DI-FCT-UNL
Computer and Network Systems Security
Segurança de Redes e Sistemas de Computadores

Public Key Cryptohgraphy

Annex
(Complem./Optional Topics)

Outline (Complementary Material)

- **Asymmetric cryptography**

Base Material

- Public-Key cryptography principles
- Public-Key algorithms
- Public Key Signatures
- RSA algorithm
  - Key-Pair Generation and Encryption/Decryption
- Diffie-Hellman key exchange
- Key-distribution with asymmetric cryptography

- Annex (complementary / optional topics):
  - RSA
  - ECC Foundations

RSA

RSA Use

- to encrypt a message M the sender:
  - obtains **public key** of recipient $PU=\{e,n\}$
  - computes: $C = M^e \bmod n$, where $0 \leq M < n$
- to decrypt the ciphertext C the owner:
  - uses their private key $PR=\{d,n\}$
  - computes: $M = C^d \bmod n$
- note that the message M must be smaller than the modulus n (block if needed)

Why RSA Works

- because of Euler's Theorem:
  - $a^{\phi(n)} \bmod n = 1$ where $\gcd(a,n)=1$
- in RSA have:
  - $n=p.q$
  - $\phi(n)=(p-1)(q-1)$
  - carefully chose e & d to be inverses $\bmod \phi(n)$
  - hence $e.d=1+k.\phi(n)$ for some k
- hence :

$$C^d = M^{e.d} = M^{1+k.\phi(n)} = M^1.(M^{\phi(n)})^k$$
$$= M^1.(1)^k = M^1 = M \bmod n$$

Operations with big integers (large numbers)

- Optimization strategy
  - Elementary addition and subtraction
    - Multiple-precision addition or subtraction of large numbers are $O(n)$
      - n the number of bits of operands
  - Modular addition and subtraction is $O(n)$
    - (x + y) mod N

      = x + y, if x + y < N

      = x + y – m , if x + y >= N
  - Large number multiplication
    - Pencil and paper method algorithm: $O(n^2)$
    - Russian Peasant Multiplication Method
      - Good for binary representations
      - Multiplication as a series of additions and shifts
      - Variable complexity: from $O(n)$ to $O(n^2)$

Modular multiplication

- Requires the computation of x.y mod N
  - First must compute x.y
  - Followed by a reduction of the result modulo M
    - Relates with a division and computation of large dimension intermediate products
- Possible optimized reductions:
  - Barret modular reduction
- Montgomery's multiplication
- Multiplication by squaring

Multiplication by squaring

$$a.b \bmod n = \{ [(a+b)^2 - a^2 - b^2] / 2 \} \bmod n$$

$$a.b \bmod n = \{ [(a+b)^2 - (a - b)^2] / 4 \} \bmod n$$

Advantage:

Can also benefit from fast calculations performed on a cryptographic coprocessor

Exponentiation can be simple

- Can use the Square and Multiply Algorithm
- A fast, efficient algorithm for exponentiation
- Conceptually is based on repeatedly squaring base and multiplying in the ones that are needed to compute the result
- Look at binary representation of exponent
- Only takes $O(\log_2 n)$ multiples for number n
  - eg. $7^5 = 7^4.7^1 = 3.7 = 10 \bmod 11$
  - eg. $3^{129} = 3^{128}.3^1 = 5.3 = 4 \bmod 11$

Exponentiation algorithm

```
c = 0; f = 1
for i = k downto 0
    do c = 2 x c
       f = (f x f) mod n
    if b_i == 1 then
       c = c + 1
       f = (f x a) mod n
 return f
```

Efficient Encryption

- Encryption uses exponentiation to power e
- Hence if e small, this will be faster
  - often choose e=65537 (2^{16}-1)
  - also see choices of e=3 or e=17
- But if e too small (eg e=3) can attack
  - using CRT (Chinese Remainder Theorem) & 3 messages with different modulii
- If e fixed must ensure $\gcd(e,\phi(n))=1$
  - ie reject any p or q not relatively prime to e
  - Need to generate those two primes with these property
    - First generate primes (randomly + primality test)
    - Check if gcd (e, ø(n)) = 1.
    - If yes, p and q are ok, if not, generate other pair (p,q)

Efficient Decryption

- Decryption uses exponentiation to power d
  - this is likely large, insecure if not
- Can use the Chinese Remainder Theorem (CRT) to compute mod p & q separately. then combine to get desired answer
  - approx 4 times faster than doing directly

- The key idea: only owner of private key who knows values of p & q can use this technique

CRT – Chinese Reminder Theorem (1)

- Can decrease the processing time involving private keys by a factor of ~4.

