Chapter

Asymmetric-Key Cryptography

10.1

Chapter 10
Objectives

- To distinguish between two cryptosystems: symmetric-key and asymmetric-key
- To introduce trapdoor one-way functions and their use in asymmetric-key cryptosystems
- To discuss the RSA cryptosystem
- To discuss the Rabin cryptosystem
- To discuss the ElGamal cryptosystem
- To discuss the elliptic curve cryptosystem

10.2

10-1 INTRODUCTION

Symmetric and asymmetric-key cryptography will exist in parallel and continue to serve the community. We actually believe that they are complements of each other; the advantages of one can compensate for the disadvantages of the other.

Topics discussed in this section:

10.1.1 Keys
10.1.2 General Idea
10.1.3 Need for Both
10.1.4 Trapdoor One-Way Function

10.3

10-1 INTRODUCTION

Symmetric and asymmetric-key cryptography will exist in parallel and continue to serve the community. We actually believe that they are complements of each other; the advantages of one can compensate for the disadvantages of the other.

Note

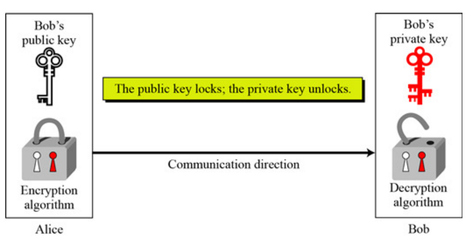
Symmetric-key cryptography is based on sharing secrecy; asymmetric-key cryptography is based on personal secrecy.

10.4

10.1.1 Keys

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

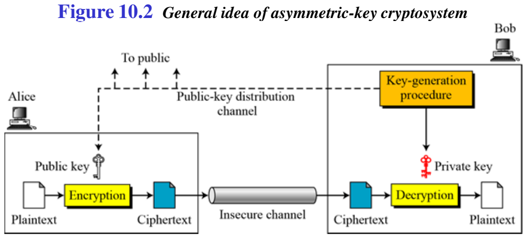
Figure 10.1 ***Locking and unlocking in asymmetric-key cryptosystem***



10.5

10.1.2 General Idea

Figure 10.2 ***General idea of asymmetric-key cryptosystem***



10.6

10.1.2 Continued

Plaintext/Ciphertext

Unlike in symmetric-key cryptography, plaintext and ciphertext are treated as integers in asymmetric-key cryptography.

Encryption/Decryption

$$\boxed{C = e(K_{public}, P) \qquad P = d(K_{private}, C)}$$

10.7

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.

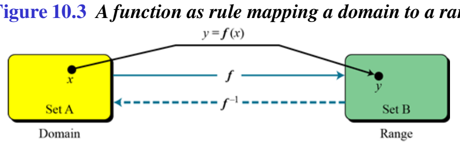
10.8

10.1.4 Trapdoor One-Way Function

The main idea behind asymmetric-key cryptography is the concept of the trapdoor one-way function.

Functions

Figure 10.3 *A function as rule mapping a domain to a range*



10.9

10.1.4 Continued

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.

10.10

10.1.4 Continued

Example 10. 1

When n is large, $n = p \times q$ is a one-way function.
Easy Given p and q → calculate n
Difficult Given n → calculate p and q
This is the factorization problem.

Example 10. 2

When n is large, the function $y = x^k \bmod n$ is a trapdoor one-way function.
Easy Given x, k, and n → calculate y
Difficult Given y, k, and n → *calculate x*
This is the discrete logarithm problem.
However, if we know the trapdoor, k′ such that $k \times k' = 1 \bmod \phi(n)$, we can use $x = y^{k'} \bmod n$ to find x.

10.11

10-2 RSA CRYPTOSYSTEM

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

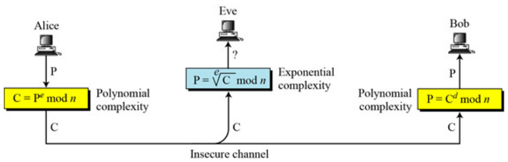
Topics discussed in this section:

10.2.1 Introduction
10.2.2 Procedure
10.2.3 Some Trivial Examples
10.2.4 Attacks on RSA

10.12

10.2.1 Introduction

Figure 10.5 *Complexity of operations in RSA*

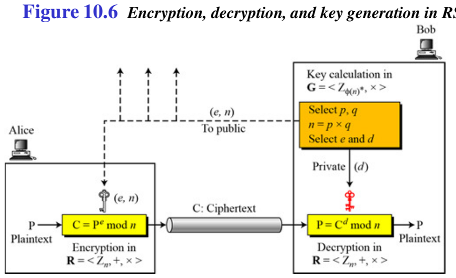


RSA uses modular exponentiation for encryption/decryption; To attack it, Eve needs to calculate $\sqrt[e]{C}$ mod n.

