Cryptography and Network Security Chapter 10

Fifth Edition
by William Stallings
Lecture slides by Lawrie Brown
(with edits by RHB)

Chapter 10 – Other Public Key Cryptosystems

Amongst the tribes of Central Australia every man, woman, and child has a secret or sacred name which is bestowed by the older men upon him or her soon after birth, and which is known to none but the fully initiated members of the group. This secret name is never mentioned except upon the most solemn occasions; to utter it in the hearing of men of another group would be a most serious breach of tribal custom. When mentioned at all, the name is spoken only in a whisper, and not until the most elaborate precautions have been taken that it shall be heard by no one but members of the group. The native thinks that a stranger knowing his secret name would have special power to work him ill by means of magic.

—*The Golden Bough,* Sir James George Frazer

Outline

- will consider:
  - Diffie-Hellman key exchange
  - ElGamal cryptography
  - Elliptic Curve cryptography
  - Pseudorandom Number Generation (PRNG) based on Asymmetric Ciphers (RSA & ECC)

Diffie-Hellman Key Exchange

- first public-key type scheme proposed
- by Diffie & Hellman in 1976 along with the exposition of public key concepts
  - note: now know that Williamson (UK CESG) secretly proposed the concept in 1970
- is a practical method for public exchange (really **creation**) of a secret key
- used in a number of commercial products

Diffie-Hellman Key Exchange

- a public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key
  - known only to the two participants
- value of key depends on the participants (and their private and public key information)
- based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) – easy
- security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard

Diffie-Hellman Setup

- all users agree on global parameters:
  - large prime integer or polynomial q
  - a being a primitive root mod q
- each user (eg. A) generates their key
  - chooses a secret key (number): $x_A < q$
  - compute their **public key**: $y_A = a^{x_A} \bmod q$
- each user makes public that key y_A

Diffie-Hellman Key Exchange

- shared session key for users A & B is K_{AB}:

  $K_{AB} = a^{x_A \cdot x_B} \bmod q$

  $= y_A^{x_B} \bmod q$ (which **B** can compute)

  $= y_B^{x_A} \bmod q$ (which **A** can compute)
- K_{AB} is used as session key in private-key encryption scheme between Alice and Bob
- if Alice and Bob subsequently communicate, they will have the **same** key as before, unless they choose new public-keys
- attacker needs an x, must solve discrete log

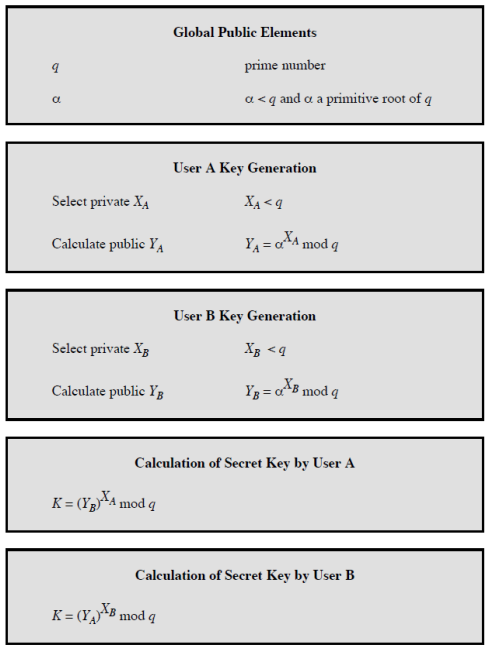


Figure 10.1 The Diffie-Hellman Key Exchange Algorithm

Diffie-Hellman Example

- users Alice & Bob who wish to swap keys:
- agree on prime $q = 353$ and $a = 3$
- select random secret keys:
  - A chooses $x_A = 97$, B chooses $x_B = 233$
- compute respective public keys:
  - $y_A = 3^{97} \bmod 353 = 40$ (Alice)
  - $y_B = 3^{233} \bmod 353 = 248$ (Bob)
- compute shared session key as:
  - $K_{AB} = y_B^{x_A} \bmod 353 = 248^{97} = 160$ (Alice)
  - $K_{AB} = y_A^{x_B} \bmod 353 = 40^{233} = 160$ (Bob)

