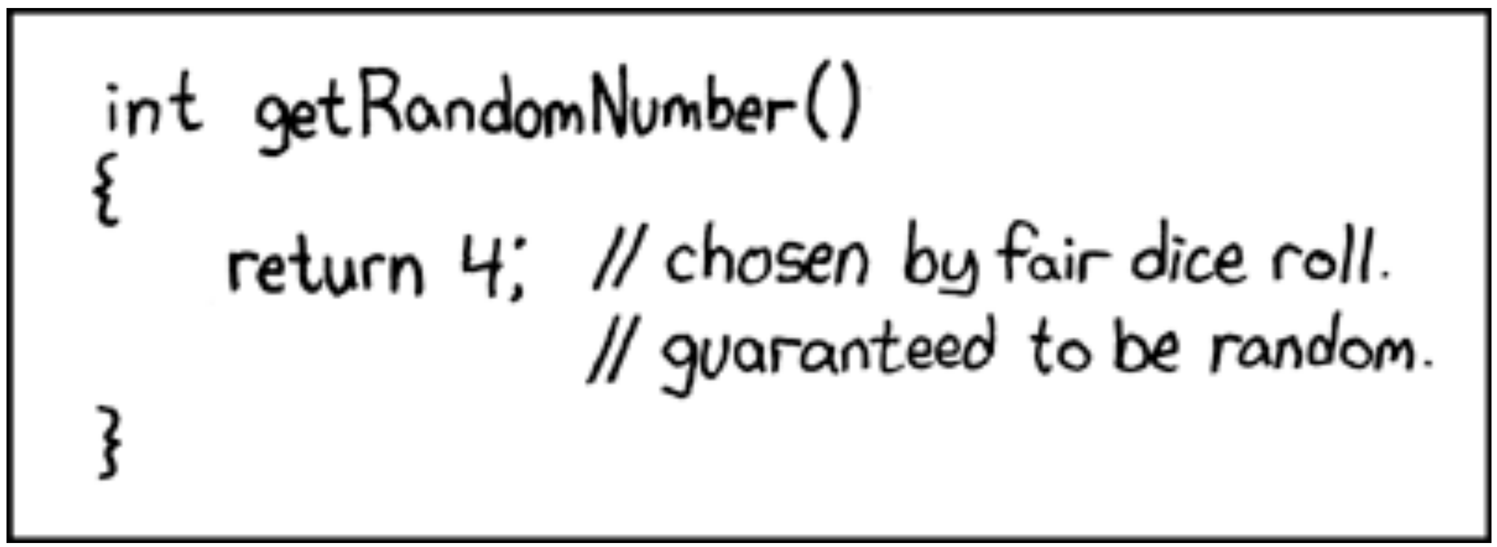
Public-Key Cryptography

February 3, 2022



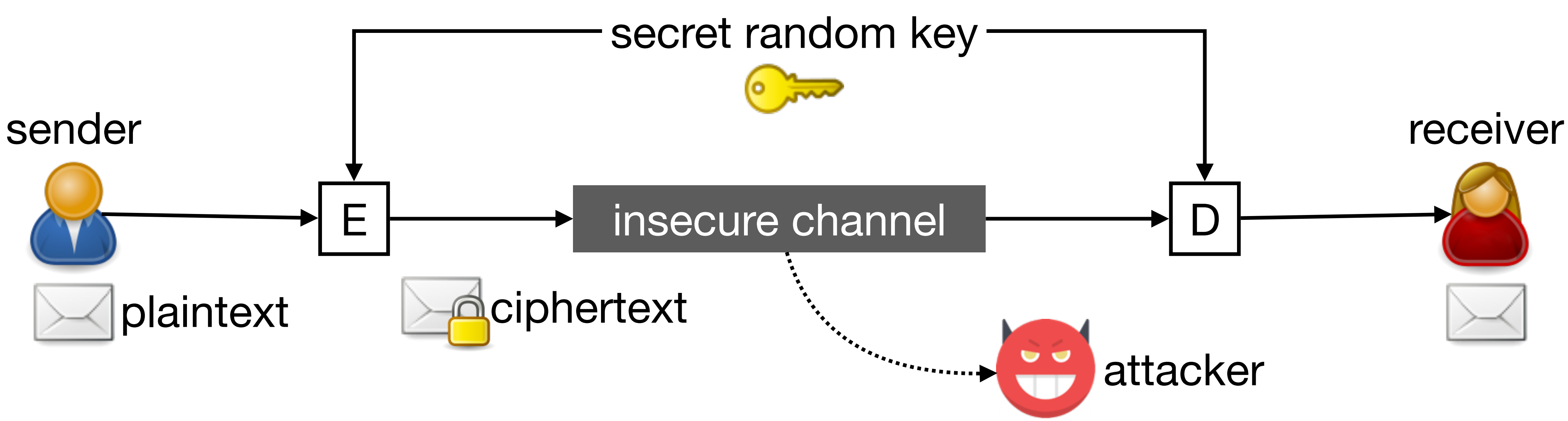
Homework 1 & Today

- Homework 1
  - available on **Blackboard**
  - based on cryptography lectures, **requires Python or Java programming**
  - due **February 20th** (Sunday) at 11:59pm
- Today: **public-key encryption**

  Where do secret keys come from?
  - **RSA**
  - ElGamal, elliptic curves

Feedback: `https://forms.gle/JGbNCmCsU69iWaTv8`

Secret-Key Encryption