10.13

10.2.2 Procedure

Figure 10.6 *Encryption, decryption, and key generation in RSA*



10.14

Public key encryption algorithms

Requirements:

1. need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that

   $K_B^-(K_B^+(m)) = m$

2. given public key K_B^+, it should be impossible to compute private key K_B^-

RSA: Rivest, Shamir, Adleman algorithm

8: Network Security 8-15

RSA: Choosing keys

1. Choose two large prime numbers p, q. (e.g., 1024 bits each)
2. Compute $n = pq$, $z = (p-1)(q-1)$
3. Choose e (with $e<n$) that has no common factors with z. (e, z are "relatively prime").
4. Choose d such that $ed-1$ is exactly divisible by z. (in other words: $ed \bmod z = 1$).
5. *Public* key is $\underbrace{(n,e)}_{K_B^+}$. *Private* key is $\underbrace{(n,d)}_{K_B^-}$.

8: Network Security 8-16

RSA: Encryption, decryption

0. Given (n,b) and (n,a) as computed above
1. To encrypt bit pattern, m, compute
   $x = m^e \bmod n$ (i.e., remainder when m^e is divided by n)
2. To decrypt received bit pattern, c, compute
   $m = x^d \bmod n$ (i.e., remainder when c^d is divided by n)

Magic happens! $m = (\underbrace{m^e \bmod n}_{x})^d \bmod n$

8: Network Security 8-17

RSA example:

Bob chooses $p=5$, $q=7$. Then $n=35$, $z=24$.
$e=5$ (so e, z relatively prime).
$d=29$ (so $ed-1$ exactly divisible by z.

encrypt:

letter	m	m^e	$c = m^e \bmod n$
l	12	1524832	17

decrypt:

c	c^d	$m = c^d \bmod n$	letter
17	481968572106750915091411825223071697	12	l

8: Network Security 8-18

RSA: Why is that $m = (m^e \bmod n)^d \bmod n$

Useful number theory result: If p,q prime and $n = pq$, then:

$$x^y \bmod n = x^{y \bmod (p-1)(q-1)} \bmod n$$

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

$$= m^{ed \bmod (p-1)(q-1)} \bmod n$$

(using number theory result above)

$$= m^1 \bmod n$$

(since we chose ed to be divisible by $(p-1)(q-1)$ with remainder 1)

$$= m$$

8: Network Security 8-19

RSA: another important property

The following property will be *very* useful later:

$$\underbrace{K_B^-(K_B^+(m))}_{} = m = \underbrace{K_B^+(K_B^-(m))}_{}$$

use public key first, followed by private key

use private key first, followed by public key

Result is the same!

8: Network Security 8-20

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 $Z_{60}*$. If he chooses e to be 13, then d is 37. Note that $e \times d \bmod 60 = 1$ (they are inverses of each Now imagine that Alice wants to send the plaintext 5 to Bob. She uses the public exponent 13 to encrypt 5.

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

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

Ciphertext: 26	$P = 26^{37} = 5 \bmod 77$	Plaintext: 5

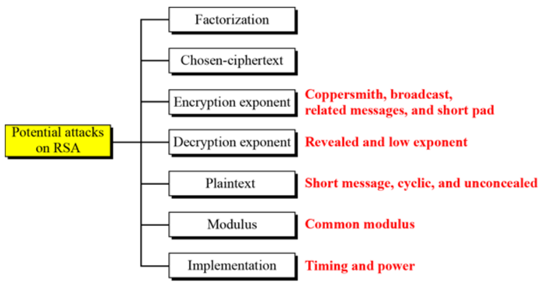
10.21

- http://www-fs.informatik.uni-tuebingen.de/~reinhard/krypto/English/4.1.e.html

10.22

10.2.4 Attacks on RSA

Figure 10.8 ***Taxonomy of potential attacks on RSA***



No devastating attacks on RSA have been yet discovered.

10.23

10.2.4 Attacks on RSA – Factorization Attack

- No devastating attacks on RSA have been yet discovered.
- Bob selects p and q and calculate n=p*q. n is public but p and q are secret. If Eve can factor n and obtain p and q, she can calculate private d from public e by
  $d = e^{-1} \bmod ((p-1)(q-1))$
- However, none of existing factorization algorithms can factor a large integer with polynomial time complexity.
- To be secure, RSA presently requires that n should be more than 300 decimal digits, which means that the modulus must be at least 1024 bits.

