# Cryptography V: Key Exchange, Public-Key Cryptography, and Digital Signature

## CSE 565: Fall 2024 Computer Security

Xiangyu Guo (xiangyug@buffalo.edu)

University at Buffalo

# Announcement

- Please sign-up at course Piazza.
- Reminder of Quiz 0 (**Due 09/19**).
  - You must obtain full score of the Quiz.
  - Updated so that you can see exactly where you got wrong.
- Assignment 1 & Project 1 has been released (**Due 09/25, 23:59**).

# Review of Last Lecture

- MAC: protects integrity (but not confidentiality)
- Hash function: collision resistance
- Authenticated Encryption
  - Encrypt-then-MAC (recommend)
  - MAC-then-Encrypt
  - Attacks
    - Padding Oracle
    - Non-atomic decrypt
    - Do not implement A.E. by yourself! Use a standard if possible.

# Today's Topic

- Key Exchange: how we establish the shared secret key
  - Diffie-Hellman
- Public Key Cryptography
  - Trapdoor function
  - RSA
  - ~~ElGamal: PKC from Diffie-Hellman~~
  - Digital Signature

# Key Exchange

# Where are we now?

- Symmetric Key Crypto
  - Enables confidentiality
    - Achieved through secret key encryption
  - Enables authentication and integrity
    - achieved through MACs and Authenticated Encryption
- In all of the above the sender and receiver **must share a secret key**
  - Need a secure channel for key distribution
  - Not possible for parties with no prior relationship

# The Diffie-Hellman Protocol

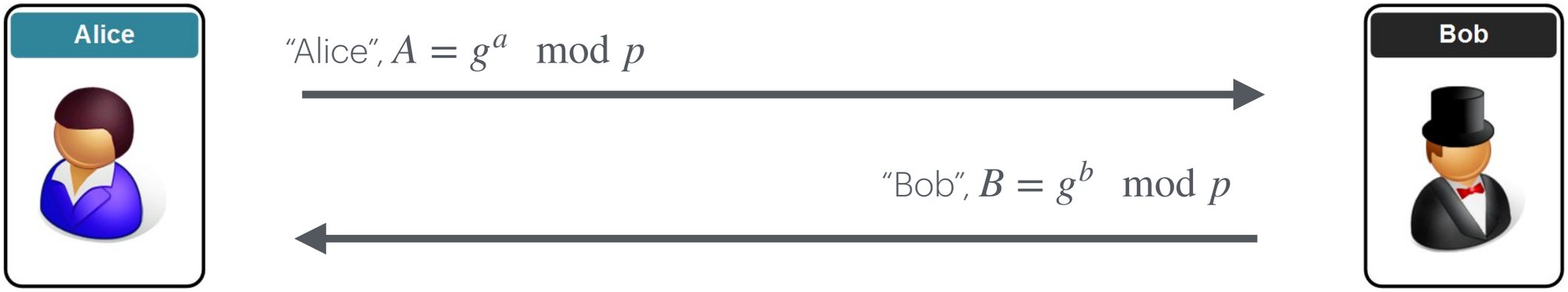
## The prime-number field version

Fix a large prime $p$ (e.g. 600 digits $\approx 2000$ bits).