- Sender and receiver know the secret key → can encrypt/decrypt
- Attacker does not know the secret key → cannot encrypt/decrypt
- Exchanging or agreeing on a key
  - either using a secure side channel
  - or before communicating over the insecure channel

Practical Problem: Key Exchange

web browser on a laptop

insecure channel
(public WiFi network, Internet, etc.)

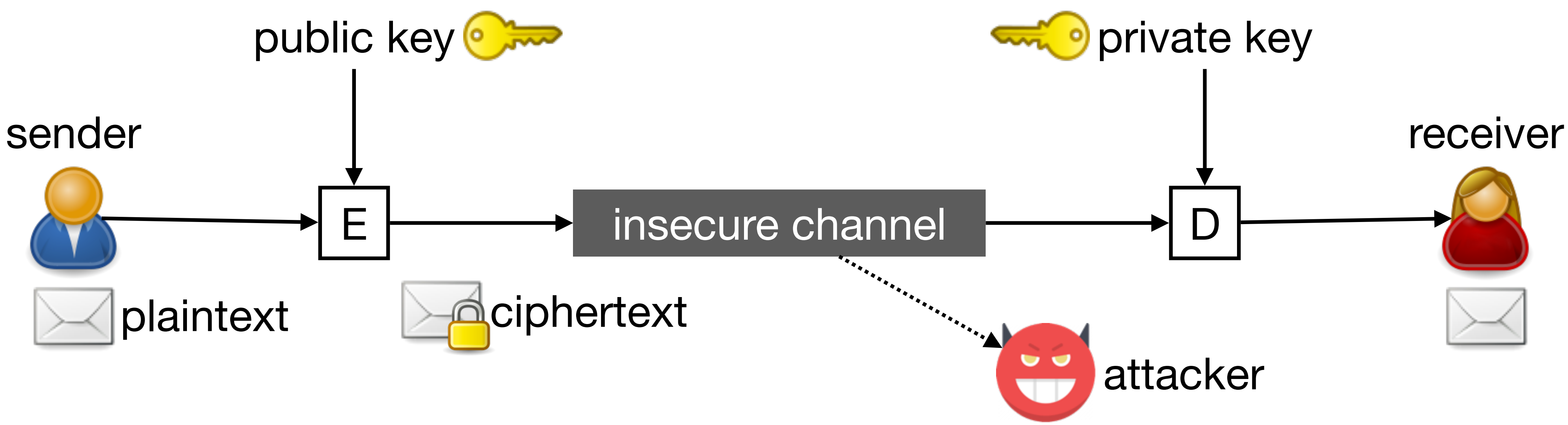
web server
(online banking,
webmail, etc.)

How or when can the two endpoints exchange a secret key?