10.24

10.2.4 Attacks on RSA – Plaintext attack

- Plaintext attack
  - Short message attack – if it is known that Alice is sending a four-digit number to Bob, Eve can easily try plaintext numbers from 0000 to 9999 to find the plaintext. Therefore, short msg must be padded with random bits.
  - Cycling attack – the continuous encryption of the ciphertext will eventually result in the plaintext. The complexity of this is equivalent to the complexity of factoring n.

10.25

10.2.4 Attacks on RSA

- Encryption exponent

To reduce the encryption time, it is tempting to use a small encryption exponent e. the common value is 3. to be secure, the recommendation is to use $e = 2^{16}$+1=6537

- Decryption exponent
  - Revealed decryption exponent attack: In RSA, if d is comprised, then p, q, n, e, and d must be regenerated.
  - Low decryption exponent attack. In RSA, the recommendation is to have $d \geq \frac{1}{3} n^{\frac{1}{4}}$ to prevent low decryption exponent attack.

10.26

10.2.4 Attacks on RSA – Chosen ciphertext attack

- Chosen-ciphertext attack
  - Assume Alice creates the ciphertext and sends C to Bob. Also assume that Bob will decrypt an arbitrary ciphertext for Eve, other than C.
  - Eve intercepts C and uses the following steps:
    - Eve chooses a random integer X
    - Eve calculates Y = C * X^e mod n
    - Eve sends Y to Bob for decryption and get Z=Y^d mod n; this step is an instance of a chosen ciphertext attack.
    - Eve can easily find C because
    - Z = Y^d mod n = (C * X^e)^d mod n = (C^d * X^ed) mod n = (C^d * X) mod n = (P*X)mod n
    - Z = (P*X)mod n → P = Z*X^(-1) mod n

10.27

10.2.4 Attacks on RSA -- Attacks on implementation

- Timing attack
  - RSA fast-exponential algorithm uses
    - only squaring if the corresponding bit in the private exponent *d* is 0. requires shorter time to decrypt.
    - Both squaring and multiplication if the corresponding bit is 1. requires longer time to decrypt
  - This timing difference allows Eve to find the value of bits in *d*, one by one.
- Powering attack
  - An iteration involving multiplication and squaring consumes more power than an iteration that uses only squaring.

10.28

10.2.4 Attacks on RSA -- Attacks on implementation

- Two methods to thwart timing attacks
  - Add random delays to the exponentiations to make each exponentiation take the same amount of time
  - Blinding: multiply the ciphertext by a random number before decryption.

    1.select a secret random number r between 1 and (n-1)

    2.Calculate $C_1 = C \times r^e \bmod n$

    3.Calculate $P_1 = C_1^d \bmod n$

    4.Calculate $P = P_1 \times r^{-1} \bmod n$

  This adds multiplication to the iteration involving squaring operation only.

10.29

RSA Recommendations

1.The number of bits for n should be at least 1024. This means that n should be around 2^{1024}, or 309 decimal digits.
2.The two primes p and q must each be at least 512 bits.
3.The values of p and q should not be very close to each other.
4.Both p-1 and q-1 should have at least one large prime factor.
5.The ratio p/q should not be close to a rational number with a small enumerator or denominator.
6.The modulus n must not be shared.
7.The value of e should be $2^{16}+1$
8.If the private key d is leaked, Bob must immediately change n as well as both e and d. It has been proven that knowledge of n and one pair (e,d) can lead to the discovery of another pairs of the same modulus.
9. Message must be padded with OAEP. **A short message in RSA makes the ciphertext vulnerable to short message attack.**

10.30

Optimal asymmetric encryption padding (OAEP)

- P = P1 || P2, where P1 is the masked version of the padded message M; P2 is sent to allow Bob to find the mask
- Encryption
- Decryption
- If there is a single bit error during transmission, RSA will fail. Transmission media must be made error-free.

10.31

OAEP

- Encryption
  - Pad the plaintext to make ***m-bit*** message **M**, if M is less than m-bit
  - Choose a random number **r** of ***k-bits***. (used only once)
  - Use one-way function **G** that inputs r-bit integer and outputs m-bit integer. This is the mask.
  - $P1 = M \oplus G(r)$
  - $P2 = H(P1) \oplus r$, function **H** inputs ***m-bit*** and outputs ***k-bit***
  - C = E(P1 || P2). Use RSA encryption here.

