adleman).

**Topics discussed in this section:**
- 10.2.1 Introduction
- 10.2.2 Procedure
- 10.2.3 Some Trivial Examples
- 10.2.4 Attacks on RSA
- 10.2.5 Recommendations
- 10.2.6 Optimal Asymmetric Encryption Padding (OAEP)
- 10.2.7 Applications
**10.2.1 Introduction**

**Figure 10.5** *Complexity of operations in RSA*

RSA uses modular exponentiation for encryption/decryption; To attack it, Eve needs to calculate $e \sqrt[\phi(n)]{C} \mod n$. 
10.2.2 Procedure

Figure 10.6 Encryption, decryption, and key generation in RSA
10.2.2 Continued

Two Algebraic Structures

Encryption/Decryption Ring: \( R = <\mathbb{Z}_n, +, \times > \)

Key-Generation Group: \( G = <\mathbb{Z}_{\phi(n)} *, \times > \)

RSA uses two algebraic structures:
a public ring \( R = <\mathbb{Z}_n, +, \times > \) and a private group \( G = <\mathbb{Z}_{\phi(n)} *, \times > \).

In RSA, the tuple \((e, n)\) is the public key; the integer \(d\) is the private key.
10.2.2 Continued

**Algorithm 10.2  RSA Key Generation**

`RSA_Key_Generation`

```plaintext
{
    Select two large primes $p$ and $q$ such that $p \neq q$.
    $n \leftarrow p \times q$
    $\phi(n) \leftarrow (p - 1) \times (q - 1)$
    Select $e$ such that $1 < e < \phi(n)$ and $e$ is coprime to $\phi(n)$
    $d \leftarrow e^{-1} \mod \phi(n)$  \hspace{1cm} // d is inverse of $e$ modulo $\phi(n)$
    Public_key $\leftarrow (e, n)$ \hspace{1cm} // To be announced publicly
    Private_key $\leftarrow d$ \hspace{1cm} // To be kept secret
    return Public_key and Private_key
}
```
10.2.2 Continued

Encryption

Algorithm 10.3  RSA encryption

\[
\text{RSA\_Encryption} (P, e, n) \quad \text{// } P \text{ is the plaintext in } \mathbb{Z}_n \text{ and } P < n \\
\{
\text{C } \leftarrow \text{Fast\_Exponentiation} (P, e, n) \quad \text{// Calculation of } (P^e \mod n) \\
\text{return C}
\}
\]

In RSA, \( p \) and \( q \) must be at least 512 bits; \( n \) must be at least 1024 bits.
10.2.2 Continued

Decryption

Algorithm 10.4  RSA decryption

\[
\text{RSA\_Decryption} (C, d, n) \quad \text{//}C \text{ is the ciphertext in } Z_n \\
\{
    \ P \leftarrow \text{Fast\_Exponentiation} (C, d, n) \quad \text{// Calculation of } (C^d \text{ mod } n) \\
    \text{return } P
\}
\]
If $n = p \times q$, $a < n$, and $k$ is an integer, then $a^{k\phi(n)+1} \equiv a \pmod{n}$.

\[ P_1 = C^d \mod n = (P^e \mod n)^d \mod n = P^{ed} \mod n \]
\[ ed = k\phi(n) + 1 \quad \text{// $d$ and $e$ are inverses modulo $\phi(n)$} \]
\[ P_1 = P^{ed} \mod n \rightarrow P_1 = P^{k\phi(n)+1} \mod n \]
\[ P_1 = P^{k\phi(n)+1} \mod n = P \mod n \quad \text{// Euler’s theorem (second version)} \]
10.2.3 Some Trivial Examples

Example 10.5

Bob chooses 7 and 11 as $p$ and $q$ and calculates $n = 77$. The value of $\phi(n) = (7 - 1)(11 - 1)$ or 60. Now he chooses two exponents, $e$ and $d$, from $\mathbb{Z}_{60}^*$. If he chooses $e$ to be 13, then $d$ is 37. Note that $e \times d \mod 60 = 1$ (they are inverses of each other).