Public-Key Cryptography

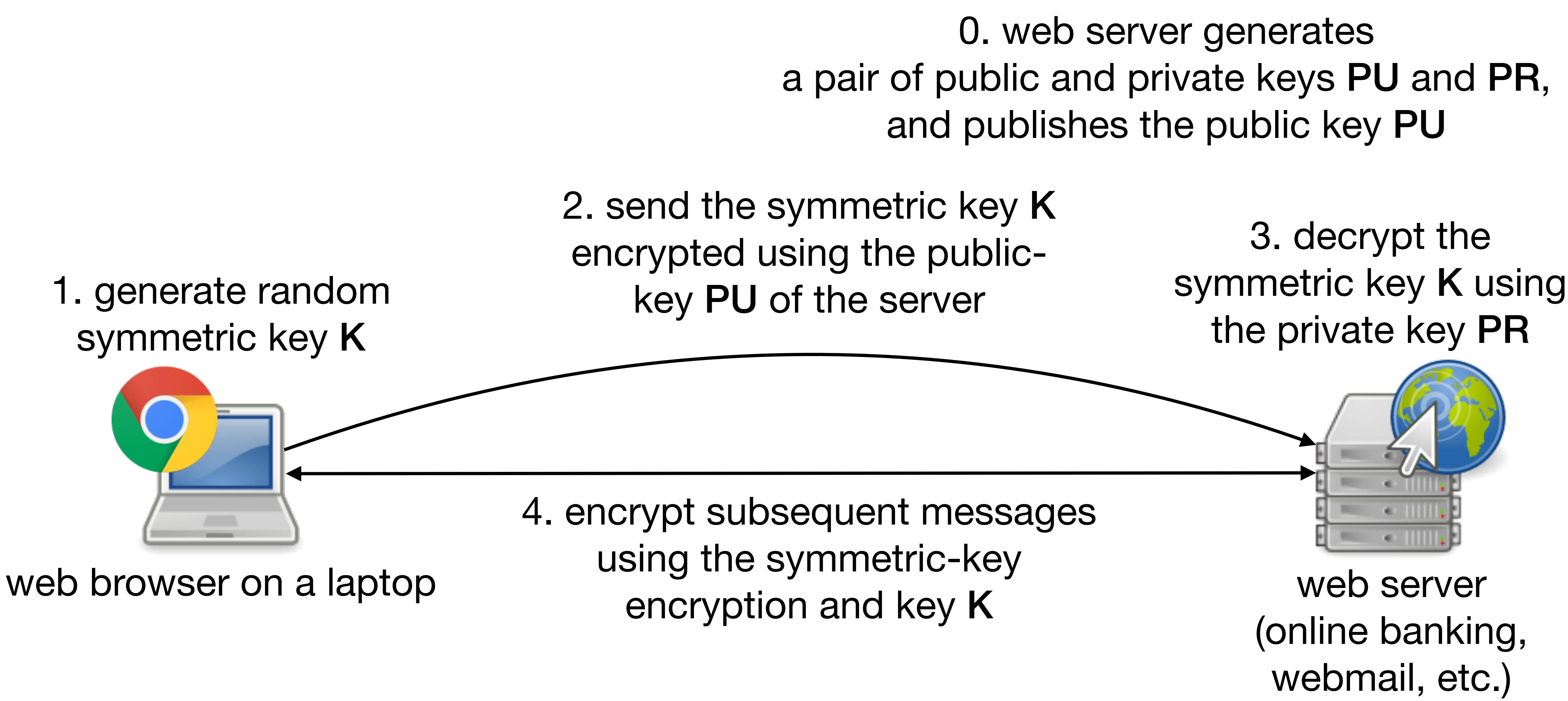
- In 1976, Whitfield Diffie and Martin Hellman proposed a fundamentally different approach to cryptography
  - first documented discovery was by British intelligence agency in 1970
- **Public-key cryptography**: instead of using a single secret key, use a pair of private and public keys
  - also called **asymmetric-key cryptography**
- Only the private key needs to be secret, the public key does not
- Public-key cryptography solves multiple problems
  - public-key encryption → key exchange
  - digital signatures → non-repudiation

Public-Key Encryption



- Everyone knows the public key → sender can encrypt
- Receiver knows the private key → receiver can decrypt
- Attacker does not know the private key → attacker cannot decrypt
- Public key can be published
  - attacker may know the public key

Public-Key Encryption
Application Example



- Secure against **passive attacks**

Public-Key Encryption Scheme

- A public-key encryption system is a triplet of algorithms **(G, E, D)**
  - **Key generation G()**: randomized algorithm, outputs **(PU, PR)**
  - Encryption **E(PU, M)**:
    takes public key **PU** and plaintext **M**, outputs ciphertext **C**
  - Decryption **D(PR, C)**:
    takes private key **PR** and ciphertext **C**, outputs plaintext **M**
- Requirements
  - for every **(PU, PR)** that was output by **G**, **D(PR, E(PU, M)) = M**
  - **G** is efficiently computable, **E** is efficiently computable given **PU** and **M**, and **D** is efficiently computable given **PR** and **C**
  - given only **PU** and **C**, an attacker cannot efficiently compute **M**