10.32

OAEP

- Decryption
  - P = D (P1 || P2)
  - Bob first recreates the value of r:

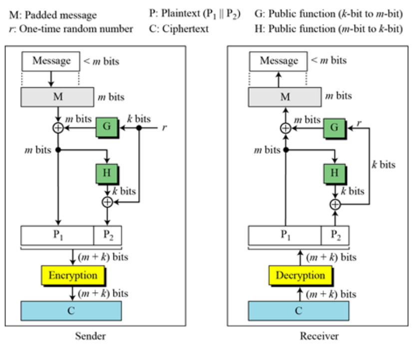
    $H(P1) \oplus P2 = H(P1) \oplus H(P1) \oplus r = r$
  - Bob recreates msg:

    $G(r) \oplus P1 = G(r) \oplus G(r) \oplus M = M$

10.33

10.2.6 OAEP

Figure 10.9 ***Optimal asymmetric encryption padding (OAEP)***



10.34

10-3 RABIN CRYPTOSYSTEM

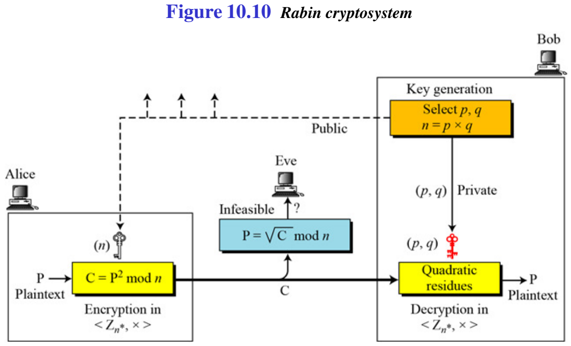
The Rabin cryptosystem can be thought of as an RSA cryptosystem in which the value of e and d are fixed.
e = 2 and d = ½
The encryption is C ≡ P² (mod n) and the decryption is P ≡ C^1/2 (mod n).

Topics discussed in this section:
10.3.1 Procedure
10.3.2 Security of the Rabin System

10.35

10-3 Continued

Figure 10.10 ***Rabin cryptosystem***



10.36

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 ≠ q.
    n ← p × q
    Public_key ← n                          // To be announced publicly
    Private_key ← (p, q)                    // To be kept secret
    return Public_key and Private_key
}
```

10.37

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 Zn*
{
    C ← P^2 mod n                // C is the ciphertext
    return C
}
```

10.38

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
{
    a1 ← +(C^((p+1)/4)) mod p
    a2 ← −(C^((p+1)/4)) mod p
    b1 ← +(C^((q+1)/4)) mod q
    b2 ← −(C^((q+1)/4)) mod q
    // The algorithm for the Chinese remainder algorithm is called four times.
    P1 ← Chinese_Remainder (a1, b1, p, q)
    P2 ← Chinese_Remainder (a1, b2, p, q)
    P3 ← Chinese_Remainder (a2, b1, p, q)
    P4 ← Chinese_Remainder (a2, b2, p, q)
    return P1, P2, P3, and P4
}
```

Note

The Rabin cryptosystem is not deterministic: Decryption creates four plaintexts.

10.39

10.3.1 Continued

Example 10. 9

Here is a very trivial example to show the idea.

1. Bob selects p = 23 and q = 7.
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 Z_{161}*.

   Encryption: $C = 24^2 \bmod 161 = 93$, and sends the ciphertext 93 to Bob.

10.40

10.3.1 Continued

Example 10. 9

5. Bob receives 93 and calculates four values:

   $a_1 = +(93^{(23+1)/4}) \bmod 23 = 1 \bmod 23$

   $a_2 = -(93^{(23+1)/4}) \bmod 23 = 22 \bmod 23$

   $b_1 = +(93^{(7+1)/4}) \bmod 7 = 4 \bmod 7$

   $b_2 = -(93^{(7+1)/4}) \bmod 7 = 3 \bmod 7$
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.

   P2 ← Chinese_Remainder(a1, b2, p, q) is to find

   x mod p = a1   24(x) mod 23(p) = 1 (a1)

   x mod q = b2   24(x) mod 7(q) = 3 (b2)

10.41

10-4 ELGAMAL CRYPTOSYSTEM

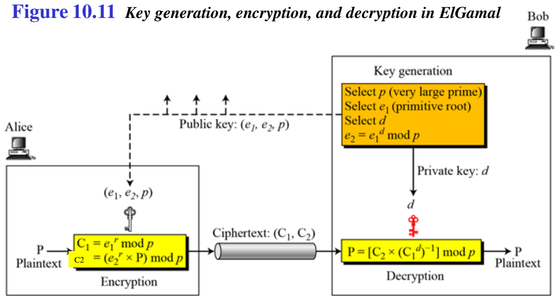
Besides RSA and Rabin, another public-key cryptosystem is ElGamal. ElGamal is based on the discrete logarithm problem discussed in Chapter 9.

Topics discussed in this section:

10.42

10.4.2 Procedure

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



10.43

10.4.2 Continued

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 $\mathbf{G} = <\mathbf{Z}_p^*, \times>$ such that $1 \le d \le p - 2$
Select e_1 to be a primitive root in the group $\mathbf{G} = <\mathbf{Z}_p^*, \times>$
$e_2 \leftarrow e_1^d \bmod p$
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
}

10.44

10.4.2 Continued

Algorithm 10.10 *ElGamal encryption*

ElGamal_Encryption $(e_1, e_2, p, \mathrm{P})$ // P is the plaintext
{
Select a random integer r in the group $\mathbf{G} = <\mathbf{Z}_p^*, \times>$
$C_1 \leftarrow e_1^r \bmod p$
$C_2 \leftarrow (P \times e_2^r) \bmod p$ // C_1 and C_2 are the ciphertexts
return C_1 and C_2
}

10.45

10.4.2 Continued

Algorithm 10.11 *ElGamal decryption*