- If the integers $n_1, n_2, n_3, \ldots n_k$ are pairwise relatively prime, then the system of simultaneous congruences

$$x \equiv a_1 \bmod n_1$$
$$x \equiv a_2 \bmod n_2$$
$$\ldots$$
$$x \equiv a_k \bmod n_k$$

Has an unique solution x, for 0 <= x <= n
with $n = n_1 n_2 n_3 \ldots n_k$

CRT – Chinese Reminder Theorem (2)

Solution for x is computable in the following way:

$$x \equiv \sum_{i=1}^{k} a_i N_i \; N'_i$$

$$n = n_1 \, n_2 \, n_3 \, \ldots \, n_k$$

$$N_i = n/n_i \qquad N'_I = N_i^{-1} \pmod{n_i}$$

$$i = 1,2, \ldots k$$

CRT (practical example - 1)

$M=c^d \bmod n$

$p=7,\ q=11,\ e=19,\ d = e^{-1} \bmod (p-1)(q-1) = 19$

Precomputing:

$dP = e^{-1} \bmod (p-1) = d \bmod (p-1) = 19 \bmod 6 = 1$

$dQ = e^{-1} \bmod (q-1) = d \bmod (q-1) = 19 \bmod 10 = 9$

$qInv = q^{-1} \bmod p = 11^{-1} \bmod 7 = 2$

Then, storing the quintuple (p. q. dP, dQ, qInv) (as a representation of the private key)

Note) we can also try n = p x q x r x t

CRT (practical example - 2)

Then, to compute

$s=m^d \bmod n = m^d \bmod (p.q)$

We can compute (Garner's Algorithm)

$s1 = m^{dP} \bmod p = 50^1 \bmod 7 = 1$

$s2 = m^{dQ} \bmod q = 50^9 \bmod 11 = 2$

$h = qInv\ (s1-s2) \bmod p = 2(1-2) \bmod 7 = 5$

$s = s2 + hq = 2+5(11) = 57$

Similar to:

$s=m^d \bmod pq = 50^{19} \bmod 77 = 57$

RSA Key Generation

- Users of RSA must:
  - determine two primes at random - `p, q`
  - Primality test
  - select either `e` or `d` and compute the other
- Primes `p,q` must not be easily derived from modulus `n=p.q`
  - means must be sufficiently large
  - Difficult for factorization
  - typically guess and use probabilistic test
    - (ex., Probabilistic Rabin-Miller)
- exponents `e, d` are inverses, so use Inverse algorithm to compute the other
  - Euclid's Inverse Algorithm

RSA Security

- Possible approaches to attacking RSA are:
  - Brute force key search (infeasible given size of numbers)
  - Mathematical attacks (based on difficulty of computing ø(n), by factoring modulus n)
  - Timing attacks (on running of decryption)
  - Chosen ciphertext attacks (given properties of RSA)

Factoring Problem

- mathematical approach takes 3 forms:
  - factor $n=p.q$, hence compute $\varnothing(n)$ and then d
  - determine $\varnothing(n)$ directly and compute d
  - find d directly
- currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - as of May-05 best is 200 decimal digits (663 bit) with LS (LATTICE SIEVE)
  - biggest improvement comes from improved algorithm
    - cf QS to GNFS to LS
  - currently assume 1024-2048 bit RSA is secure
    - ensure p, q of similar size and matching other constraints
    - But observations and studies evolve, considering also that computers will continue to get faster

Timing Attacks

- Developed by Paul Kocher in mid-1990's
  - Applicable to any public-key crypto system
  - Ciphertext only attack
- Exploit timing variations in operations
  - eg. multiplying by small vs large number
  - or IF's varying which instructions executed
- Infer operand size based on "time taken"
- RSA exploits time taken in exponentiation
- Countermeasures
  - Use constant exponentiation time
  - Add random delays
  - Blind values used in calculations
  - Secure message padding can also help

Chosen Ciphertext Attacks

RSA is vulnerable to a Chosen Ciphertext Attack (CCA)

Attackers chooses ciphertexts & gets decrypted plaintext back

Choose ciphertext to exploit properties of RSA to provide info to help cryptanalysis

Can counter with random pad (important) of plaintext or use Optimal Asymmetric Encryption Padding (OAEP)

DSA (or DSS)

DSA Security (see the DSA Alg.)