Now imagine that Alice wants to send the plaintext 5 to Bob. She uses the public exponent 13 to encrypt 5.

| Plaintext: 5 | $C = 5^{13} = 26 \mod 77$ | Ciphertext: 26 |

Bob receives the ciphertext 26 and uses the private key 37 to decipher the ciphertext:

| Ciphertext: 26 | $P = 26^{37} = 5 \mod 77$ | Plaintext: 5 |
10.2.3 Some Trivial Examples

Example 10.6

Now assume that another person, John, wants to send a message to Bob. John can use the same public key announced by Bob (probably on his website), 13; John’s plaintext is 63. John calculates the following:

Plaintext: 63 \quad C = 63^{13} \equiv 28 \pmod{77} \quad \text{Ciphertext: 28}

Bob receives the ciphertext 28 and uses his private key 37 to decipher the ciphertext:

Ciphertext: 28 \quad P = 28^{37} \equiv 63 \pmod{77} \quad \text{Plaintext: 63}
Jennifer creates a pair of keys for herself. She chooses $p = 397$ and $q = 401$. She calculates $n = 159197$. She then calculates $\phi(n) = 158400$. She then chooses $e = 343$ and $d = 12007$. Show how Ted can send a message to Jennifer if he knows $e$ and $n$.

Suppose Ted wants to send the message “NO” to Jennifer. He changes each character to a number (from 00 to 25), with each character coded as two digits. He then concatenates the two coded characters and gets a four-digit number. The plaintext is 1314. Figure 10.7 shows the process.
Figure 10.7 Encryption and decryption in Example 10.7

Ted

"NO"

Encode

P = 1314

C = 1314^{343} \mod 159197

(343, 159197)

C = 33677

Jennifer

"NO"

Decode

P = 33677^{12007} \mod 159197

(12007)

P = 1314
10.2.4 Attacks on RSA

Figure 10.8 Taxonomy of potential attacks on RSA

- Factorization
  - Chosen-ciphertext
  - Encryption exponent
  - Decryption exponent
  - Plaintext
  - Modulus
  - Implementation

Coppersmith, broadcast, related messages, and short pad
Revealed and low exponent
Short message, cyclic, and unconcealed
Common modulus
Timing and power
10.2.6 OAEP

Figure 10.9  *Optimal asymmetric encryption padding (OAEP)*

M: Padded message  P: Plaintext ($P_1 \parallel P_2$)  G: Public function ($k$-bit to $m$-bit)
r: One-time random number  C: Ciphertext  H: Public function ($m$-bit to $k$-bit)

Sender

Receiver
Here is a more realistic example. We choose a 512-bit $p$ and $q$, calculate $n$ and $\phi(n)$, then choose $e$ and test for relative primeness with $\phi(n)$. We then calculate $d$. Finally, we show the results of encryption and decryption. The integer $p$ is a 159-digit number.

\[
p = 9613034531358350457419158128061542790930984559499621582258315087964794045505647063849125716018034750312098666606492420191808780667421096063354219926661209
\]

\[
q = 12060191957231446918276794204450896001555925054637033936061798321731482148483764659215389453209175225273226830107120695604602513887145524969000359660045617
\]
The modulus $n = p \times q$. It has 309 digits.

\[
n = \begin{array}{l}
115935041739676149688925098646158875237714573754541447754855261376 \\
147885408326350817276878815968325168468849300625485764111250162414 \\
552339182927162507656772727460097082714127730434960500556347274566 \\
628060099924037102991424472292215772798531727033839381334692684137 \\
327622000966676671831831088373420823444370953
\end{array}
\]

\[\phi(n) = (p - 1)(q - 1)\] has 309 digits.

\[
\phi(n) = \begin{array}{l}
115935041739676149688925098646158875237714573754541447754855261376 \\
147885408326350817276878815968325168468849300625485764111250162414 \\
552339182927162507656772727460097082714127730434960500556347274566 \\
013923444405716989581728196098226361075467211864612171359107358640 \\
614008885170265377277264467341066243857664128
\end{array}
\]
10.2.6 Continued

Example 10.8  Continued

Bob chooses $e = 35535$ (the ideal is 65537) and tests it to make sure it is relatively prime with $\phi(n)$. He then finds the inverse of $e$ modulo $\phi(n)$ and calls it $d$.

