Computer and Network Security: Asymmetric Encryption

Kameswari Chebrolu

Outline

- **Modern Cryptography**
  - Overview
  - **Confidentiality**
    - Background: Definition, Crypto-analysis, One Time Pads
    - Symmetric key encryption, Block modes
    - **Asymmetric key encryption**
  - Integrity (includes Authentication)
    - Hashes, MAC, Digital signature

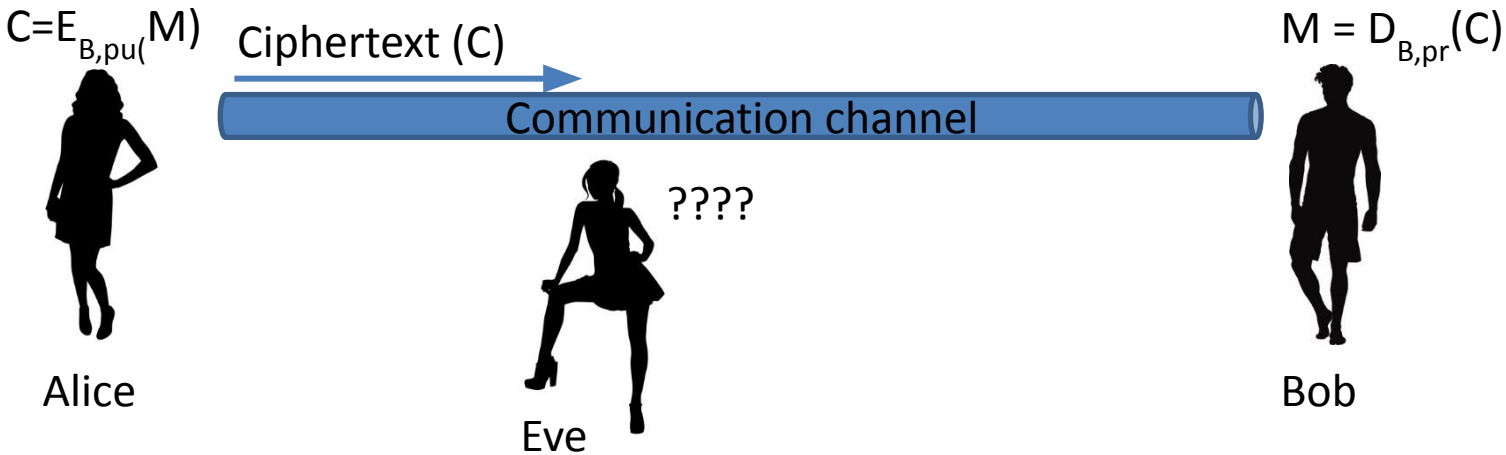
Outline

- **Why public key systems?**
- RSA public key algorithm
  - Background: Modular Arithmetic
  - Basic Idea
  - Implementation aspects
  - Attacks and best practices
  - PKCS standard

Recap

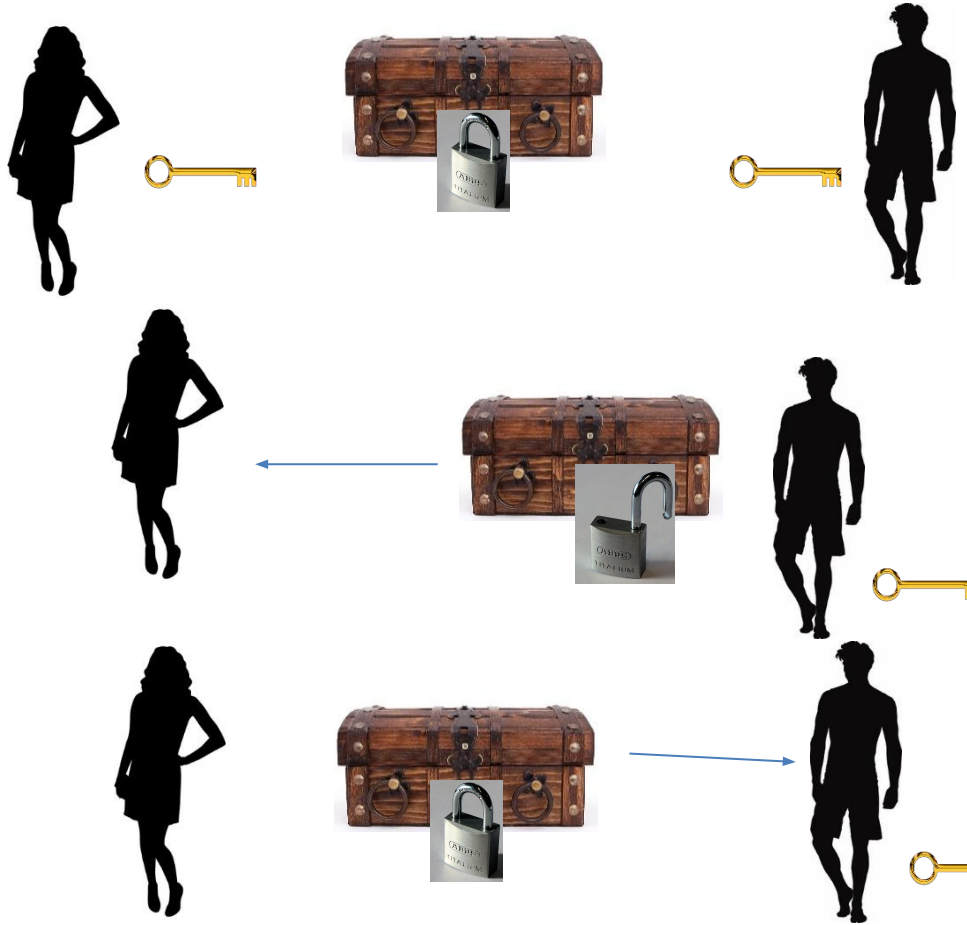
- Asymmetric/Public key:
  - Bob's public key (B,pu) open
    - Both Eve and Alice have access to this
  - Bob keeps private key (B,pr) secret
  - Alice encrypts message with Bob's public key