```
ElGamal_Decryption (d, p, C1, C2)              // C1 and C2 are the ciphertexts
{
    P ← [C2 (C1^d)^-1] mod p                   // P is the plaintext
    return P
}
```

Note

The bit-operation complexity of encryption or decryption in ElGamal cryptosystem is polynomial.

10.46

Proof of ElGamal Cryptosystem

$$[C_2 \times (C_1{}^d)^{-1}] \bmod p$$
$$= [(e_2{}^r \times P) \times (e_1{}^{rd})^{-1}] \bmod p$$
$$= (e_1{}^{rd}) \times P \times (e_1{}^{rd})^{-1} = P$$

10.47

10.4.3 Continued

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 C1 and C2 for the plaintext 7.

Plaintext: 7
$C_1 = e_1{}^r \bmod 11 = 16 \bmod 11 = 5 \bmod 11$
$C_2 = (P \times e_2{}^r) \bmod 11 = (7 \times 4096) \bmod 11 = 6 \bmod 11$
Ciphertext: (5, 6)

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

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

10.48

10.4.3 Continued

Example 10. 11

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

Note

For the ElGamal cryptosystem, *p* must be at least 300 digits and *r* must be new for each encipherment.

10.49

10.4.3 Continued

Example 10. 12

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:

$p =$	115348992725616762449253137170143317404900945326098349598143469219 056898698622645932129754737871895144368891765264730936159299937280 61165964347353440008577
$e_1 =$	2
$d =$	1007
$e_2 =$	978864130430091895087668569380977390438800628873376876100220622332 554507074156189212318317704610141673360150884132940857248537703158 2066010072558707455

10.50

10.4.3 Continued

Example 10. 10

Alice has the plaintext P = 3200 to send to Bob. She chooses r = 545131, calculates C1 and C2, and sends them to Bob.

P =	3200
$r =$	545131
$C_1 =$	887297069383528471022570471492275663120260067256562125018188351429 417223599712681114105363661705173051581533189165400973736355080295 736788569060619152881
$C_2 =$	708454333048929944577016012380794999567436021836192446961774506921 244696155165800779455593080345889614402408599525919579209721628879 6813505827795664302950

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

P =	3200

10.51

- http://www-fs.informatik.uni-tuebingen.de/~reinhard/krypto/English/4.2.en.html

10.52

10-5 ELLIPTIC CURVE CRYPTOSYSTEMS

- Although RSA and ElGamal are secure asymmetric-key cryptosystems, they 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
- newer, but not as well analyzed

10.53

Finite Elliptic Curves

- ECC is an approach to public key cryptography based on the algebraic structure of elliptic curves over finite fields.
- Its security is based on the possibility of efficient additive exponentiation and absence of efficient (classical) algorithms for additive logarithm.
- 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

10.54

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

10.55

10.5.1 Elliptic Curves over Real Numbers

The general equation for an elliptic curve is

$$y^2 + b_1xy + b_2y = x^3 + a_1x^2 + a_2x + a_3$$

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

$$y^2 = x^3 + ax + b$$ where 4a^3 + 27b^2!=0

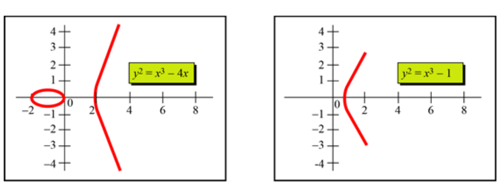
The left-hand side has a degree of 2 while the right-hand side has a degree of 3. This means that a horizontal line can intersects the curve in three points if all roots are real. However, a vertical line can intersects the curve at most in two points.

10.56

Example 10. 13

Figure 10.12 shows two elliptic curves with equations $y^2 = x^3 - 4x$ and $y^2 = x^3 - 1$. 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*



10.57

Elliptic Curves over Real Numbers

- An Abelian (commutative) Group
  - All points on an elliptic curve. A tuple P(x1, y1) represents a point on the curve if x1 and y1 are coordinates of a point on the curve that satisfy the equation of the curve.
  - For example, the points P(2, 0), Q(0, 0), R(-2, 0), S(10, 30.98) are all points on the curve $y^2 = x^3 - 4x$
  - Each point is represented by two real number.

10.58

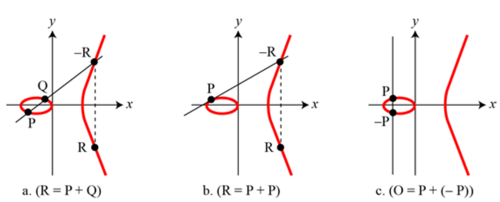
Elliptic Curves over Real Numbers

- Set
  - We define the set as the points on the curve, where each point is a pair of real numbers
  - E={(2, 0), (0, 0), (-2, 0), (10, 30.98) (10, -30.98)}
- Operation
  - We can define an addition operation on the points of the curve. Addition operation is different from the integer addition.

10.59

10.5.1 Continued

Figure 10.13 *Three adding cases in an elliptic curve*



10.60

10.5.1 Continued

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

1.

$$\lambda = (y_2 - y_1) / (x_2 - x_1)$$

$$x_3 = \lambda^2 - x_1 - x_2 \qquad y_3 = \lambda (x_1 - x_3) - y_1$$

2.

$$\lambda = (3x_1^2 + a)/(2y_1)$$

$$x_3 = \lambda^2 - x_1 - x_2 \qquad y_3 = \lambda (x_1 - x_3) - y_1$$

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.

10.61

10.5.2 Elliptic Curves over GF(p)

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). Because 2+11 mod 13 = 0

Finding Points on the Curve

Algorithm 10.12 shows the pseudocode for finding the points on the curve Ep(a, b).

10.62

10.5.2 Continued

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 Z_p) output (x, √w) (x, − √w)
      x ← x + 1
   {
}
```