<table>
<thead>
<tr>
<th>$e =$</th>
<th>35535</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d =$</td>
<td>580083028600377639360936612896779175946690620896509621804228661113 805938528223587317062869100300217108590443384021707298690876006115 306202524959884448047568240966247081485817130463240644077704833134 010850947385295645071936774061197326557424237217617674620776371642 0760033708533328853214470885955136670294831</td>
</tr>
</tbody>
</table>
Alice wants to send the message “THIS IS A TEST”, which can be changed to a numeric value using the 00–26 encoding scheme (26 is the space character).

\[ P = 1907081826081826002619041819 \]

The ciphertext calculated by Alice is \( C = P^e \), which is

\[ C = 475309123646226827206365550610545180942371796070491716523239243054 \]
\[ 452960613199328566617843418359114151197411252005682979794571736036 \]
\[ 101278218847892741566090480023507190715277185914975188465888632101 \]
\[ 148354103361657898467968386763733765777465625079280521148141844048 \]
\[ 14184430812773059004692874248559166462108656 \]
10.2.6 Continued

Example 10.8 Continued

Bob can recover the plaintext from the ciphertext using
\[ P = C^d, \text{ which is} \]

\[
\begin{array}{c|c}
P = & 1907081826081826002619041819 \\
\end{array}
\]

The recovered plaintext is “THIS IS A TEST” after decoding.
The Rabin cryptosystem can be thought of as an RSA cryptosystem in which the value of $e$ and $d$ are fixed. The encryption is $C \equiv P^2 \pmod{n}$ and the decryption is $P \equiv C^{1/2} \pmod{n}$.

**Topics discussed in this section:**

10.3.1 Procedure

10.3.2 Security of the Rabin System
Figure 10.10  

**Rabin cryptosystem**

- **Key generation**
  - Select $p, q$
  - $n = p \times q$

- **Public**
  - $(p, q)$

- **Private**
  - $(p, q)$

- **Alice**
  - $P$ (Plaintext)
  - $C = P^2 \mod n$
  - Encryption in $<Z_{n^*}, \times>$

- **Bob**
  - $P$ (Plaintext)

- **Eve**
  - $P = \sqrt{C} \mod n$
  - Infeasible

- **Quadratic residues**
  - Decryption in $<Z_{n^*}, \times>$
10.3.1 Procedure

Key Generation

Algorithm 10.6  Key generation for Rabin cryptosystem

Rabin-Key_Generation

{  
Choose two large primes $p$ and $q$ in the form $4k + 3$ and $p \neq q$.
    \[ n \leftarrow p \times q \]
Public_key $\leftarrow n$  // To be announced publicly
Private_key $\leftarrow (q, n)$  // To be kept secret
return Public_key and Private_key
}

10.3.1 Continued

Encryption

Algorithm 10.7 Encryption in Rabin cryptosystem

Rabin_Encryption (n, P) // n is the public key; P is the ciphertext from \( \mathbb{Z}_n^* \)
{
    C \leftarrow P^2 \mod n  // C is the ciphertext
    return C
}