- With DSA, the entropy, secrecy, and uniqueness of the random signature value *k* are critical.
- Violating any one of those three requirements can reveal the entire private key to an attacker
- Using the same value twice (even while keeping *k* secret), using a predictable value, or leaking even a few bits of *k* in each of several signatures, is enough to reveal the private key *x*
- *Practical issue (ex., RFC 6976):*

  $K = H(M \,||\, Kpriv)$

DSA Computations

- Modular Exponentiations: use of exponentiation by squaring

- Signature constructions and verifications:
  - For the involved exponentiations, can use the extended Euclidean Alg. or the Fermat Little Theorem

ECC

ECC: Why ?

Why ECC ? What's Wrong with RSA ?

- RSA is based upon the 'belief' that factoring is 'difficult' – never been proven
- Prime numbers are getting too large (more and more large keys for security …. Slow ?)
- Amount of research currently devoted to factoring algorithms
- … Quantum computing will make RSA obsolete overnight ?

ECC: Why ?

- ECC: Based on Properties of Elliptic Curves and Operations on Elliptic Curves

- The discrete logarithm problem on elliptic curve groups is believed to be more difficult than the corresponding problem in (the multiplicative group of nonzero elements of) the underlying finite field.

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

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

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**

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

ECC Practical Use

- Pseudo-Random Generation
- Integer Factorization Algorithms

- Key Agreement Protocols
- Digital Signatures
- Encryption/Decryption

Practical use and standardization effort

- US NIST
  - has endorsed elliptic curve cryptography in its SUITE B Algorithms Recommendation for the following use:
    - ECDH – Elliptic Curve Diffie Hellman Key Exchanges
    - ECDSA – Elliptic Curve DSA Signatures
- in August 2015, the NSA announced that it is planned to replace Suite B with a new cipher suite due to concerns about quantum computing attacks on ECC

What is an Elliptic Curve ?

What is an Elliptic Curve ?

- Let $a \in \mathbb{R}$, $b \in \mathbb{R}$, be constants such that $4a^3 + 27b^2 \neq 0$.

- *A non-singular elliptic curve (or a plane curve over a finite field)* is the set E of solutions (points x,y) $\in \mathbb{R} \times \mathbb{R}$ to the equation:

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

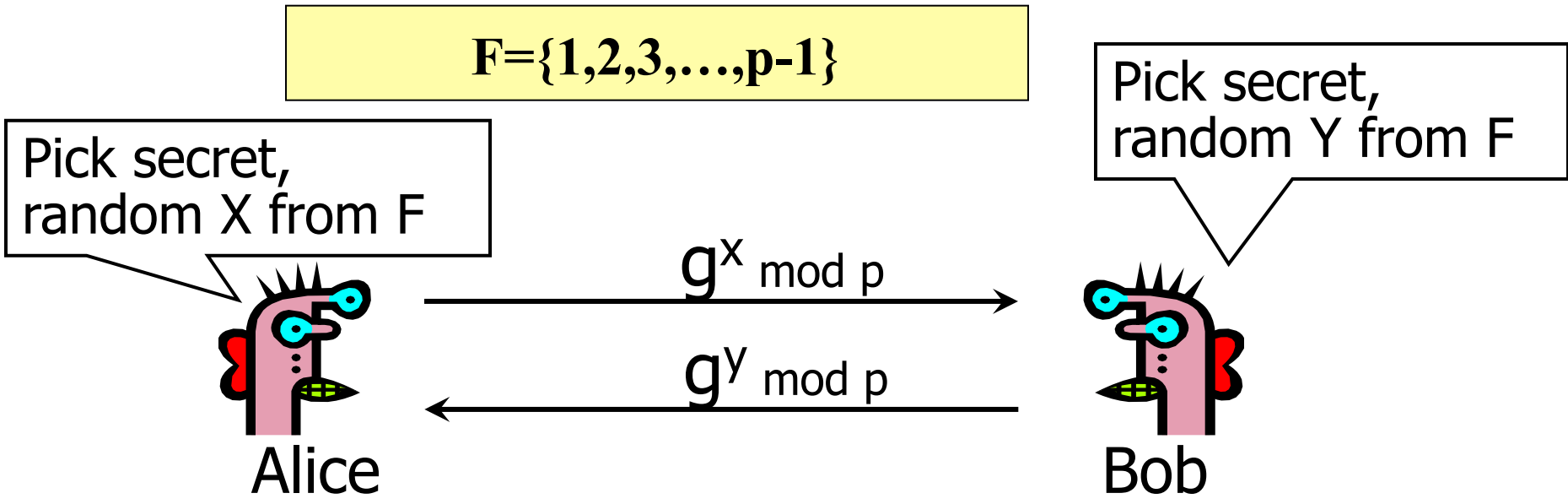
together with a special point O called the *point at infinity*.