10.63

Example 10. 14

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

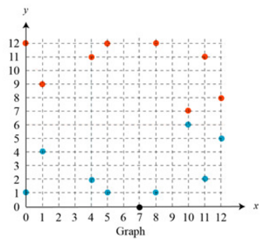
Figure 10.14 *Points on an elliptic curve over GF(p) where p is 13*

x1 = x2 and **y1+y2 mod p = 0**

P	−P
(0, 1)	(0, 12)
(1, 4)	(1, 9)
(4, 2)	(4, 11)
(5, 1)	(5, 12)
(7, 0)	(7, 0)
(8, 1)	(8, 12)
(10, 6)	(10, 7)
(11, 2)	(11, 11)
(12, 5)	(12, 18)

P = −P → (7, 0)

Points



10.64

10.5.4 ECC Simulating ElGamal

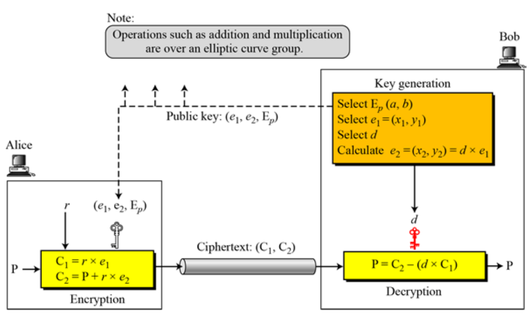


Figure 10.16 *ElGamal cryptosystem using the elliptic curve*

10.65

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 $\quad C_1 = r \times e_1 \qquad C_2 = P + r \times e_2$

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.