10.3.1 Continued

**Decryption**

Algorithm 10.8  **Decryption in Rabin cryptosystem**

Rabin_Decryption \((p, q, C)\)  // \(C\) is the ciphertext; \(p\) and \(q\) are private keys

\[
\begin{align*}
\{ & a_1 \leftarrow + (C^{(p+1)/4}) \mod p \\
& a_2 \leftarrow - (C^{(p+1)/4}) \mod p \\
& b_1 \leftarrow + (C^{(q+1)/4}) \mod q \\
& b_2 \leftarrow - (C^{(q+1)/4}) \mod q \\
\} \\
// The algorithm for the Chinese remainder algorithm is called four times. \\
P_1 \leftarrow \text{Chinese}_\text{Remainder} (a_1, b_1, p, q) \\
P_2 \leftarrow \text{Chinese}_\text{Remainder} (a_1, b_2, p, q) \\
P_3 \leftarrow \text{Chinese}_\text{Remainder} (a_2, b_1, p, q) \\
P_4 \leftarrow \text{Chinese}_\text{Remainder} (a_2, b_2, p, q) \ \text{return} \ P_1, P_2, P_3, \text{and} \ P_4
\end{align*}
\]

**Note**

The Rabin cryptosystem is not deterministic: Decryption creates four plaintexts.
Here is a very trivial example to show the idea.

1. Bob selects $p = 23$ and $q = 7$. Note that both are congruent to 3 mod 4.

2. Bob calculates $n = p \times q = 161$.

3. Bob announces $n$ publicly; he keeps $p$ and $q$ private.

4. Alice wants to send the plaintext $P = 24$. Note that 161 and 24 are relatively prime; 24 is in $\mathbb{Z}_{161}^*$. She calculates $C = 24^2 = 93$ mod 161, and sends the ciphertext 93 to Bob.
5. Bob receives 93 and calculates four values:
   \[ a_1 = +(93 \ (23+1)/4) \mod 23 = 1 \mod 23 \]
   \[ a_2 = -(93 \ (23+1)/4) \mod 23 = 22 \mod 23 \]
   \[ b_1 = +(93 \ (7+1)/4) \mod 7 = 4 \mod 7 \]
   \[ b_2 = -(93 \ (7+1)/4) \mod 7 = 3 \mod 7 \]

6. Bob takes four possible answers, \((a_1, b_1), (a_1, b_2), (a_2, b_1),\) and \((a_2, b_2),\) and uses the Chinese remainder theorem to find four possible plaintexts: 116, 24, 137, and 45. Note that only the second answer is Alice’s plaintext.
Besides RSA and Rabin, another public-key cryptosystem is ElGamal. ElGamal is based on the discrete logarithm problem discussed in Chapter 9.

**Topics discussed in this section:**
- 10.4.1 ElGamal Cryptosystem
- 10.4.2 Procedure
- 10.4.3 Proof
- 10.4.4 Analysis
- 10.4.5 Security of ElGamal
- 10.4.6 Application
10.4.2 Procedure

**Figure 10.11** *Key generation, encryption, and decryption in ElGamal*

- **Key generation**
  - Select $p$ (very large prime)
  - Select $e_1$ (primitive root)
  - Select $d$
  - $e_2 = e_1^d \mod p$
  - Private key: $d$

- **Encryption**
  - $C_1 = e_1^r \mod p$
  - $C_1 = (e_2^r \times P) \mod p$

- **Decryption**
  - $P = [C_2 \times (C_1^d)^{-1}] \mod p$

Alice → Public key: $(e_1, e_2, p)$ → (Alice’s secret key) → $P = e_1^r \mod p$ → Bob

Bob → Ciphertext: $(C_1, C_2)$ → (Bob’s secret key) → $P = [C_2 \times (C_1^d)^{-1}] \mod p$ → Plaintext

Alice → Plaintext → Bob

Bob → Ciphertext → Alice

Bob → Plaintext
Key Generation

Algorithm 10.9  ElGamal key generation

ElGamal_Key_Generation
{
    Select a large prime \( p \)
    Select \( d \) to be a member of the group \( \mathbf{G} = \langle \mathbb{Z}_p^*, \times \rangle \) such that \( 1 \leq d \leq p - 2 \)
    Select \( e_1 \) to be a primitive root in the group \( \mathbf{G} = \langle \mathbb{Z}_p^*, \times \rangle \)
    \( e_2 \leftarrow e_1^d \mod p \)
    Public_key \( \leftarrow (e_1, e_2, p) \)  // To be announced publicly
    Private_key \( \leftarrow d \)  // To be kept secret

    return Public_key and Private_key
}
Algorithm 10.10 \textit{ElGamal encryption}

\textbf{ElGamal\_Encryption}$(e_1, e_2, p, P)$ \hspace{1cm} // $P$ is the plaintext