Public key algorithms can be used for encryption, integrity and key agreement



Postal Analogy

- Symmetric key
- Asymmetric key



Motivation

- Encryption based on a key
- How do two communicating parties (two machines say) who never met communicate?
- Challenge: Key Management
  - Often the most difficult part of the cryptography
  - Involves key generation, **distribution**, storage, use and replacement
  - Distribution: Make key available to entities requiring secure communication

Key Generation

- Symmetric: Often random k-bit string; simple to generate
- Asymmetric: Special structure, expensive to generate
- Key lengths are not comparable
  - 3072-bit RSA key equivalent in strength to 128-bit symmetric key

Key Distribution

- Symmetric Encryption:
  - Key distribution not easy
    - Can agree to key in person, but what about machines?
  - n users. How many keys needed?
  - $^{n}c_{2}$: $n(n-1)/2 \rightarrow o(n^2)$

- Asymmetric Encryption:
  - Easier: Want to send a message confidentially to me? Encrypt using my public key
  - n users. How many keys needed?
  - 2n; o(n)
  - Catch: Need to ensure integrity of public key (not confidentiality)

Key Distribution: Big topic, will be covered later in depth

Outline

- Why public key systems?
- **RSA public key algorithm**
  - **Background: Modular Arithmetic**
  - Basic Idea
  - Implementation aspects
  - Attacks and best practices
  - PKCS standard

Background

- 1976: Diffie and Hellman demonstrated a radically different and elegant **asymmetric key cryptosystem**
  - Allows two entities to produce a shared secret over insecure channel (key management)
  - Can be used to derive shared keys for symmetric encryption like AES or DES
- Led to development of many Public/asymmetric key crypto systems
  - Not all are practical or secure
  - Based on solving a NP problem
  - Usage: **confidentiality**, Integrity, authenticity, key agreement

Examples: Asymmetric Encryption

- Knapsack Algorithms (Merkle and Hellman)
- **RSA (Rivest, Shamir and Adleman)**
  - Based on difficulty of factoring large numbers
- Rabin Scheme
  - Based on difficulty of finding square roots modulo a composite number
- ElGamal
  - Based on difficulty of calculating discrete logarithms in a finite field
- …….

Background: Modular Arithmetic

- x mod n □ remainder of x when divided by n
- Arithmetic in Z_n = {0,1,2 ……. n-1}
  - Reduce all results to a value in Z_n
- Example:
  - (3+9) mod 10 = 2
  - (6*8) mod 10 = 8

Modular Inverse

- y is modular inverse of x, modulo n if

  $xy \bmod n = 1$
- Eg. 2 is inverse of 7 mod 13 ($14 \bmod 13 = 1$)
- Not every element in Z_n may admit a modular inverse
- If n is a prime, every element other than 0 in Z_n has a modular inverse
- General Rule: All number **relative prime** to n will have modular inverses
  - Relative primes do not share any common factor other than 1

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

n=8

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11	12
2	0	2	4	6	8	10	12	1	3	5	7	9	11
3	0	3	6	9	12	2	5	8	11	1	4	7	10
4	0	4	8	12	3	7	11	2	6	10	1	5	9
5	0	5	10	2	7	12	4	9	1	6	11	3	8
6	0	6	12	5	11	4	10	3	9	2	8	1	7
7	0	7	1	8	2	9	3	10	4	11	5	12	6
8	0	8	3	11	6	1	9	4	12	7	2	10	5
9	0	9	5	1	10	6	2	11	7	3	12	8	4
10	0	10	7	4	1	11	8	5	2	12	9	6	3
11	0	11	9	7	5	3	1	12	10	8	6	4	2
12	0	12	11	10	9	8	7	6	5	4	3	2	1

n=13

Totient Function

- $\phi(n)$: Number of elements in Z_n relative prime to n
- If n is prime, $\phi(n) = n - 1$
- If n is product of primes p and q,

$$\phi(n) = (p-1)(q-1)$$

Modular Exponentiation

- x^y mod n
- E.g. 4^6 mod 10 = 6
- $\phi(n)$= 4 for n=10
  - 1,3,7,9 are relative prime to 10
- $x^y \bmod n = x^{y \bmod \phi(n)} \bmod n$

(from Euler's Theorem)

x^y	0	1	2	3	4	5	6	7	8	9
0		0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	6	2	4	8	6	2
3	1	3	9	7	1	3	9	7	1	3
4	1	4	6	4	6	4	6	4	6	4
5	1	5	5	5	5	5	5	5	5	5
6	1	6	6	6	6	6	6	6	6	6
7	1	7	9	3	1	7	9	3	1	7
8	1	8	4	2	6	8	4	2	6	8
9	1	9	1	9	1	9	1	9	1	9

Exponentiation modulo 10

Above not true for all n, only for n's that are square free i.e. not have p^2 as a factor for any prime p. It is true for primes and product of distinct primes (what matters in RSA)

Outline

- Why public key systems?
- RSA public key algorithm
  - Background: Modular Arithmetic
  - **Basic Idea**
  - Implementation aspects
  - Attacks and best practices
  - PKCS standard

RSA (Rivest, Shamir and Adleman)

- Public key cryptographic algorithm that does encryption/decryption (and digital signature)
- Key size = cipher block size > input block size
- Variable key size
  - Typically 1024 to 4096 bits
  - E.g. 2048 bit key (256 bytes); Block size 214 bytes
- View Plaintext/Cipher text as large numbers represented by thousands of bits