10.66

10.5.4 Continued

- The P calculated by Bob is the same as that intended by Alice.

$$P = C2 - (d \times C1)$$
$$= P + r \times e2 - (d \times r \times e1)$$
$$= P + (r \times d \times e1) - (r \times d \times e1)$$
$$= P + O$$

Known: $e2 = d \times e1$

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

10.67

10.5.4 Continued

Example 10. 19

Here is a very trivial example of encipherment using an elliptic curve over GF(p).

1. *Bob selects $E_{67}(2, 3)$ as the elliptic curve over GF(p).*
2. *Bob selects $e_1 = (35, 1)$ and $d = 4$.*
3. *Bob calculates $e_2 = (23, 25)$, 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 $P = (25, 0)$ to Bob. She selects $r = 2$.*

10.68

- http://www-fs.informatik.uni-tuebingen.de/~reinhard/krypto/English/4.3.en.html
- http://www-fs.informatik.uni-tuebingen.de/~reinhard/krypto/English/

10.69

ECC Security

- relies on elliptic curve logarithm problem
- 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

10.70

ECC Security

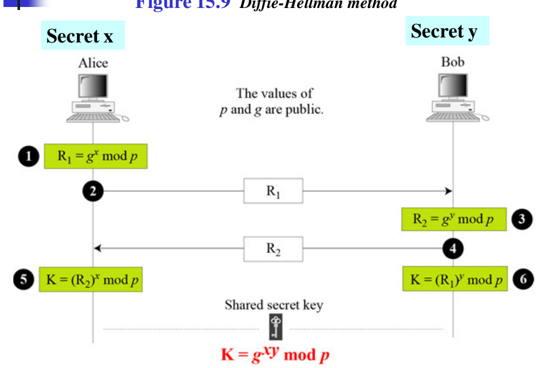
RSA can be broken with an integer factorization algorithm that scales as $\exp\left(1.923\sqrt[3]{n \log^2 n}\right)$

To break ECC, the best known classical algorithm requires $O\left(2^{n/2}\right)$ search.

10.71

Diffie-Hellman Key Agreement

Figure 15.9 ***Diffie-Hellman method***



10.72

Continued

Note

The symmetric (shared) key in the Diffie-Hellman method is K = g^{xy} mod p.

10.73

Continued

Example 15.1

Let us give a trivial example to make the procedure clear. Our example uses small numbers, but note that in a real situation, the numbers are very large. Assume that g = 7 and p = 23. The steps are as follows:

1. **Alice chooses x = 3 and calculates R_1 = 73 mod 23 = 21.**
2. **Bob chooses y = 6 and calculates R_2 = 76 mod 23 = 4.**
3. **Alice sends the number 21 to Bob.**
4. **Bob sends the number 4 to Alice.**
5. **Alice calculates the symmetric key K = 43 mod 23 = 18.**
6. **Bob calculates the symmetric key K = 216 mod 23 = 18.**
7. **The value of K is the same for both Alice and Bob; g^{xy} mod p = 718 mod 35 = 18.**

10.74

Continued

Example 15.2

Let us give a more realistic example. We used a program to create a random integer of 512 bits (the ideal is 1024 bits). The integer p is a 159-digit number. We also choose g, x, and y as shown below:

p	764624298563493572182493765955030507476338096726949748923573772860925 235666660755423637423309661180033338106194730130950414738700999178043 6548785807987581
g	2
x	557
y	273

10.75

Continued

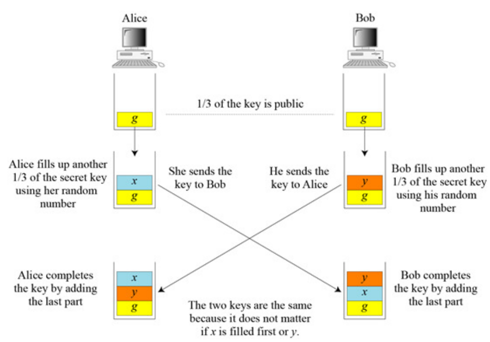
Example 15.2 *Continued*

The following shows the values of R_1, R_2, and K.