Singular Elliptic Curve

- If $4a^3 + 27b^2 = 0$, then we have a *singular elliptic curve*. This could potentially lead to having to not having 3 distinct roots

- Therefore, we must deal with non-singular elliptic curves with the condition $4a^3 + 27b^2 \neq 0$, in order to assure that we have 3 distinct roots.
- This will allow us to establish the fact that the solution set E forms an Abelian group.

Discrete Logarithms in Finite Fields (DH)



Compute $k=(g^y)^x= g^{xy}$ mod p

Compute $k=(g^x)^y= g^{xy}$ mod p

The adversary has to compute g^{xy} from g^x and g^y
without knowing x and y…
She/He is faced to the Discrete Logarithm Problem in finite fields

What is a Group and Its Properties ?

Suppose we have any binary operation, such as addition (+), that is defined for every element in a *set G*, which is denoted *(G, +)*

- Then *G* is a *group* with respect to addition if the following conditions hold:

  1.) ***G* is *closed* under addition**: $x \in G$, $y \in G$, imply $x + y \in G$

  2.) **+ is associative**. For all $x, y, z, \in G$, $x + (y + z) = (x + y) + z$

  3.) ***G* has an identity element *e***. There is an *e* in *G* such that $x + e = e + x = x$ for all $x \in G$.

  4.) ***G* contains inverses**. For each $x \in G$, there exists $y \in G$, such that $x + y = y + x = e$.

What is an Abelian Group ?

- An Abelian group contains all the rules of a group, but also must meet the following criteria:

5.) **+ is commutative.**

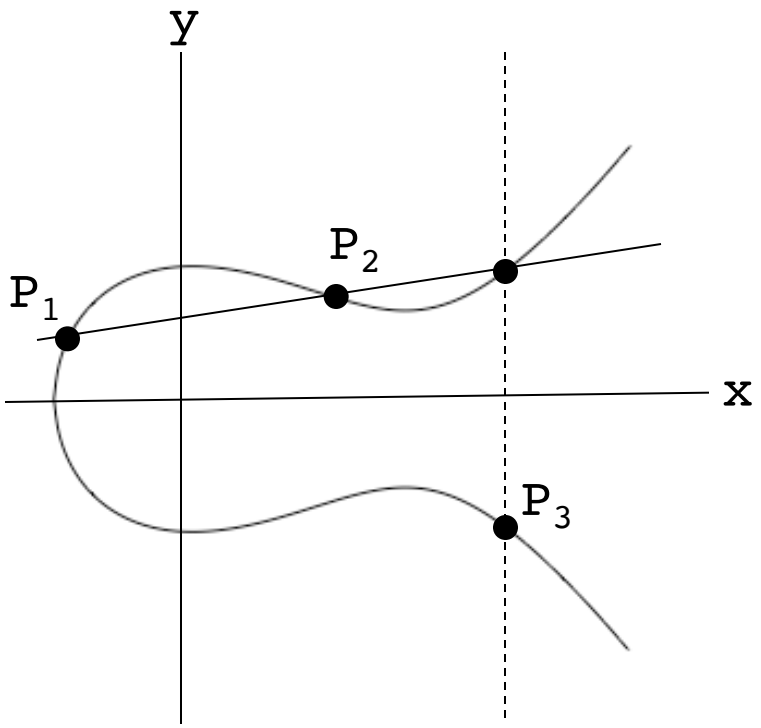
For all $x \in G, y \in G, x + y = y + x$.