Key Generation

- p,q: two large prime numbers
- n = p*q; $\phi(n)$= (p-1)(q-1)
- Pick e relative prime to $\phi(n)$
- Set $d = e^{-1} \, mod \, \phi(n)$
- Public key (e,n)
- Private key (d,n)

Encryption/Decryption

- Encrypt m (<n) to generate ciphertext c

$$c = m^e \, mod \, n$$

- Decrypt c to recover m

$$m = c^d \, mod \, n$$

- Smaller e/d □ faster encryption/decryption
  - e is often a small integer (3 or 65537); ok security wise since n is still large
  - d has to be large to avoid brute-force search

Real World Example

- **p**
  12131072439211271897323671531612440428472427633701410925634549312301964373042085619324197365322416866541017057361365214171711713797974299334871062829803541
- **q**
  12027524255478748885956220793734512128733387803682075433653899983955179850988797899869146900809131611153346817050832096022160146366346391812470987105415233
- **n**
  14590676800758332323018693934907063529240187237535716439958187101987343879900535893836957140267014980212181808629246742282815702292207674690654340122488967247240792696998710058129010319931785875366371086235765651050788371429711563734278891146353510271203276516651841172685983798867211183720508552634661874005 3
- ***ϕ*(*n*)**
  14590676800758332323018693934907063529240187237535716439958187101987343879900535893836957140267014980212181808629246742282815702292207674690654340122488964831381123227996631730139777785236530154784827347887129722205858745715289160645926971811926897116355507080264399952954964411681194751651393818429668352128 0
- **e**

  65537
- **d** - the private key
  89489425009274444368228545921773093919669586065884257445497854456487674839629818390934941973262879616797970608917283679875499331574161113854088813275488110588247193077582527278437906504015680623423550067240042466665654232383502922215493623289472138866445818789127946123407807725702626644091036502372545139713

Why RSA works?

$$
\begin{aligned}
c^d \bmod n &= (m^e)^d \bmod n \\
&= m^{ed \bmod \phi(n)} \bmod n && \text{from Euler's theorem} \\
&= m \bmod n && \text{since } ed = 1 \bmod \phi(n) \\
&= m && \text{since } m < n
\end{aligned}
$$

Outline

- Why public key systems?
- RSA public key algorithm
  - Background: Modular Arithmetic
  - Basic Idea
  - **Implementation aspects**
  - Attacks and best practices
  - PKCS standard

Implementation: Generating Key

- Primality Testing: To choose p and q
  - No practical way to determine if a number is prime
  - Efficient/Fast probabilistic algorithms exist (e.g. Miller Rabin test)
- Compute gcd: To choose e relative prime to $\phi(n)$
  - Have a small constant e
    - Choose p, q such that (p-1) and (q-1) are relative prime to e
  - Popular e: 3 or 65537 (makes encryption faster)

- Compute gcd: to choose e relative prime to $\phi(n)$
  - Have a small constant e
    - Choose p, q such that (p-1) and (q-1) are relative prime to e
  - Popular e: 3 or 65537 (makes encryption faster)
  - Small constant e does not weaken security
    - e=3 requires message be padded with random bits. Why?
- Compute modular inverse: to calculate d
  - Can be done efficiently using Euclid's algorithm

Implementation: Using key

- Less computationally intensive than generating key
  - Good because this is more often used
- Involves Modular Power: to calculate

$$m^e \, mod \, n \, or \, c^d \, mod \, n$$

  - Decryption: d is a very large number
  - Naïve approach: d modular multiplications
  - Efficient approach: Repeated squaring
    - Determine c, c^2, c^4, c^8, c^{16}, …. (modulo n)
    - Take binary representation of number d
    - Multiply elements in series whose position corresponds to 1 in binary representation

Example

- Say we want to calculate c^{43}
- 43 in binary is 101011
- So do: $c^1 * c^2 * c^8 * c^{32} = c^{43}$ (all mod n)
- Number of multiplications proportional to number of bits used to represent d (log(d))

Outline

- Why public key systems?
- RSA public key algorithm
  - Background: Modular Arithmetic
  - Basic Idea
  - Implementation aspects
  - **Attacks and best practices**
  - PKCS standard

Security

- Assumes factoring a big number is hard
  - Difficult to find d, given e and n □ difficult to find $\phi(n)$ □ difficult to factor n
    - If we can factor n, then we know $\phi(n)$ (=(p-1)(q-1))
  - Factoring a 512 bit number took 8400 MIPs years in 1999
  - Use of 2048-bit **keys,** sufficient until 2030
- There may be other means (other than factoring) of breaking RSA

Attacks: Details Matter

- **Small message set:** Say we need to send one name confidentially (using RSA) out of 100 names
  - Not secure. Why?
  - Attacker: encrypt each name with public key; match these with transmitted message
- **Identical** messages also problematic (similar to block modes; deterministic)
- Solution: Concatenate/Pad name with large random number (say 128 bits)
    - Attacker needs to check $100 * 2^{128}$ messages

- **Constant and small encryption exponent (e=3):**
  - Same message sent to more than e recipients (with different public keys; n's differ), can recover m
  - Solution: Pad message with random values
- **Small d**
  - (M. Weiner) Can recover d if p is between q and $2q$ and $d < 1/3 * n^{1/4}$
  - Solution: Choose large d

- RSA is **malleable**: Given a ciphertext c that encrypts a message m, possible to generate a ciphertext c' that encrypts a related message m'
  - E.g. Attacker can convert m into m^k without decrypting
  - Solution: Padding

- **Side channel attacks:**
  - Monitor the timing and power measurements on a device; induce faults into a chip
  - Time/Errors in computing c^d mod n for many c may reveal d
- Other Attacks possible:
  - Using same key for confidentiality and digital signature
  - Using common n across a group of users (e is different)