Symmetric vs. Asymmetric-Key Encryption in Practice

	Symmetric-key encryption	Asymmetric-key encryption
Typical design	series of substitutions and permutations	hard mathematical problems
Key	completely random	special structure, expensive to generate
Recommended key size	128 - 256 bits	2048 - 15360 bits
Performance	fast	slow

RSA Encryption

RSA Cryptosystem

- Developed in 1977 by Ron **R**ivest, Adi **S**hamir, and Len **A**dleman
  - in 1973, Clifford Cocks, an English mathematician working for a British intelligence agency, described an equivalent system (however, this was classified until 1997)
- For their work on public-key cryptography, Rivest, Shamir, and Adleman received a Turing Award in 2002
- One of the most widely accepted and implemented general-purpose approach to public-key encryption
- Idea
  - represent fixed-length plaintext **M** and ciphertext **C** as numbers
  - encryption: $\mathbf{C = M^e \bmod n}$
  - decryption: $\mathbf{M = C^d \bmod n}$, where private key **d** is such that $\mathbf{(M^e)^d = M \bmod n}$

RSA Mathematical Background

- Prime: an integer $p > 1$ is a prime number if its only positive divisors are 1 and p
- Greatest common divisor: $gcd(a, b)$ of integers a and b is the largest positive integer c that is a divisor of both a and b
    - a and b are relatively prime if $gcd(a, b) = 1$
    - if a and m are relatively prime, then a has a multiplicative inverse a^{-1} in modulo m
- Integer factorization problem:
  decompose a non-prime number into a product of smaller integers
    - widely believed to be a **computationally hard problem**
      (cannot be solved efficiently, i.e., in polynomial time)
    - however, this hardness has not been proven

RSA Key Generation

1. pick two large and random prime numbers p and q, $p \neq q$
2. calculate $n = p \cdot q$
3. calculate Euler's totient function $\phi(n) = (p - 1) \cdot (q - 1)$
4. pick e such that $gcd(e, \phi(n)) = 1$ and $1 < e < \phi(n)$
5. calculate d, so that $d \cdot e = 1 \bmod \phi(n)$
   (d is the multiplicative inverse e^{-1} of e in $\bmod \phi(n)$)
6. let the public key be $PU = (e, n)$
7. let the private key be $PR = (d, n)$

RSA Encryption and Decryption

- Encryption:
  given plaintext **M** (**M** < **n**), the ciphertext is **C = M^e mod n**
- Decryption:
  given ciphertext **C** (**C** < **n**), the plaintext is **M = C^d mod n**
- Consistency proof:

$$C^d \bmod n = (M^e)^d \bmod n = M^{(e \cdot d)} \bmod n$$

since **$d \cdot e = 1 \bmod \phi(n)$**, we have that **$d \cdot e = 1 + \phi(n) \cdot i$**, where **$i$** is some integer

$$C^d \bmod n = M^{(1 + \phi(n) \cdot i)} \bmod n = M \cdot M^{(\phi(n) \cdot i)} \bmod n$$

$$= M \cdot (M^{\phi(n)})^i \bmod n = M \cdot 1^i \bmod n = M \bmod n$$

Euler's theorem: if **a** and **n** are relatively prime, then **$a^{\phi(n)} = 1 \bmod n$**

RSA Example

- Key generation
  - pick two prime numbers, $p = 17$ and $q = 11$
  - calculate $n = p \cdot q = 17 \cdot 11 = 187$
  - calculate $\phi(n) = (p - 1)(q - 1) = 16 \cdot 10 = 160$
  - pick $e = 7$, which satisfies $1 = \gcd(e, \phi(n)) = \gcd(7, 160)$
  - calculate $d = 23$, so that $1 = d \cdot e = 23 \cdot 7 = 161 = 1 \bmod \phi(n)$
- Encryption
  - given plaintext $M = 88$, the ciphertext is $88^7 = 11 \bmod 187$
    (we can compute it as $88^4 \cdot 88^2 \cdot 88 \bmod 187$)
- Decryption
  - given ciphertext $C = 11$ and $d = 23$, the plaintext is
    $$11^{23} = 88^{7 \cdot 23} = 88^{161} = 88 \cdot 88^{160} = 88 \bmod 187$$

Security of RSA

- **RSA assumption**:
  given a public key (n, e) generated at random and a ciphertext C chosen at random, the probability of an attacker finding $C^{1/e}$ mod n using an efficient algorithm is negligible
- Most efficient known method is factoring the modulus n into p and q, and then computing d such that $d \cdot e = 1 \bmod \phi(n)$
  - hence, finding an RSA private key is at least as easy as integer factorization
- We do not know
  - if finding an RSA private key is at least as hard as integer factorization (it is probably easier)
  - if integer factorization is actually hard (it is suspected to be hard)