\{
    \begin{itemize}
    \item Select a random integer $r$ in the group $G = \langle Z_p^*, \times \rangle$
    \item $C_1 \leftarrow e_1^r \mod p$
    \item $C_2 \leftarrow (P \times e_2^r) \mod p$ \hspace{1cm} // $C_1$ and $C_2$ are the ciphertexts
    \item return $C_1$ and $C_2$
    \end{itemize}
\}
The bit-operation complexity of encryption or decryption in ElGamal cryptosystem is polynomial.
Example 10.10

Here is a trivial example. Bob chooses $p = 11$ and $e_1 = 2$. and $d = 3$  
$e_2 = e_1^d = 8$. So the public keys are $(2, 8, 11)$ and the private key is 3. Alice chooses $r = 4$ and calculates $C_1$ and $C_2$ for the plaintext 7.

Plaintext: 7

$C_1 = e_1^r \mod 11 = 16 \mod 11 = 5 \mod 11$

$C_2 = (P \times e_2^r) \mod 11 = (7 \times 4096) \mod 11 = 6 \mod 11$

Ciphertext: (5, 6)

Bob receives the ciphertexts (5 and 6) and calculates the plaintext.

$[C_2 \times (C_1^d)^{-1}] \mod 11 = 6 \times (5^3)^{-1} \mod 11 = 6 \times 3 \mod 11 = 7 \mod 11$

Plaintext: 7
Example 10.11

Instead of using $P = [C_2 \times (C_1^d)^{-1}] \mod p$ for decryption, we can avoid the calculation of multiplicative inverse and use $P = [C_2 \times C_1^{p-1-d}] \mod p$ (see Fermat’s little theorem in Chapter 9). In Example 10.10, we can calculate $P = [6 \times 5^{11-1-3}] \mod 11 = 7 \mod 11$.

Note

For the ElGamal cryptosystem, $p$ must be at least 300 digits and $r$ must be new for each encipherment.
Bob uses a random integer of 512 bits. The integer \( p \) is a 155-digit number (the ideal is 300 digits). Bob then chooses \( e_1, d, \) and calculates \( e_2 \), as shown below:

| \( p = \) | 11534899272561676244925313717014331740490094532609834959814346921905689869862264593212975473787189514436889176526473093615929993728061165964347353440008577 |
| \( e_1 = \) | 2 |
| \( d = \) | 1007 |
| \( e_2 = \) | 9788641304300918950876685693809773904388006288733768761002206223325545070741561892123183177046101416733601508841329408572485377031582066010072558707455 |
Alice has the plaintext $P = 3200$ to send to Bob. She chooses $r = 545131$, calculates $C_1$ and $C_2$, and sends them to Bob.

<table>
<thead>
<tr>
<th>$P =$</th>
<th>3200</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r =$</td>
<td>545131</td>
</tr>
<tr>
<td>$C_1 =$</td>
<td>887297069383528471022570471492275663120260067256562125018188351429417223599712681114105363661705173051581533189165400973736355080295736788569060619152881</td>
</tr>
<tr>
<td>$C_2 =$</td>
<td>70845433304892994457701601238079499956743602183619244696177450692124469615516580077945555930803458896144024085995259195792097216288796813505827795664302950</td>
</tr>
</tbody>
</table>

Bob calculates the plaintext $P = C_2 \times ((C_1)^d)^{-1} \mod p = 3200 \mod p$. 

| $P =$ | 3200 |
Although RSA and ElGamal are secure asymmetric-key cryptosystems, their security comes with a price, their large keys. Researchers have looked for alternatives that give the same level of security with smaller key sizes. One of these promising alternatives is the elliptic curve cryptosystem (ECC).