Example

Conditional Multiply

C^d mod n; d=(d_0,d_1,d_2,…….d_k) with d_0=1

```
x=c
for(i=1;i<=k;i++){
      x=x^2 mod n; //square
      If (di==1)
         x=x * c mod n;
//multiply
}
return x
```

Unconditional Multiply

C^d mod n;

```
x=c
For(i=1;i<=k;i++) {
      x=x^2 mod n; //square
      y=x * c mod n; //multiply
      If (di==1)
         x=y
}
return x
```

Lessons Learnt

- Pad messages with random values
- Make sure m is about the same size as n
- Choose a large value of d
- Don’t share n among a group of users
- Always hash a document before signing with RSA

Outline

- Why public key systems?
- RSA public key algorithm
  - Background: Modular Arithmetic
  - Basic Idea
  - Implementation aspects
  - Attacks and best practices
  - **PKCS standard**

Public Key Cryptography Standards (PKCS)

- Helps different implementations interoperate
- Also avoids common pitfalls
- Set of standards ranging from 1 to 15
- PKCS#1:
  - format of RSA public and private keys
  - Basic algorithms
  - encoding/padding schemes for performing RSA encryption, decryption, and producing and verifying signatures.

Table 1: PKCS Specifications

No.	PKCS title	Comments
1	RSA Cryptography Standard	
2,4		incorporated into PKCS #1
3	Diffie-Hellman Key Agreement Standard	superseded by IEEE 1363a etc.
5	Password-Based Cryptography Standard	
6	Extended-Certificate Syntax Standard	never adopted
7	Cryptographic Message Syntax Standard	superseded by RFC 3369 (CMS)
8	Private-Key Information Syntax Standard	
9	Selected Object Classes and Attribute Types	
10	Certification Request Syntax Standard	
11	Cryptographic Token Interface Standard	referred to as CRYPTOKI
12	Personal Information Exchange Syntax Standard	
13	(*reserved for ECC*)	never been published
14	(*reserved for pseudo random number generation*)	never been published
15	Cryptographic Token Information Syntax Standard	

Encoding in PKCS#1

00	02	Random non-zero bytes (>=8)	00	Plaintext

- First octet: ensure m < n
- Second octet: format type
  - 2 is for encryption, 1 is for signature etc
- Padding : helps with attacks
  - Small message set, small and constant e

Checkout Bleichenbacher attack and solution in form of OAEP padding

Comparison

	Symmetric	Asymmetric
Keys	1	2
Key exchange	Difficult	Easier
Number of keys	Exponential growth	Linear growth
Key Length	56-256 bits	1024-15360 bits
Speed	Fast (bit manipulations)	Very slow (large number)

Key comparison:

Symmetric 80 eq
Asymmetric 1024

Symmetric 256 eq
Asymmetric 15360

Hybrid Usage

- Use RSA to encrypt a shared key
- Then encrypt message with AES using the shared key
- Example: A wants to send a message m to B
  - A sends $E_{(B,pu)}(k)$ to B using RSA
  - A sends $E_k(m)$ to B using AES; B can decrypt it with k

Elliptical Curve Cryptography (ECC)

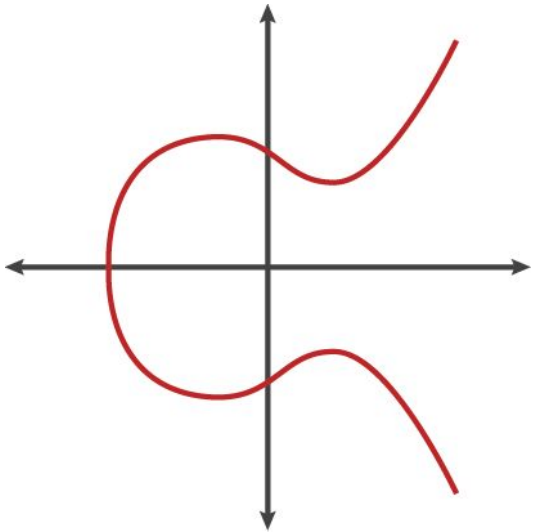
- ECC is taking over RSA. Why?
  - Use of 2048-bit **keys,** sufficient until 2030
  - Post this need longer keys (e.g. 4096)
  - Longer key → more resources
    - unsuitable in embedded system, IoT etc
- ECC: Same strength but at much smaller keysize
  - appealing for devices with limited storage or processing power

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 1: NIST Recommended Key Sizes

ECC

- Proposed in 1985 based on an esoteric branch of mathematics called elliptic curves
  - Complex math, not as easy as RSA
- $y^2=x^3+ax+b$
  - Over real numbers is too slow and inaccurate due to rounding errors, hence use only whole numbers in a fixed range
    - pick the maximum to be a prime number
    - The elliptic curve is called a prime curve and has excellent cryptographic properties



- **The equation of the elliptic curve over $\mathbb{F}_p$ is defined as:**

$$y^2 \bmod p = (x^3 + ax + b) \bmod p$$

$$where : (4a^3 + 27b^2) \bmod p \neq 0$$

$$x, y, a, b \in [0, p-1]$$

- **The points on *E* are denoted as:**

$E(\mathbb{F}_p)=\{(x,y):x,y\in\mathbb{F}_p$ satisfy $y^2=x^3+ax+b\}\cup\{\mathcal{O}\}$