RSA Factoring Challenge

- RSA Laboratories published a list of RSA moduli in 1991

Number of Bits	Number of Decimal Digits	Year Achieved
330	100	1991
576	174	2003
640	193	2005
768	232	2009

- According to NIST, 15360-bit RSA keys are equivalent to 256-bit symmetric keys in strength

RSA Conclusion

- Security
  - best known attack (if implemented properly): **integer factorization** of modulus **n**
  - 768-bit keys have been broken, 1024-bit keys might become breakable soon
  - comparable symmetric-key security (*e.g.*, AES)

Symmetric (*e.g.*, AES)	RSA
80 bits	1024 bits
128 bits	3072 bits
256 bits	15360 bits

- Efficiency: **very slow**

→ use it to encrypt a secret key, and then switch to symmetric-key encryption

ElGamal Encryption

ElGamal Encryption

- Proposed in 1984 by Taher **Elgamal**
- Developed from the public-key cryptographic key exchange proposed by Diffie and Hellman in 1976
- Security is based on the **difficulty of computing discrete logarithms**
  - discrete logarithm problem: given **g**, **y**, and **p**, find an **x** that satisfies

$$y = g^{x} \bmod p$$

  - widely believed to be a computationally hard problem

ElGamal Key Generation

1. pick a large prime **q**
2. pick an integer $\alpha < q$ such that α is a primitive root* of **q**
3. pick an integer X such that $1 < X < q - 1$
4. compute $Y = \alpha^X \bmod q$
5. let the public key be $PU = (q, \alpha, Y)$
6. let the private key be $PR = (q, \alpha, X)$

* α is a primitive root if $\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{(q-1)}$ are different mod q

ElGamal Encryption and Decryption

Key generation:

- public key **PU = (q, α, Y)**
- private key be **PR = (q, α, X)**

where **$Y = \alpha^X \bmod q$**

- Encryption: given plaintext M ($M < q$),
  1. pick a random integer k such that $0 < k < q - 1$
  2. compute $K = Y^k \bmod q$
  3. let the ciphertext be (C_1, C_2), where
     $C_1 = \alpha^k \bmod q$
     $C_2 = K \cdot M \bmod q$
- Decryption: given ciphertext (C_1, C_2),
  1. compute $K = C_1^X \bmod q$
  2. compute $M = C_2 \cdot K^{-1} \bmod q$
- Consistency: $K = C_1^X = (\alpha^k)^X = (\alpha^X)^k = Y^k = K \bmod q$

ElGamal Example

- Key generation
  - pick prime $q = 19$, primitive root $\alpha = 10$, and integer $X = 5$
  - compute $Y = \alpha^X = 10^5 = 100000 = 3 \bmod 19$
- Encryption: given plaintext $M = 17$
  - pick $k = 6$ and compute $K = Y^k = 3^6 = 729 = 7 \bmod 19$
  - compute $C_1 = \alpha^k = 10^6 = 1000000 = 11 \bmod 19$
  - compute $C_2 = K \cdot M = 7 \cdot 17 = 119 = 5 \bmod 19$
- Decryption
  - compute $K = C_1{}^X = 11^5 = 161051 = 7 \bmod 19$
  - compute $K^{-1} = 7^{-1} = 11 \bmod 19$
  - compute $M = C_2 \cdot K^{-1} = 5 \cdot 11 = 55 = 17 \bmod 19$

ElGamal Security and Efficiency

- Computing discrete logarithm is widely believed to be a computationally hard problem
  - **recovering private key X**:
    requires computing the logarithm of **Y** to base α in modulo **q**
  - **recovering factor k:**
    requires computing the logarithm of $\mathbf{C_1}$ to base α in modulo **q**
- Efficiency
  - ciphertext is twice as long as the plaintext
  - encryption requires two exponentiations, while decryption requires only one
    → decryption is faster

Elliptic Curve Cryptography

Elliptic Curve Cryptography

- Problem with public-key cryptography based on modular arithmetic

Symmetric (*e.g.*, AES)	RSA
80 bits	1024 bits
128 bits	3072 bits
256 bits	15360 bits