Topics discussed in this section:

10.5.1 Elliptic Curves over Real Numbers
10.5.2 Elliptic Curves over GF($p$)
10.5.3 Elliptic Curves over GF($2^n$)
10.5.4 Elliptic Curve Cryptography Simulating ElGamal
The general equation for an elliptic curve is

\[ y^2 + b_1 xy + b_2 y = x^3 + a_1 x^2 + a_2 x + a_3 \]

Elliptic curves over real numbers use a special class of elliptic curves of the form

\[ y^2 = x^3 + ax + b \]
Example 10.13

Figure 10.12 shows two elliptic curves with equations $y^2 = x^3 - 4x$ and $y^2 = x^3 - 1$. Both are nonsingular. However, the first has three real roots ($x = -2$, $x = 0$, and $x = 2$), but the second has only one real root ($x = 1$) and two imaginary ones.

Figure 10.12  Two elliptic curves over a real field

a. Three real roots

b. One real and two imaginary roots
Figure 10.13  *Three adding cases in an elliptic curve*

a. \((R = P + Q)\)  
b. \((R = P + P)\)  
c. \((O = P + (\neg P))\)
10.5.1 Continued

\[ \lambda = \frac{(y_2 - y_1)}{(x_2 - x_1)} \]
\[ x_3 = \lambda^2 - x_1 - x_2 \quad y_3 = \lambda (x_1 - x_3) - y_1 \]

\[ \lambda = \frac{(3x_1^2 + a)}{(2y_1)} \]
\[ x_3 = \lambda^2 - x_1 - x_2 \quad y_3 = \lambda (x_1 - x_3) - y_1 \]

3. The intercepting point is at infinity; a point O as the point at infinity or zero point, which is the additive identity of the group.
Finding an Inverse
The inverse of a point \((x, y)\) is \((x, -y)\), where \(-y\) is the additive inverse of \(y\). For example, if \(p = 13\), the inverse of \((4, 2)\) is \((4, 11)\).

Finding Points on the Curve
Algorithm 10.12 shows the pseudocode for finding the points on the curve \(E_p(a, b)\).
Algorithm 10.12  Pseudocode for finding points on an elliptic curve

```
ellipticCurve_points (p, a, b) // p is the modulus
{
    x ← 0
    while (x < p)
    {
        w ← (x^3 + ax + b) mod p // w is y^2
        if (w is a perfect square in \( \mathbb{Z}_p \)) output (x, \sqrt{w}) (x, -\sqrt{w})
        x ← x + 1
    }
}
```
Example 10.14

The equation is \( y^2 = x^3 + x + 1 \) and the calculation is done modulo 13.

Figure 10.14  Points on an elliptic curve over GF(p)
Example 10.15

Let us add two points in Example 10.14, \( R = P + Q \), where \( P = (4, 2) \) and \( Q = (10, 6) \).

a. \( \lambda = (6 - 2) \times (10 - 4)^{-1} \mod 13 = 4 \times 6^{-1} \mod 13 = 5 \mod 13 \).

b. \( x = (5^2 - 4 - 10) \mod 13 = 11 \mod 13 \).

c. \( y = [5 (4 -11) - 2] \mod 13 = 2 \mod 13 \).

d. \( R = (11, 2) \), which is a point on the curve in Example 10.14.
To define an elliptic curve over GF($2^n$), one needs to change the cubic equation. The common equation is

$$y^2 + xy = x^3 + ax^2 + b$$

**Finding Inverses**

If $P = (x, y)$, then $-P = (x, x + y)$.

**Finding Points on the Curve**

We can write an algorithm to find the points on the curve using generators for polynomials discussed in Chapter 7.
Finding Inverses
If $P = (x, y)$, then $-P = (x, x + y)$.