Key Exchange Protocols

- users could create random private/public D-H keys each time they communicate
- users could create a known private/public D-H key and publish public in a directory; this is then consulted and used to securely communicate with them
- both of these are vulnerable to a Man-in-the-Middle Attack
- (so authentication of the keys is needed)

Man-in-the-Middle Attack

1. Darth prepares by creating two private / public keys
2. Alice transmits her public key to Bob
3. Darth intercepts this and transmits his first public key to Bob. Darth also calculates a shared key with Alice
4. Bob receives the public key and calculates the shared key (with Darth instead of Alice)
5. Bob transmits his public key to Alice
6. Darth intercepts this and transmits his second public key to Alice. Darth calculates a shared key with Bob
7. Alice receives the key and calculates the shared key (with Darth instead of Bob)

Darth can then intercept, decrypt, re-encrypt, and forward all messages between Alice & Bob

ElGamal Cryptography

- public-key cryptosystem related to D-H
- uses exponentiation in a finite (Galois) field
- with security based difficulty of computing discrete logarithms, as in D-H
- each user (eg. A) generates their key
  - chooses a secret key (number): $1 < x_A < q - 1$
  - computes their **public key**: $y_A = a^{x_A} \bmod q$

ElGamal Message Exchange

- Bob encrypt a message to send to Alice
  - Bob represents message M in range $0 <= M <= q - 1$
    - longer messages must be sent as blocks
  - Bob chooses random integer k with $1 <= k <= q - 1$
  - Bob computes one-time key $K = y_A^k \bmod q$
  - Bob encrypts M as a pair of integers (C_1, C_2) where
    - $C_1 = a^k \bmod q$ and $C_2 = KM \bmod q$
- Alice then recovers message by
  - recovering key K as $K = C_1^{xA} \bmod q$ (cf. D-H)
  - computing M as $M = C_2 K^{-1} \bmod q$
- a unique secret k must be used each time
  - otherwise result is insecure

Global Public Elements

q	prime number
α	$\alpha < q$ and α a primitive root of q

Key Generation by Alice

Select private X_A	$X_A < q - 1$
Calculate Y_A	$Y_A = \alpha^{X_A} \bmod q$
Public key	$PU = \{q, \alpha, Y_A\}$
Private key	X_A

Encryption by Bob with Alice's Public Key

Plaintext:	$M < q$
Select random integer k	$k < q$
Calculate K	$K = (Y_A)^k \bmod q$
Calculate C_1	$C_1 = \alpha^k \bmod q$
Calculate C_2	$C_2 = KM \bmod q$
Ciphertext:	(C_1, C_2)

Decryption by Alice with Alice's Private Key

Ciphertext:	(C_1, C_2)
Calculate K	$K = (C_1)^{X_A} \bmod q$
Plaintext:	$M = (C_2 K^{-1}) \bmod q$

Figure 10.3 The ElGamal Cryptosystem

ElGamal Example

- use field GF(19) $q = 19$ and $a = 10$
- Alice computes her key:
  - chooses $x_A = 5$; computes $y_A = 10^5 \bmod 19 = 3$
- Bob send message $m = 17$ as $(11, 5)$ by
  - choosing random $k = 6$
  - computing $K = y_A^k \bmod q = 3^6 \bmod 19 = 7$
  - computing $C_1 = a^k \bmod q = 10^6 \bmod 19 = 11$;
    $C_2 = KM \bmod q = 7.17 \bmod 19 = 5$
- Alice recovers original message by computing:
  - recover $K = C_1^{xA} \bmod q = 11^5 \bmod 19 = 7$
  - compute inverse $K^{-1} = 7^{-1} = 11$
  - recover $M = C_2 K^{-1} \bmod q = 5.11 \bmod 19 = 17$