  - very long keys, heavy processing load

- *Idea*: replace modular arithmetic with operations over elliptic curves

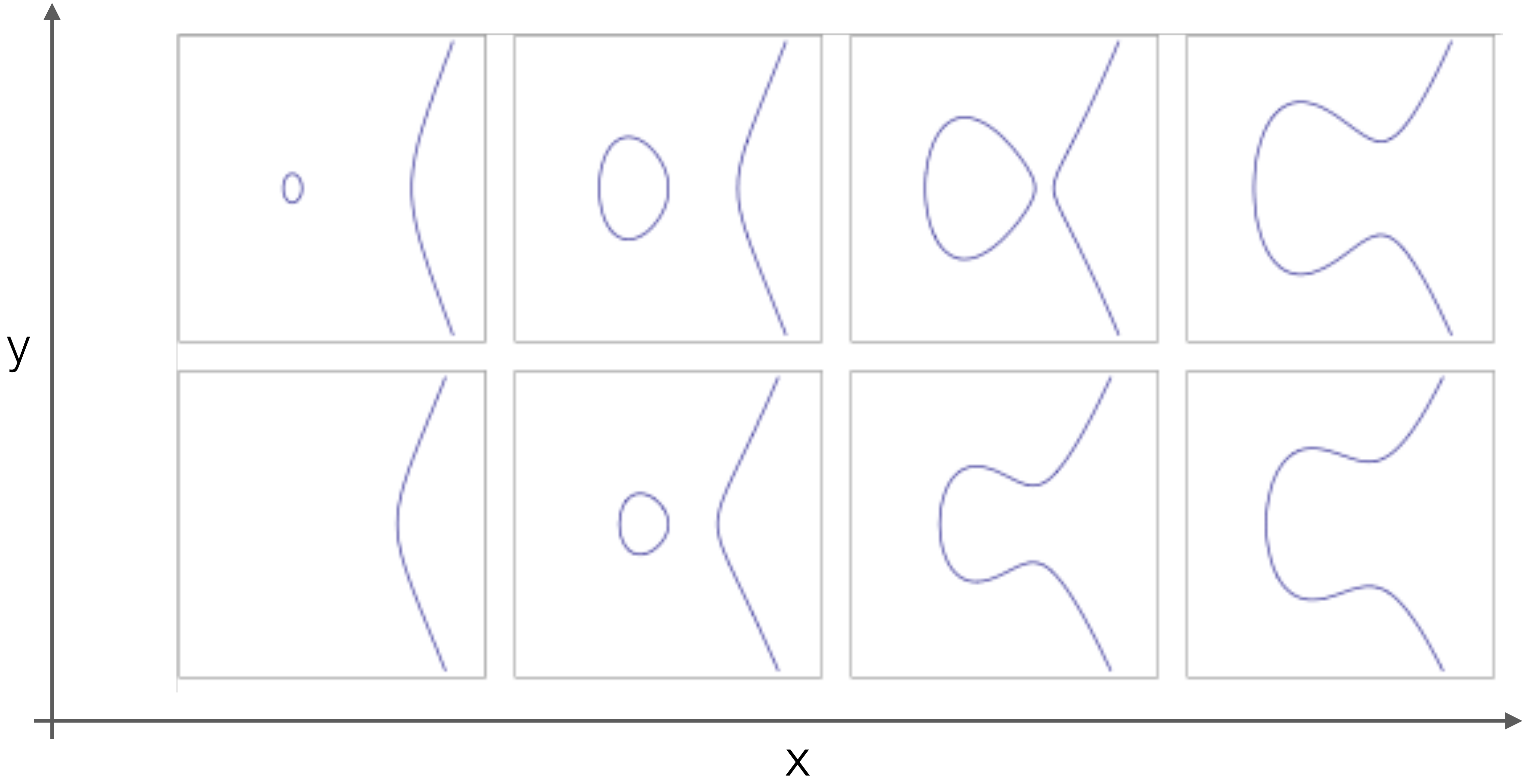
- **Elliptic Curve Cryptography** (ECC)
  - first suggested in 1985, but had not been widely used before the mid 2000s
  - 160-bit ECC key is comparable in security to a 1024-bit RSA public key
  - NIST and NSA endorsed ECC as a recommended approach, even for most classified information