Finding Points on the Curve
We can write an algorithm to find the points on the curve using generators for polynomials discussed in Chapter 7. This algorithm is left as an exercise. Following is a very trivial example.
10.5.3 Continued

Example 10.16

We choose $GF(2^3)$ with elements \{0, 1, g, g^2, g^3, g^4, g^5, g^6\} using the irreducible polynomial of $f(x) = x^3 + x + 1$, which means that $g^3 + g + 1 = 0$ or $g^3 = g + 1$. Other powers of $g$ can be calculated accordingly. The following shows the values of the g’s.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>$g^3 = g + 1$</td>
<td>011</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>$g^4 = g^2 + g$</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>010</td>
<td>$g^5 = g^2 + g + 1$</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>$g^2$</td>
<td>100</td>
<td>$g^6 = g^2 + 1$</td>
<td>101</td>
<td></td>
</tr>
</tbody>
</table>
Example 10.16  Continued

Using the elliptic curve $y^2 + xy = x^3 + g^3 x^2 + 1$, with $a = g^3$ and $b = 1$, we can find the points on this curve, as shown in Figure 10.15.

Figure 10.15  Points on an elliptic curve over $GF(2^n)$
**Adding Two Points**

1. If $P = (x_1, y_1)$, $Q = (x_2, y_2)$, $Q \neq -P$, and $Q \neq P$, then $R = (x_3, y_3) = P + Q$ can be found as

\[
\lambda = \frac{y_2 + y_1}{x_2 + x_1}
\]

\[
x_3 = \lambda^2 + \lambda + x_1 + x_2 + a
\]

\[
y_3 = \lambda (x_1 + x_3) + x_3 + y_1
\]

If $Q = P$, then $R = P + P$ (or $R = 2P$) can be found as

\[
\lambda = \frac{x_1 + y_1}{x_1}
\]

\[
x_3 = \lambda^2 + \lambda + a
\]

\[
y_3 = x_1^2 + (\lambda + 1) x_3
\]
Example 10.17

Let us find $R = P + Q$, where $P = (0, 1)$ and $Q = (g^2, 1)$. We have $\lambda = 0$ and $R = (g^5, g^4)$.

Example 10.18

Let us find $R = 2P$, where $P = (g^2, 1)$. We have $\lambda = g^2 + 1/g^2 = g^2 + g^5 = g + 1$ and $R = (g^6, g^5)$. 
10.5.4 ECC Simulating ElGamal

Figure 10.16 ElGamal cryptosystem using the elliptic curve

Note:
Operations such as addition and multiplication are over an elliptic curve group.

Key generation
Select $E_p(a, b)$
Select $e_1 = (x_1, y_1)$
Select $d$
Calculate $e_2 = (x_2, y_2) = d \times e_1$

Alice

$r$
$(e_1, e_2, E_p)$

Encryption
$C_1 = r \times e_1$
$C_2 = P + r \times e_2$

Ciphertext: $(C_1, C_2)$

Decryption
$P = C_2 - (d \times C_1)$

Bob
10.5.4 Continued

Generating Public and Private Keys

\[ E(a, b) \quad e_1(x_1, y_1) \quad d \quad e_2(x_2, y_2) = d \times e_1(x_1, y_1) \]

Encryption

\[ C_1 = r \times e_1 \]

\[ C_2 = P + r \times e_2 \]

Decryption

\[ P = C_2 - (d \times C_1) \]

The minus sign here means adding with the inverse.

Note

The security of ECC depends on the difficulty of solving the elliptic curve logarithm problem.
Example 10.19

Here is a very trivial example of encipherment using an elliptic curve over $\text{GF}(p)$.

1. Bob selects $E_{67}(2, 3)$ as the elliptic curve over $\text{GF}(p)$.

2. Bob selects $e_1 = (2, 22)$ and $d = 4$.

3. Bob calculates $e_2 = (13, 45)$, where $e_2 = d \times e_1$.

4. Bob publicly announces the tuple $(E, e_1, e_2)$.

5. Alice wants to send the plaintext $P = (24, 26)$ to Bob. She selects $r = 2$. 