Fix an integer $g$ in $\{2,\ldots,p\}$

Choose a random
$a \in \{1,\ldots,p-1\}$

Choose a random
$b \in \{1,\ldots,p-1\}$

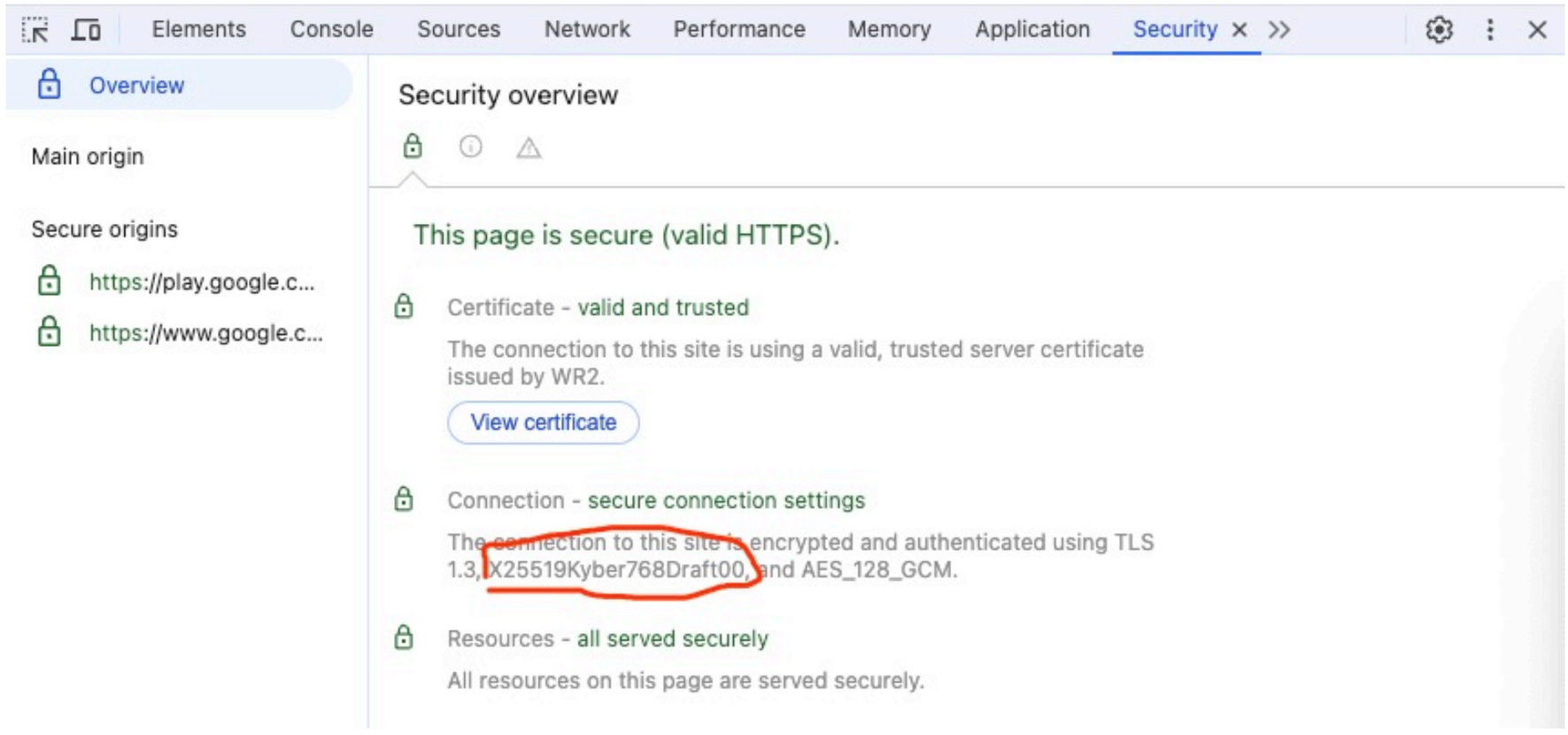



$$(g^b)^a = B^a \mod p \quad = \quad k_{AB} = g^{ab} \mod p \quad = \quad (g^a)^b = A^b \mod p$$

shared key

# The Diffie-Hellman Protocol

In Chrome's developer tool, you can find the key exchange alg used.




**X25519**: elliptic curve-based DH key exchange

Kyber768: post-quantum key exchange scheme

# Security of Diffie-Hellman

## Against Eavesdropping

- Eavesdropper sees: $p, g, A = g^a \mod p$, and $B = g^b \mod p$
  - Can compute $g^{ab} \mod p$?

- More generally, define $\text{DH}_g(g^a, g^b) = g^{ab} \mod p$
  - How hard is the DH function $\mod p$?