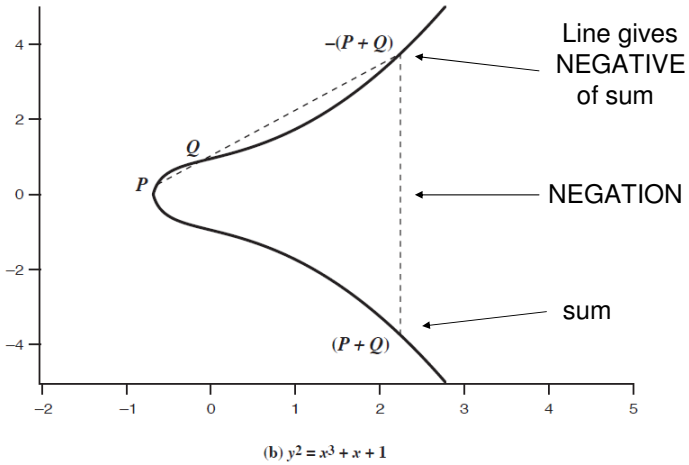
Elliptic Curve Cryptography

- majority of public-key crypto (RSA, D-H) 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 well accepted these days

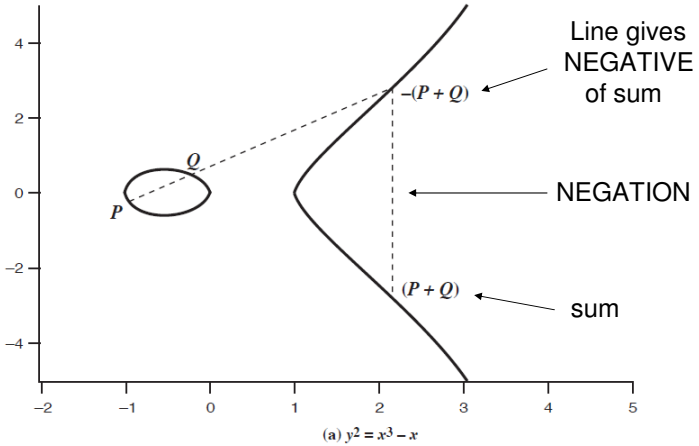
Real Elliptic Curves

- an elliptic curve is defined by an equation in two variables x and y, with real coefficients
- consider a cubic elliptic curve of form
  - $y^2 = x^3 + ax + b$
  - where x, y, a, b are all **real** numbers
  - also define zero point O
- consider set of points $E(a,b)$ that satisfy
- have addition operation for elliptic curve
  - geometrically sum of $P + Q$ is **reflection** of the intersection R

Real Elliptic Curve Example



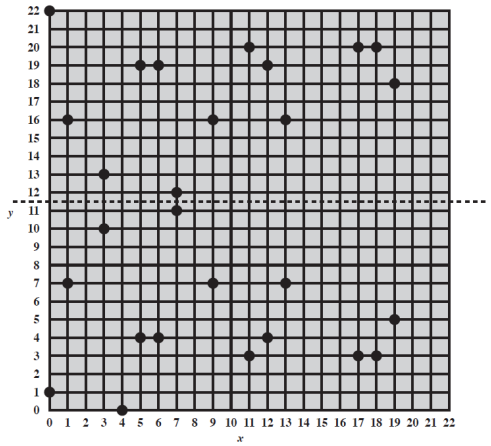
Real Elliptic Curve Example



Finite Elliptic Curves

- all the arithmetic in elliptic curves is **rational** (i.e. +,-,x,/), so it works in ANY field; eg. GF(…)
- elliptic curve cryptography uses curves whose variables and coefficients are finite field elements
- two families are commonly used:
  - prime curves $E_p(a,b)$ defined over Z_p
    - use integers modulo a prime p
    - best in software
  - binary curves $E_{2^m}(a,b)$ defined over $GF(2^m)$
    - use polynomials with binary coefficients
    - best in hardware

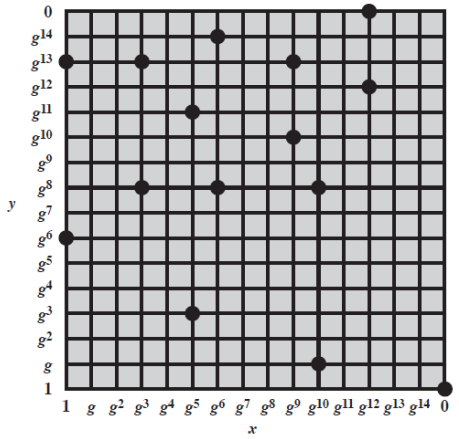
$E_{23}(1,1)$



Points on $E_{23}(1,1)$

(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)