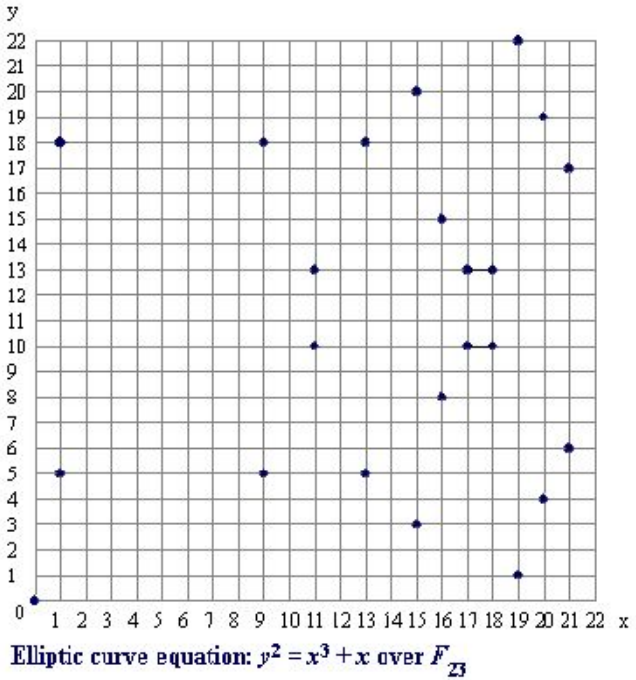
- **Example: Elliptic curve $y^2 = x^3 + x$ over the Prime field $\mathbb{F}_{23}$. The points in the curve are the Following:**

(0,0) (1,5) (1,18) (9,5) (9,18) (11,10) (11,13) (13,5)

(13,18) (15,3) (15,20) (16,8) (16,15) (17,10) (17,13) (18,10)

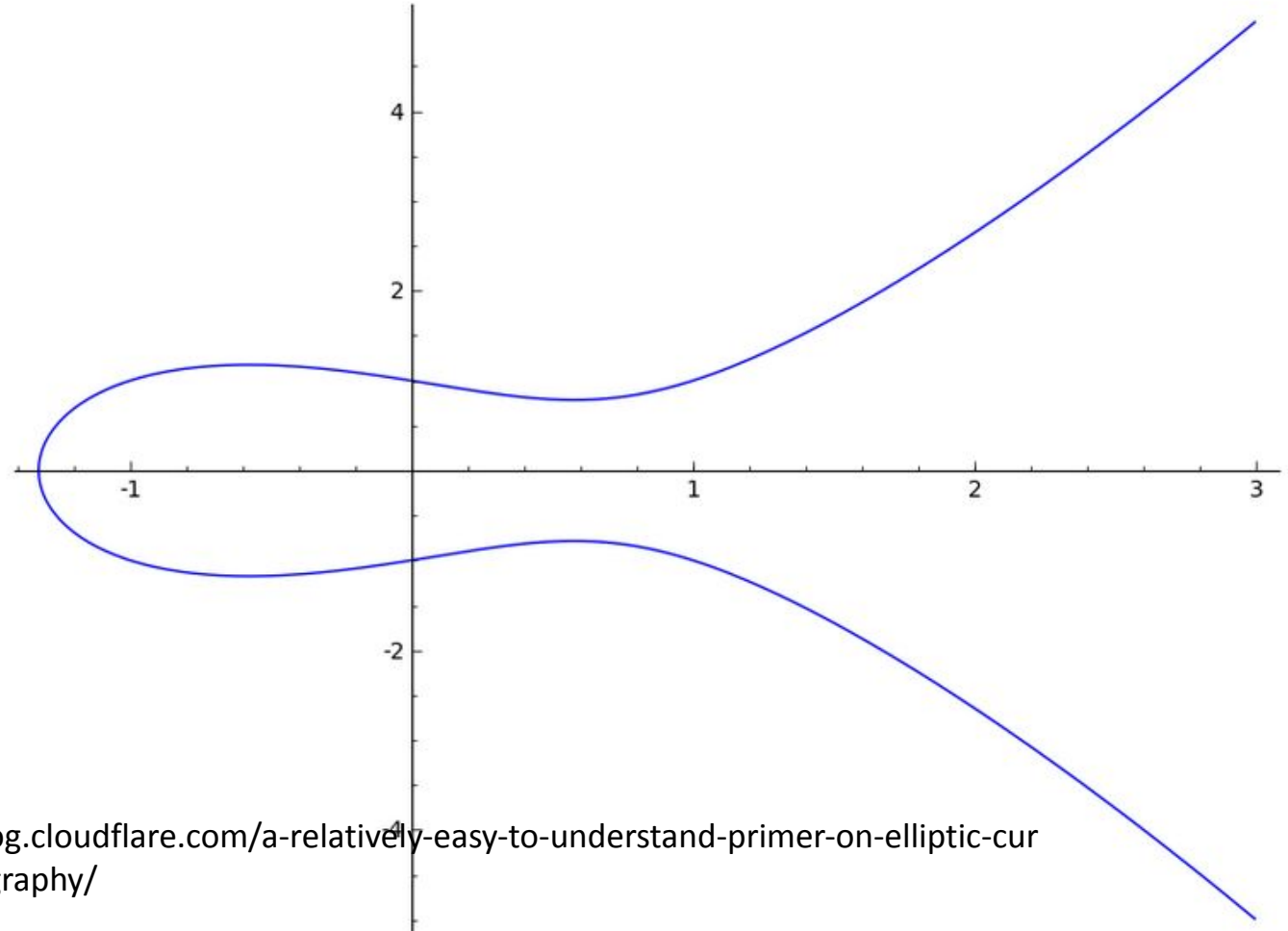
(18,13) (19,1) (19,22) (20,4) (20,19) (21,6) (21,17)

(From www.certicom.com)