Abelian Group Properties

Given two points P,Q in *E(Fp)*, there is a third point, denoted by *P+Q* on *E(Fp)*, and the following relations hold for all P,Q,R in *E(Fp)*

- $P + Q = Q + P$ (*commutativity*)
- $(P + Q) + R = P + (Q + R)$ (*associativity*)
- $P + O = O + P = P$ (*existence of an identity element*)
- there exists $(-P)$ such that $-P + P = P + (-P) = O$ (*existence of inverses*)

Elliptic Curve Representation



- Consider elliptic curve

  $E: y^2 = x^3 - x + 1$

- If P_1 and P_2 are on E, we can define

  $P_3 = P_1 + P_2$

  as shown in picture
- Addition is all we need

Solutions: 3 cases to be considered

- Suppose $P, Q \in E$, where

  $P = (x_1, y_1)$ and $Q = (x_2, y_2)$

We must consider three possible cases:

1.) $x_1 \neq x_2$

2.) $x_1 = x_2$ and $y_1 = -y_2$

3.) $x_1 = x_2$ and $y_1 = y_2$

These cases must be considered when defining "addition" for a considered solution set

Defining Addition on E : Case 2

For the case $x_1 = x_2$ and $y_1 = -y_2$, addition is defined as follows:

$(x_1,y_1) + (x_2,y_2) = (x_3,y_3) \in E$ where

$(x,y) + (x,-y) = O$, the point at infinity

Defining Addition on E: Case 1

For the case $x_1 \neq x_2$, addition is defined as follows:

$$(x_1,y_1) + (x_2,y_2) = (x_3,y_3) \in E \text{ where}$$

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1, \text{ and}$$
$$\lambda = (y_2 - y_1) / (x_2 - x_1)$$

Defining Addition on E: Case 3

For the case $x_1 = x_2$ and $y_1 = y_2$, addition is defined as follows:

$$(x_1,y_1) + (x_2,y_2) = (x_3,y_3) \in E \text{ where}$$

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1, \text{ and}$$
$$\lambda = (3x_1^2 + a) / 2y_1$$

Defining the Identity

- The point at infinity O, is the identity element. $P + O = O + P = P$, for all $P \in E$.
- From Case 2, and the Identity Element, we now have the existence of inverses
- Beyond the scope here to prove that we have commutativity and associativity as well
- Therefore the set of solutions E, forms an Abelian group (Importance of this will be shown later)

Elliptic Curves Modulo P

Let $p > 3$ be prime.

- The elliptic curve $y^2 = x^3 + ax + b$ over $\mathbb{Z}_p$ is the set of solutions $(x,y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ to the congruence:

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

where $a \in \mathbb{Z}_p$, $b \in \mathbb{Z}_p$, are constants such that $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$, together with a special point O called the *point at infinity*.

- Note: the Solutions still form an Abelian group

… For an example (P = 11)

- Let's examine the following elliptic curve as an example:

$$y^2 = x^3 + x + 6 \text{ over } \mathbb{Z}_{11}$$

X	0	1	2	3	4	5	6	7	8	9	10
$x^3 + x + 6 \bmod 11$	6	8	5	3	8	4	8	4	9	7	4
QR?	N	N	Y	Y	N	Y	N	Y	Y	N	Y
Y			4,7	5,6		2,9		2,9	3,8		2,9

Another Example (P = 5)

$y^2 = x^3 + 2x + 3 \pmod 5$

$x = 0 \Rightarrow y^2 = 3 \Rightarrow$ no solution $\pmod 5$

$x = 1 \Rightarrow y^2 = 6 = 1 \Rightarrow y = 1,4 \pmod 5$

$x = 2 \Rightarrow y^2 = 15 = 0 \Rightarrow y = 0 \pmod 5$

$x = 3 \Rightarrow y^2 = 36 = 1 \Rightarrow y = 1,4 \pmod 5$

$x = 4 \Rightarrow y^2 = 75 = 0 \Rightarrow y = 0 \pmod 5$

- Then in this case the points on the elliptic curve are

  (1,1) (1,4) (2,0) (3,1) (3,4) (4,0)

  and the point at infinity: ∞

Generating our group

- From the previous slide, and including the point at infinity *O*, we have a group with 13 points.