R_1	844920284205665505216172947491035094143433698520012660862863631067673 619959280828586700802131859290945140217500319973312945836083821943065 966020157955354
R_2	435262838709200379470747114895581627636389116262115557975123379218566 310011435718208390040181876486841753831165342691630263421106721508589 6255201288594143
K	155638000664522290596225827523270765273218046944423678520320400146406 500887936651204257426776608327911017153038674561252213151610976584200 1204086433617740

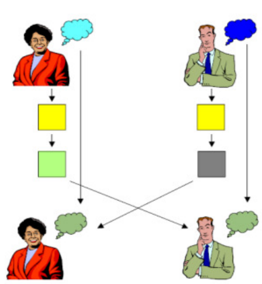
Continued

Figure 15.10 *Diffie-Hellman idea*



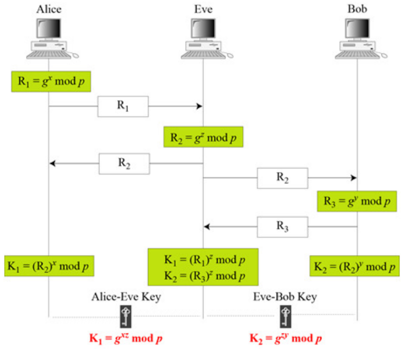
Another Analog

- Alice & Bob each think of a secret color (known only to them)
- They mix their color with yellow (agreed upon openly ahead of time) and exchange.
- They mix their color with what they've received.
- Both have the same color but observer cannot duplicate.



Man-in-the-Middle Attack

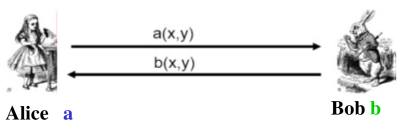
Figure 15.11 ***Man-in-the-middle attack***



10.79

ECC Diffie-Hellman

- **Public:** Elliptic curve and point B=(x,y) on curve
- **Secret:** Alice's a and Bob's b



- Alice computes shared key a(b(x,y))
- Bob computes shared key b(a(x,y))
- These are the same since ab = ba

10.80

Example – Elliptic Curve Diffie-Hellman Exchange

- Alice and Bob want to agree on a shared key.
  - Alice and Bob compute their public and private keys.
    - Alice
      - » Private Key = a
      - » Public Key = PA = a * B
    - Bob
      - » Private Key = b
      - » Public Key = PB = b * B
  - Alice and Bob send each other their public keys.
  - Both take the product of their private key and the other user's
  public key.
      - Alice → K_{AB} = a(bB)
      - Bob → K_{AB} = b(aB)
      - **Shared Secret Key = K_{AB} = abB**

10.81

Why use ECC?

- How do we analyze Cryptosystems?
  - How difficult is the underlying problem that it is based upon
    - RSA – Integer Factorization
    - DH – Discrete Logarithms
    - ECC - Elliptic Curve Discrete Logarithm problem
  - How do we measure difficulty?
- We examine the algorithms used to solve these problems

10.82

Security of ECC

- To **protect** a 128 bit AES key it would take a:
  - RSA Key Size: 3072 bits
  - ECC Key Size: 256 bits
- How do we strengthen RSA?
  - Increase the key length
- **Impractical?**

NIST guidelines for public key sizes for AES

ECC KEY SIZE (Bits)	RSA KEY SIZE (Bits)	KEY SIZE RATIO	AES KEY SIZE (Bits)
163	1024	1 : 6	
256	3072	1 : 12	128
384	7680	1 : 20	192
512	15 360	1 : 30	256

Supplied by NIST to ANSI X9F1

10.83

Comparable Key Sizes for Equivalent Security

Security (bits)	ECC-based scheme (size of n in bits)	RSA/DSA (modulus size in bits)
56	112	512
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

10.84

Applications of ECC

- Many devices are small and have limited storage and computational power
- Where can we apply ECC?
  - **Wireless communication devices**
  - Smart cards
  - Web servers that need to handle many encryption sessions
  - **Any application where security is needed but lacks the power, storage and computational power that is necessary for our current cryptosystems**

10.85

Benefits of ECC

- Same benefits of the other cryptosystems: confidentiality, integrity, authentication and non-repudiation but…
- Shorter key lengths
  - Encryption, Decryption and Signature Verification speed up
  - Storage and bandwidth savings

10.86

Summary of ECC

- "**Hard problem**" analogous to discrete log
  - **$Q=kP$, where Q,P belong to a prime curve**
    given k,P → "easy" to compute Q
    given Q,P → "hard" to find k
  - **known as the elliptic curve logarithm problem**
    - **k must be large enough**
- ECC security relies on elliptic curve logarithm problem
  - compared to factoring, can use much smaller key sizes than with RSA etc

→ for similar security ECC offers significant computational advantages

10.87