Elliptic Curves

- Elements: **points (x, y)** that satisfy

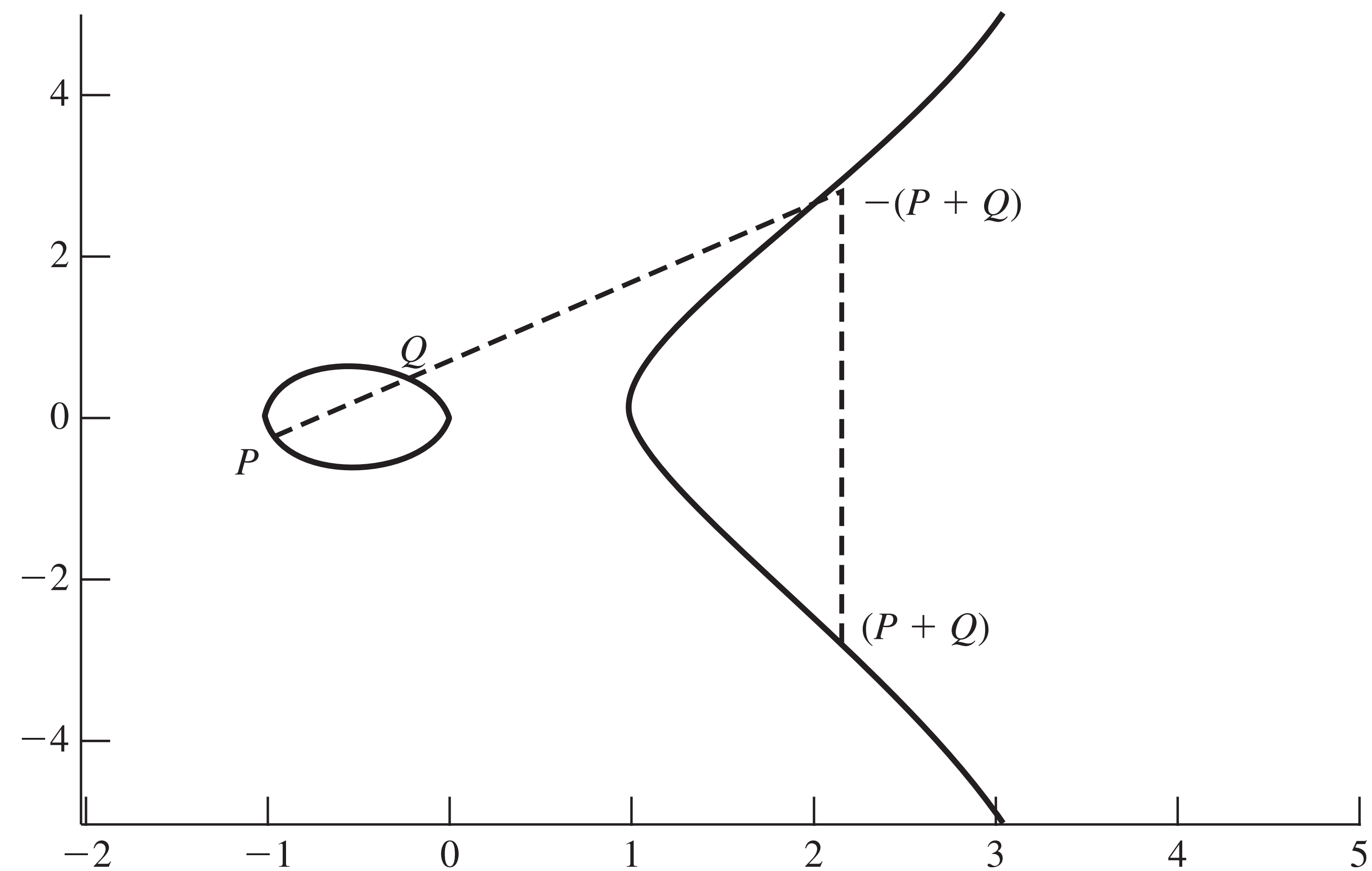
$$y^2 = x^3 + ax + b$$

where x and y are coordinates, a and b are parameters



Elliptic Curve Operation

- Operation **+**
  - **operation P + Q**: draw a line through P and Q, find the third point of intersection -(P + Q), and mirror that point vertically to get P + Q
  - **inverse element -P**: mirror point P vertically
  - **operation P + P**: draw the tangent line and find the other point of intersection, …



- Points of the elliptic curve (and a “point at infinity”) with operation + form a commutative group
  - in other words, arithmetics with this operation “works as expected”

Discrete Logarithm Problem for Elliptic Curves

- *Reminder*: with modular multiplication, it is difficult to find X such that

$$Y = \alpha^X \bmod q$$

given Y, α, and q

  - in other words, it is difficult to determine the "number of operations"