- Since the *O*(*E*) is prime, the group is cyclic.
- We can generate the group by choosing any point other then the point at infinity.

Let our generator (for ex.,) = α = (2,7)

The Group

- We can generate this by using the rules of addition we defined earlier where $2\alpha = \alpha + \alpha$

$\alpha = (2,7)$ $2\alpha = (5,2)$ $3\alpha = (8,3)$

$4\alpha = (10,2)$ $5\alpha = (3,6)$ $6\alpha = (7,9)$

$7\alpha = (7,2)$ $8\alpha = (3,5)$ $9\alpha = (10,9)$

$10\alpha = (8,8)$ $11\alpha = (5,9)$ $12\alpha = (2,4)$

So in suymmary

- An elliptic curve over a field ***K*** is a nonsingular cubic curve in two variables, *f(x,y) =0* with a rational point (which may be a point at infinity).

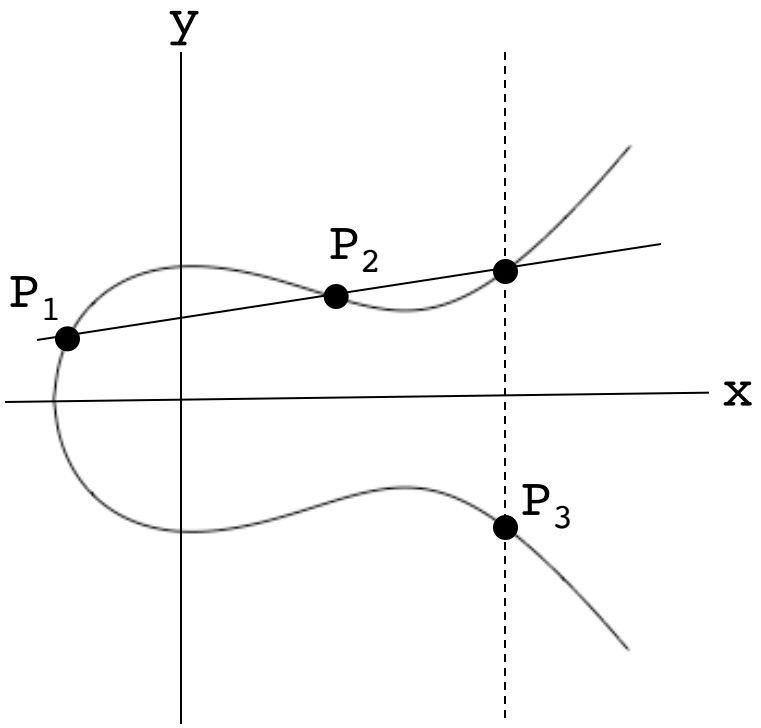
- The field ***K*** is usually taken to be the complex numbers, reals, rationals, algebraic extensions of rationals, p-adic numbers, or a **finite field**.

- Elliptic curves groups for cryptography are examined with the underlying fields of F_p (*where p>3 is a prime*) and F_{2^m} (*a binary representation with 2^m elements*).

Using Elliptic Curves In Cryptography

- The central part of any cryptosystem involving elliptic curves is the **elliptic group**.
- All public-key cryptosystems have some underlying mathematical operation.
  - RSA has exponentiation (raising the message or ciphertext to the public or private values)
  - ECC has point multiplication (repeated addition of two points).

Elliptic Curve Picture



- Consider elliptic curve

  $E: y^2 = x^3 - x + 1$

- If P_1 and P_2 are on E, we can define

  $P_3 = P_1 + P_2$

  as shown in picture
- Addition is all we need

Generic Procedures of ECC

- Both parties agree to some publicly-known data items
  - The **elliptic curve equation**
    - values of ***a*** and ***b***
    - prime, ***p***
  - The **elliptic group** computed from the elliptic curve equation
  - A **base point**, B, taken from the elliptic group
    - Similar to the generator used in current cryptosystems
- Each user generates their public/private key pair
  - Private Key = an integer, x,
    selected from the interval [1, p-1]
  - Public Key = product, Q, of private key and base point
    (Q = x*B)