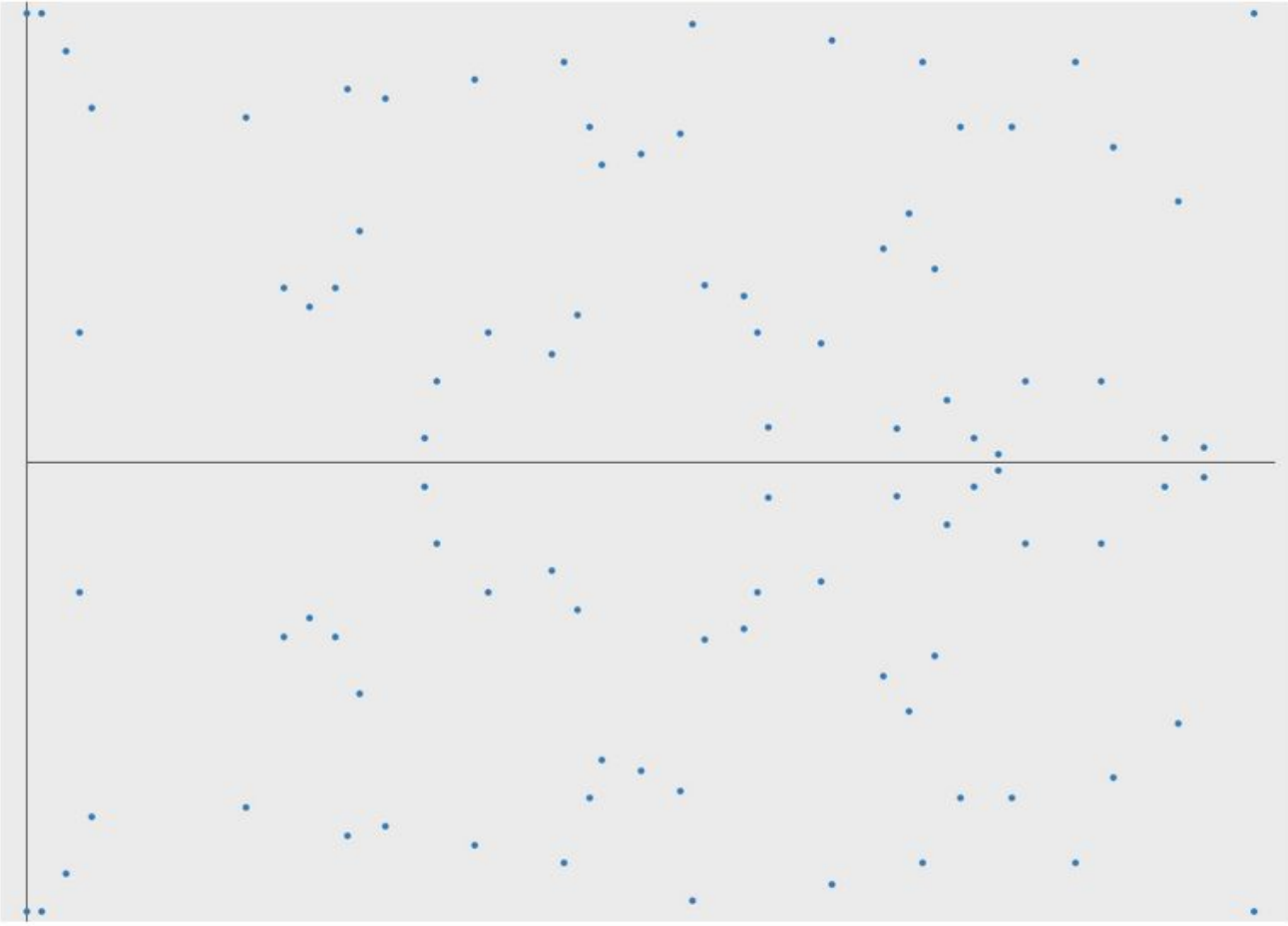
Elliptic curve equation: $y^2 = x^3 + x$ over F_{23}

Here's an example of a curve ($y^2 = x^3 - x + 1$) plotted for all numbers:



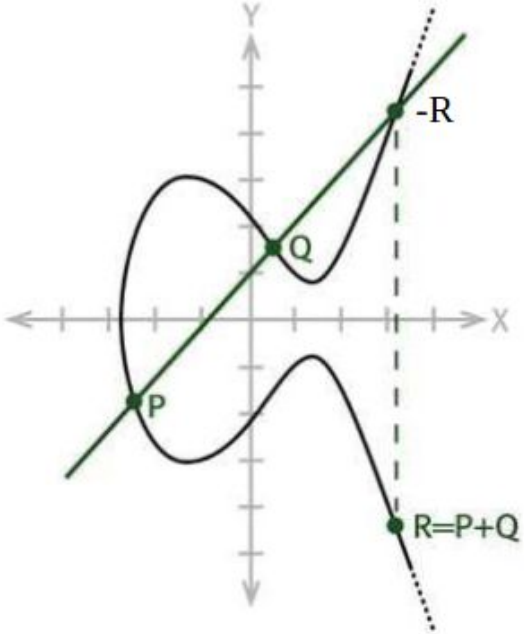
https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/

Here's the plot of the same curve with only the whole number points represented with a maximum of 97:



Geometry approach:

- *To add two distinct points P and Q on an elliptic curve, draw a straight line between them. The line will intersect the elliptic cure at exactly one more point –R. The reflection of the point –R with respect to x-axis gives the point R, which is the results of addition of points R and Q*



Addition

- For the curve E: $y^2 = x^3 + ax + b$. Let $P=(x_P,y_P)$ and $Q=(x_Q,y_Q) \in E$ with $P \neq Q$, then $R=P+Q=(x_R,y_R)$ is determined by the following formulae:

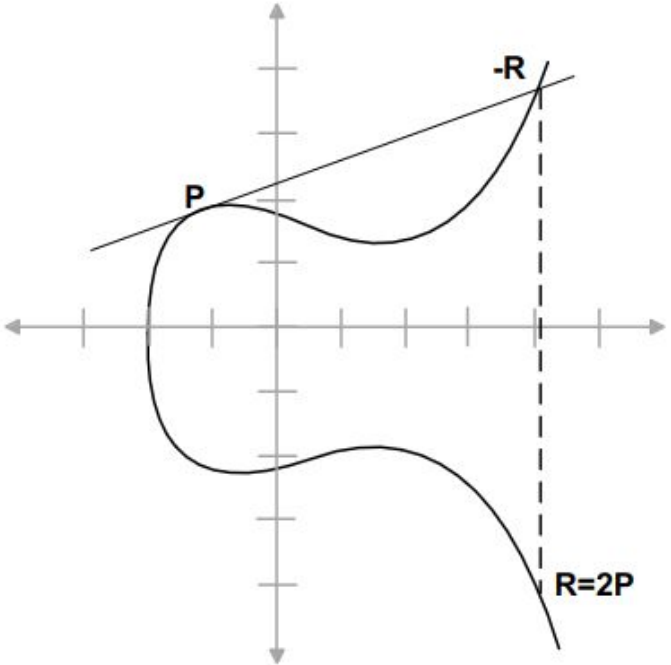
$$x_R = \lambda^2 - x_P - x_Q$$

$$y_R = \lambda(x_p - x_R) - y_P$$

$$where \ : \lambda = \frac{y_Q - y_P}{x_Q - x_p}$$

Geometry approach:

- *To the point P on elliptic curve, draw the tangent line to the elliptic curve at P. The line intersects the elliptic cure at the point –R. The reflection of the point –R with respect to x-axis gives the point R, which is the results of doubling of point P.*



Doubling Point P

- For the curve E: $y^2 = x^3 + ax + b$. Let $P=(x_P,y_P) \in E$ with $P \neq -P$, then $R=2P=(x_R,y_R)$ is determined by the following formulae:

$$x_R = \lambda^2 - 2x_P$$

$$y_R = \lambda(x_p - x_R) - y_P$$

$$where \ : \lambda = \frac{3x_P^2 + a}{2y_p}$$

Point Addition and Doubling for EC over $\mathbb{F}_p$

- **Point addition:**

$$x_R = (\lambda^2 - x_P - x_Q) \bmod p$$
$$y_R = (\lambda(x_p - x_R) - y_P) \bmod p$$
$$where \ : \lambda = \frac{y_Q - y_P}{x_Q - x_p} \bmod p$$

- **Point doubling:**

$$x_R = (\lambda^2 - 2x_P) \bmod p$$
$$y_R = (\lambda(x_p - x_R) - y_P) \bmod p$$
$$where \ : \lambda = \frac{3x_P^2 + a}{2y_p} \bmod p$$

Scalar Multiplication

- **Intuitive approach:**

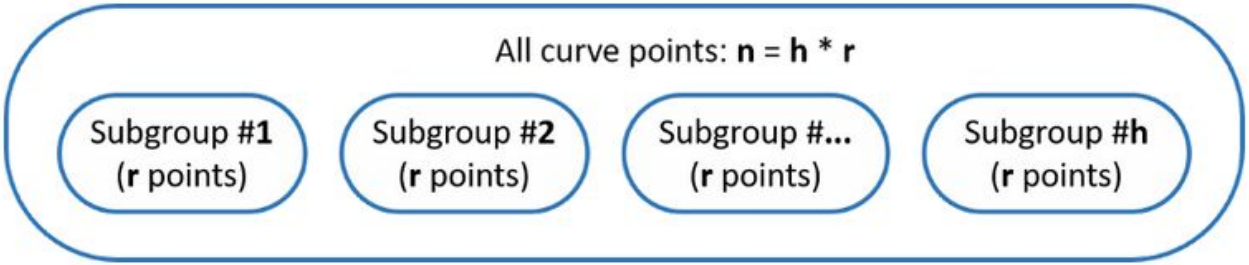
$$dP = \underbrace{P+P+\ldots+P}_{d \text{ times}}$$

It requires *d*-1 times point addition over the elliptic curve.

- **Observation: To compute 17 *P*, we could start with 2*P*, double that, and that two more times, finally add *P*, i.e. 17*P*=2(2(2(2*P*)))+*P*. This needs only 4 point doublings and one point addition instead of 16 point additions in the intuitive approach. This is called Double-and-Add algorithm.**

Generator

- O**rder of the curve** is the **total number of all EC points** on the curve
- A cyclic group: two EC points are added or an EC point is multiplied to an integer, the result is another EC point from the same cyclic group
- Some curves form a single **cyclic group** (holding all their EC points)
- Others form several non-overlapping cyclic groups
  - **h** cyclic subgroups (partitions), each of order **r** (each subgroup holds equal number of points)
  - The **order** of entire group is **n = h * r**
  - The number of subgroups **h** holding the EC points is called **cofactor**
- Example of elliptic curve having **cofactor = 1** is secp256k1



- Generator: a special pre-defined EC point which can generate any other point in its subgroup by multiplying G by some integer in the range [0...r]
  - The number r is the "order" i.e the total number of all points in the subgroup
- When G and n are carefully selected, and the cofactor = 1
  - All possible EC points on the curve can be generated from G by multiplying it by integer in the range [1...n]

Domain parameters for the curve secp256k1 (the Bitcoin curve):