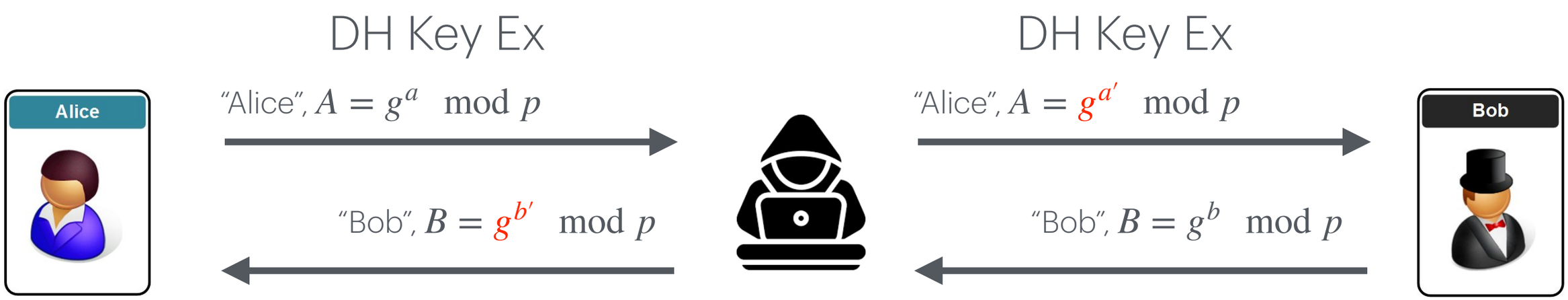
# Security of Diffie-Hellman

- Suppose the prime $p$ is $n$ bits long.
  - Best known algorithm (GNFS): $\exp\left(\tilde{O}(\sqrt[3]{n}\right)$ run time

| Cipher key size (e.g. AES) | Modulus size | Elliptic Curve size |
|---|---|---|
| 80 bits | 1024 bits | 160 bits |
| 128 bits | 3072 bits | 256 bits |
| 256 bits | *15360 bits* | 512 bits |

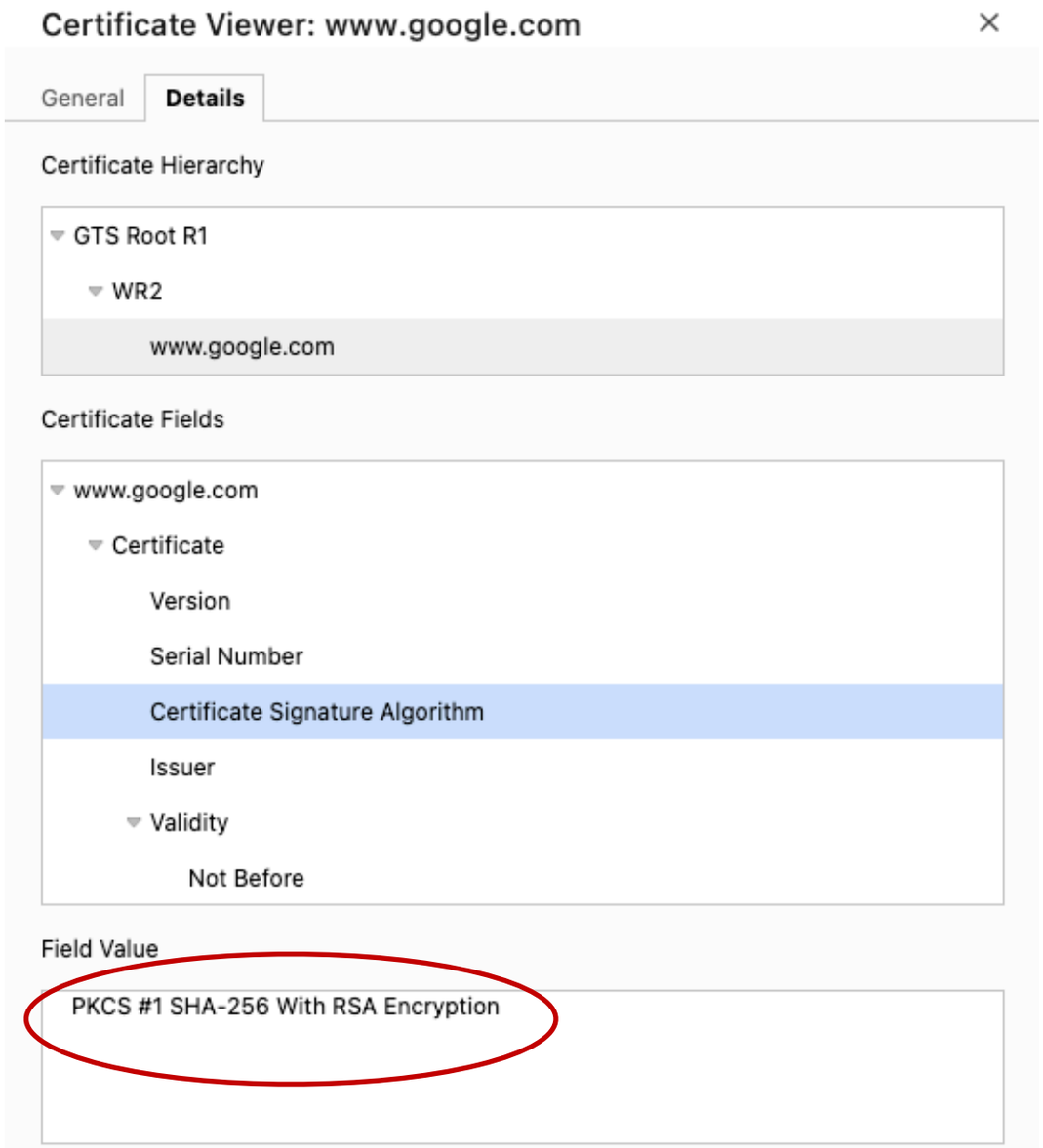
As a result: transition away from $\mod p$ to elliptic curves