Example – ECC Analog to El Gamal

- Suppose Alice wants to send to Bob an encrypted message.
  - Both agree on a base point, B.
  - Alice and Bob create public/private keys.
    - Alice
      - Private Key = a
      - Public Key = P_A = a * B
    - Bob
      - Private Key = b
      - Public Key = P_B = b * B
  - Alice takes plaintext message, M, and encodes it onto a point, P_M, from the elliptic group

Example – ECC Analog to El Gamal

- Alice chooses another random integer, k from the interval [1, p-1]
- The ciphertext is a pair of points
  - $P_C = [\ (kB),\ (P_M + kP_B)\ ]$

- To decrypt, Bob computes the product of the first point from P_C and his private key, b
  - $b * (kB)$
- Bob then takes this product and subtracts it from the second point from P_C
  - $(P_M + kP_B) - [b(kB)] = P_M + k(bB) - b(kB) = P_M$
- Bob then decodes P_M to get the message, M.

Encryption Rules

- Suppose we let $\alpha = (2,7)$ and choose the private key to be 7
- Then $\beta = 7\alpha = (7,2)$

- Encryption:

$$e_K(x,k) = (k(\alpha), x + k(\beta))$$
$$e_K(x,k) = (k(2,7), x+k(7,2)) ,$$

where $x \in E$ and $0 \leq k \leq 12$

Decryption Rule

- Decryption:

$$d_K(y_1, y_2) = y_2 - K_{priv} y_1$$

$$d_K(y_1, y_2) = y_2 - 7y_1$$

(This is based on the ElGamal scheme of elliptic curve encryption)

How Alice can use the Scheme

- Suppose Alice wants to send a message to Bob.
- Plaintext is x = (10,9) which is a point in *E*

- Alice chooses a random value for k, ex: k = 3

  So now calculate (y_1, y_2):

  $y_1 = 3(2,7) = (8,3)$

  $y_2 = (10,9) + 3(7,2) = (10,9) + (3,5) = (10,2)$

- Alice transmits y = ((8,3),(10,2))

How Bob can decrypt

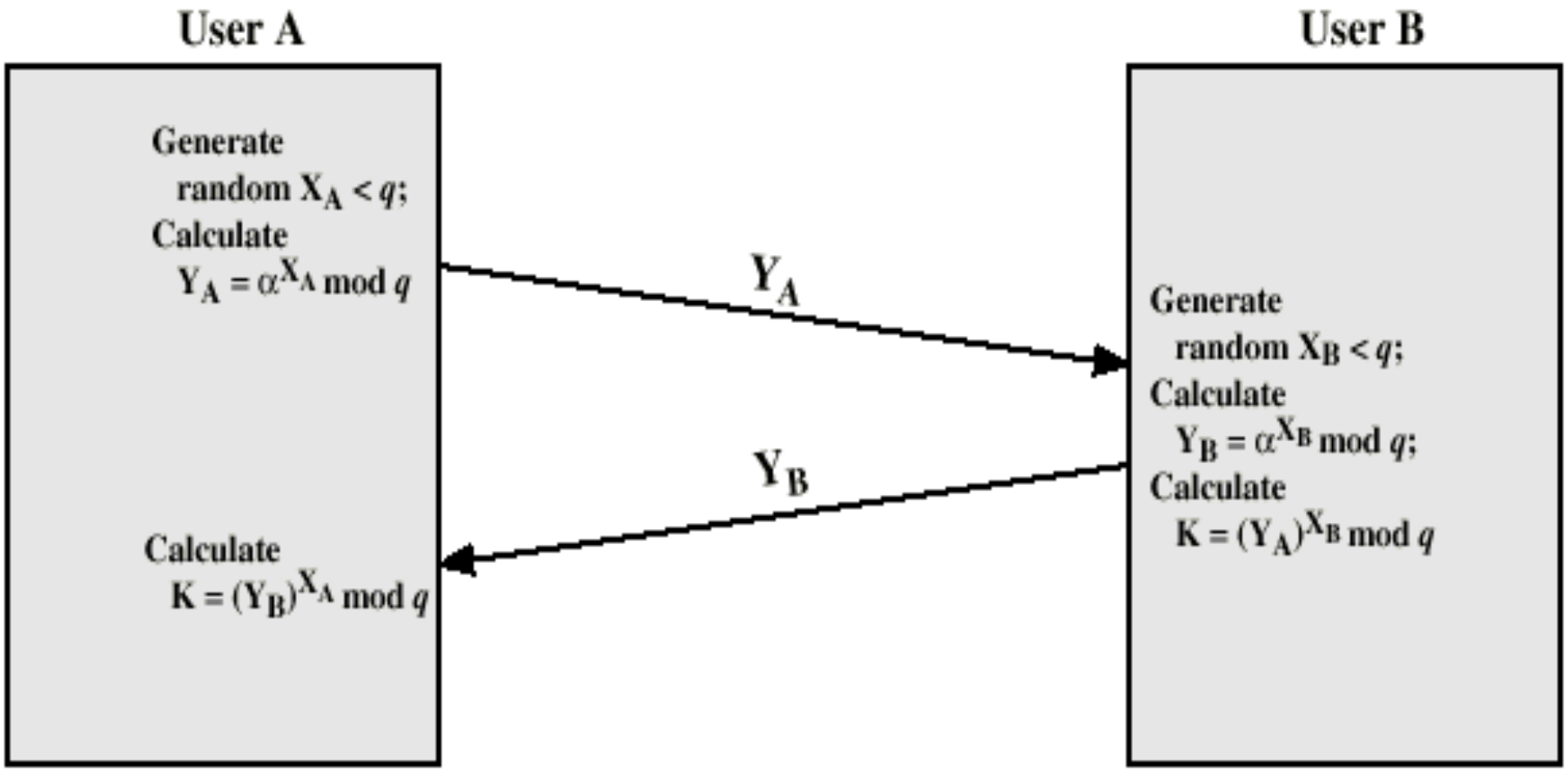
- Bob receives y = ((8,3),(10,2))
- Calculates