- <u>Discrete logarithm problem for elliptic curves</u>: find k such that

$$Q = k \cdot P$$

given Q and P

- where $k \cdot P = \underbrace{P + P \ldots + P}_{\text{k terms}}$

- We can "generalize" ElGamal encryption to elliptic curves in a straightforward manner

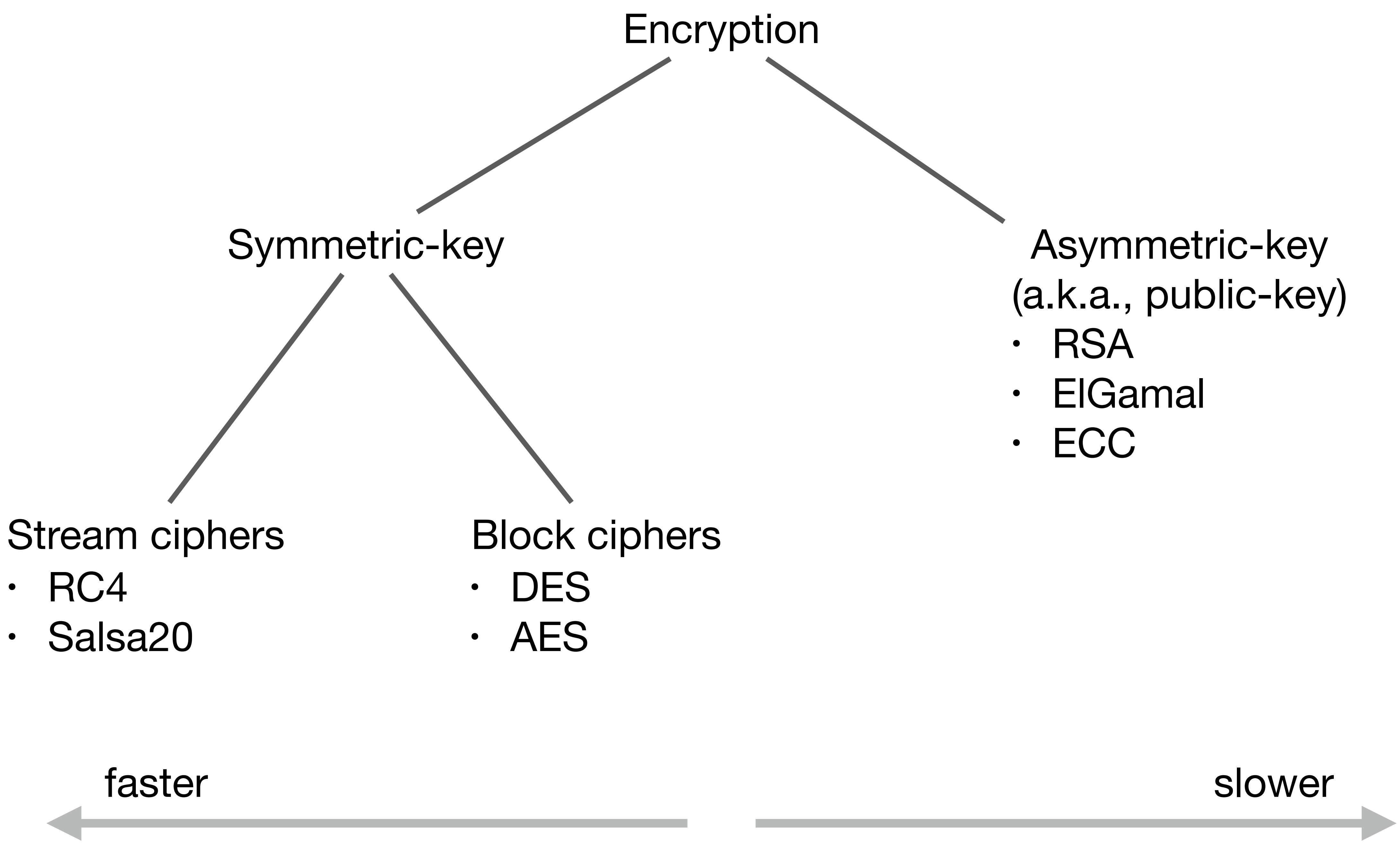
Comparison of Key Sizes

Symmetric-key algorithm	RSA	ECC
80	1024	160 - 223
112	2048	224 - 255
192	7680	384 - 511
256	15360	512+

- however, ECC might be more vulnerable to quantum computing attacks

Conclusion of Encryption

Types of Encryption



Next lecture:

Integrity