$E_{2^4}(g^4,1)$



Points on $E_{2^4}(g^4,1)$

$(0, 1)$	(g^5, g^3)	(g^9, g^{13})
$(1, g^6)$	(g^5, g^{11})	(g^{10}, g)
$(1, g^{13})$	(g^6, g^8)	(g^{10}, g^8)
(g^3, g^8)	(g^6, g^{14})	$(g^{12}, 0)$
(g^3, g^{13})	(g^9, g^{10})	(g^{12}, g^{12})

Elliptic Curve Cryptography

- ECC **addition** is analog of **modulo multiply**
- ECC repeated addition is analog of modulo exponentiation
- need "hard" problem equiv to discrete log
  - $Q = kP$, where Q, P belong to a prime curve
  - is "easy" to compute Q given k, P
  - but "hard" to find k given Q, P
  - known as the elliptic curve logarithm problem
- Certicom example: $E_{23}(9, 17)$

ECC Diffie-Hellman

- can do key exchange analogous to D-H
- users select a suitable curve $E_q(a, b)$
- select base point $G = (x_1, y_1)$
  - with large order n s.t. $nG = O$
- A & B select private keys $n_A < n$, $n_B < n$
- compute public keys: $P_A = n_A G$, $P_B = n_B G$
- compute shared key: $K = n_A P_B$, $K = n_B P_A$
  - same since $K = n_A n_B G$
- attacker would need to find K, hard

Global Public Elements

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

User A Key Generation

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

User B Key Generation

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

Calculation of Secret Key by User A

$K = n_A \times P_B$

Calculation of Secret Key by User B

$K = n_B \times P_A$

Figure 10.7 ECC Diffie-Hellman Key Exchange

ECC Encryption/Decryption

- several alternatives, simplest is like ElGamal
- must first encode any message M as a point on the elliptic curve P_m
- select suitable curve and point G as in D-H
- receiver chooses private key $n_A < n$
- receiver computes public key $P_A = n_A G$
- sender chooses private random key k
- sender encrypts P_m : $C_m = \{kG, P_m + kP_b\}$
- decrypt C_m compute:

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

ECC Security

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

Comparable Key Sizes for Equivalent Security

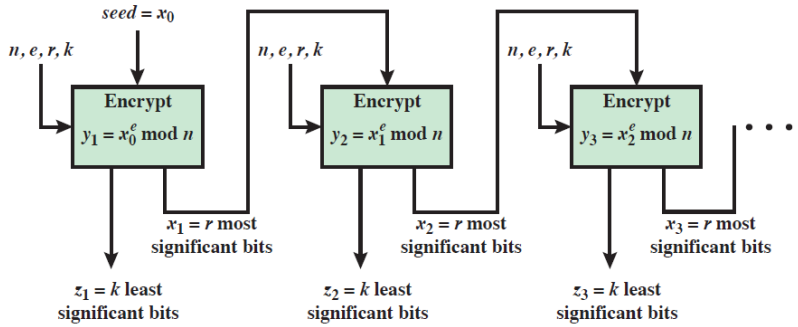
Symmetric scheme (key size in bits)	ECC-based scheme (size of n in bits)	RSA/DSA (modulus size in bits)
56	112	512
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

Pseudorandom Number Generation (PRNG) based on Asymmetric Ciphers

- asymmetric encryption algorithm produce apparently random output (like any crypto)
- hence can be used to build a pseudorandom number generator (PRNG)
- much slower than symmetric algorithms
- hence only use to generate a short pseudorandom bit sequence (eg. key)

PRNG based on RSA

- have Micali-Schnorr PRNG using RSA
  - in ANSI X9.82 and ISO 18031



PRNG based on ECC

- dual elliptic curve PRNG
  - NIST SP 800-9, ANSI X9.82 and ISO 18031
- algorithm

  s_0 = random{0...#E(GF(p))-1}
  for i = 1 to k do
  set s_i = x_coord_of(s_{i-1}P)
  set r_i = lsb_{240}(x_coord_of(s_iQ))
  end for
  return r_1 , . . . , r_k
- would only use if only ECC available
- controversy on security/inefficiency
- an example of an algorithm, known by NSA to be weak, deliberately promoted by NSA (so they can easily listen)