$$x = (10,2) - 7(8,3)$$
$$= (10,2) - (3,5)$$
$$= (10,2) + (3,6)$$
$$= (10,9)$$

Which was the plaintext

ECDH Model

Remembering Diffie-Hellman (DH) Key Exchange



ECC Diffie-Hellman

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

a(x,y)

b(x,y)

Alice, A

Bob, B

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

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 = P_A = a * B
    - Bob
      - » Private Key = b
      - » Public Key = P_B = 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**

Examples for a specific Curve

- Curve P-192 (Defined by NSA)

p = 6277101735386680763857894232076664160839087003902496127 9

r = 6277101735386680763857894231760590137671947731828422840 81

a = 3099d2bb bfcb2538 542dcd5f b078b6ef 5f3d6fe2 c745de65

b = 64210519 e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1

G_x = 188da89e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012

G_y = 07192b95 ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811

ECCDSA SIGNTAURES

Initial Parameters

Alice and Bob must share:

- The curve to use: *CURVE*
- *G*: elliptic curve base point, a generator of the elliptic curve with large prime order *n*
- *n*: integer order of *G*, means that n.*G* = O

ECCDSA Keypair

- Alice creates a key pair, consisting of a private key integer d_A randomly selected in the interval [1, n-1] and a public key curve point $Q_A = d_A \times G$ (Eliptic Curve Point Mutiplication by a Scalar)

ECCDSA Signature construction

- h = H(M) // ex., SHA-2
- Take the z Ln lefmost bits
- Select a crypto secure random K
- Calculate the curve point (x1,y1) = K.G
- Calculate r = x1 mod n, if r=0 go back to select K again
- Calculate $s = k^{-1} (z + r.d_A) \bmod n$
  - If s=0, go cack and recompute K
- Os s !=0, the signature is (r,s)

This is a DSA Based Signature

The validation folloes a DSA-based validation

ECC Concerns

- ***Political concerns***: the trustworthiness of NIST - produced curves being questioned after revelations that the NSA willingly inserts backdoors into software, hardware components and published standards were made;
  - well-known respectable cryptographers have expressed doubts about how the NIST curves were designed, and voluntary tainting has already been proved in the past.

- ***Technical concerns***: the difficulty to properly implement the standard and the slowness and design flaws which reduce security in insufficiently precautions implementations on random number generations