- ***p*** (modulus) =
  `0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F`
- ***n*** (order; size; the count of all possible EC points) =
  `0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141`
- ***a*** (the constant "a" in y^2 ≡ x^3 + a*x + b (mod p)) =
  `0x0000000000000000000000000000000000000000000000000000000000000000`
- ***b*** (the constant "b" in y^2 ≡ x^3 + a*x + b (mod p)) =
  `0x0000000000000000000000000000000000000000000000000000000000000007`
- ***g*** (the curve generator point G {x, y}) =
  (`0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798` ,
  `0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8`)
- ***h*** (cofactor, typically 1) = 01

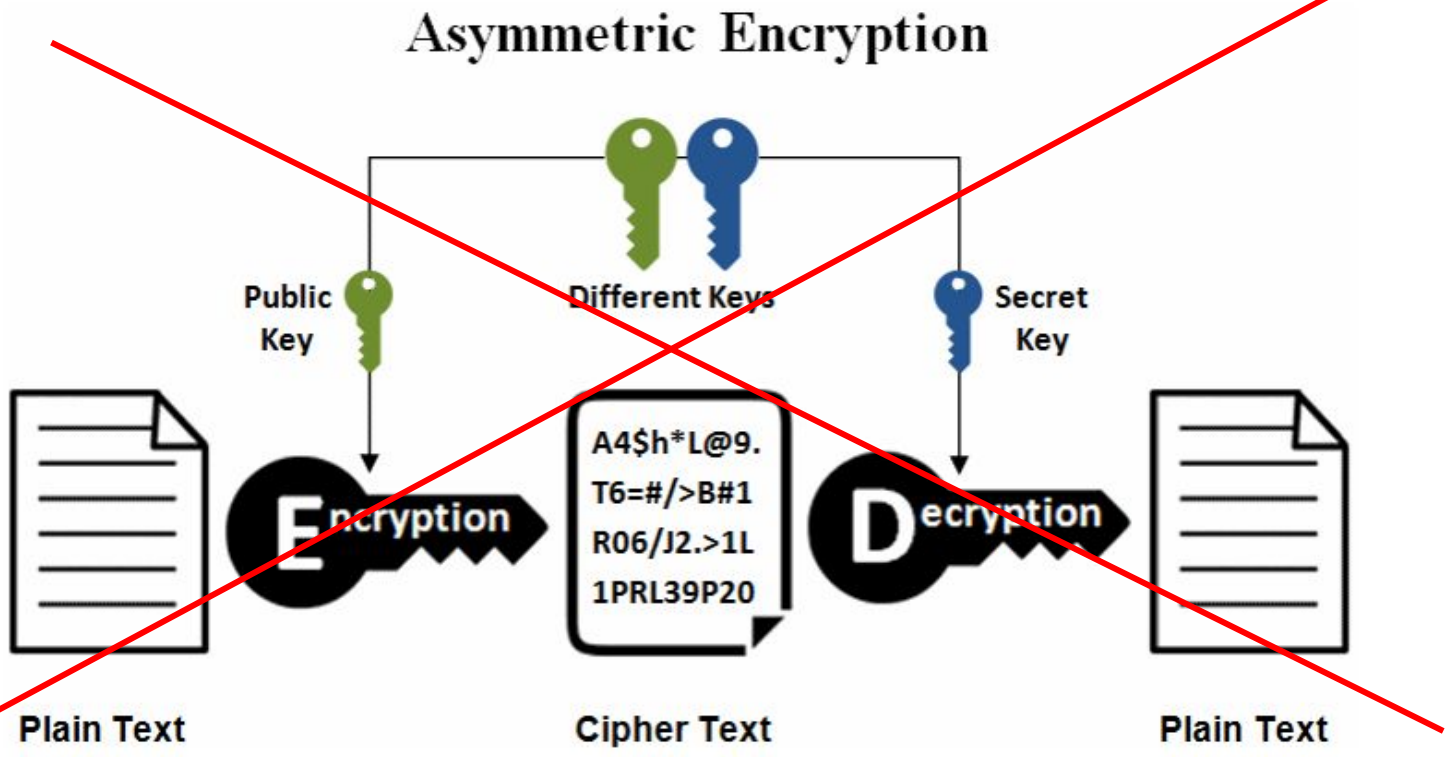
Public/Private Keys

- **Elliptic curve** (EC) over finite field $\mathbb{F}p$
- **G** == **generator point** (fixed constant, a base point on the EC)
- **k** == **private key** (integer)
- **P** == **public key** (point)

P = kG

- k is a random number (private key, e.g. 256 bit)
- Public key is a point (but can be represented in compressed form in 257 bits. how?)
- Very fast to calculate P (double and add algorithm, $\log_2(k)$)
- Infeasible (for large k) to calculate k = P / G
  - Elliptic Curve Discrete Logarithm Problem (ECDLP))
  - Starting with a point G and final point P (both Public), very hard to determine 'k' , number of point additions

Encryption/Decryption



- Alice has a private key a and a public key A=aG
  - She publishes her public key A
- Bob wants to send a message to Alice
  - Picks a random value b and computes the points bG and bA
  - Gives the point bA to a key derivation function h, which produces a symmetric key
  - Use the symmetric keys to encrypt the message
  - Send the values bG and $\text{Encrypt}_{(h(bA)}$(Message)
- Alice decrypts the message
  - Alice receives bG and $\text{Encrypt}_{(h(bA)}$(Message)
  - She first computes the point a(bG) (same as b(aG)=bA)
  - She passes bA to key derivation function to get the symmetric key
  - She then decrypts the value $\text{Encrypt}_{(h(bA)}$(Message)

https://en.wikipedia.org/wiki/Elliptic_curve_point_multiplication

https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc

Summary

- Public key algorithms: elegant and beautiful; based on strong mathematics
  - Eases key distribution (for confidentiality)
- Focus: RSA in depth, ECC at some high level
- Idea to implementation: Need attention to details
  - Variety of attacks possible
- Standard (PKCS) helps in interoperability and takes care of implementation pitfalls