# Insecure against Man-in-The-Middle



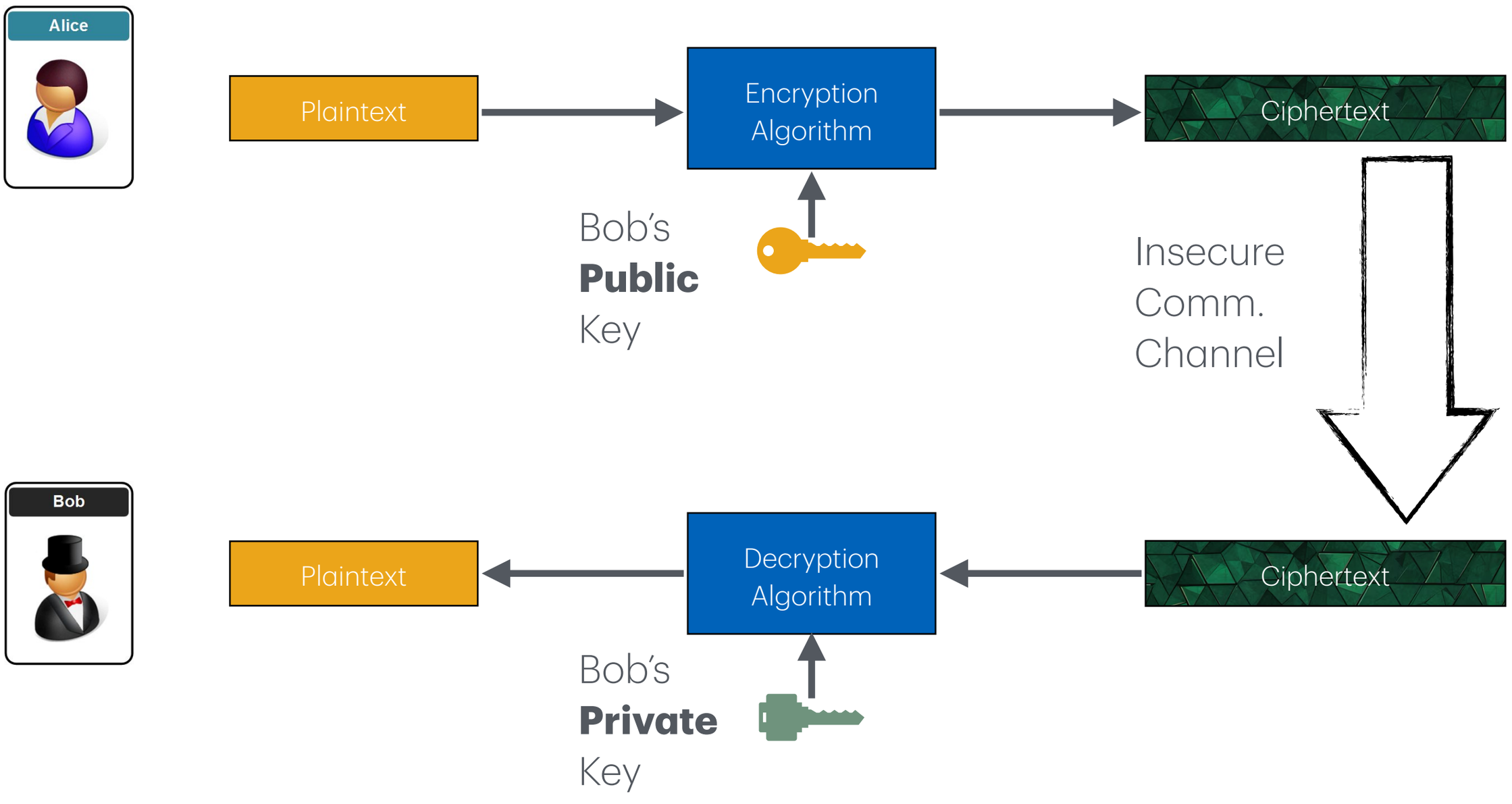
# Insecure against Man-in-The-Middle

- Diffie-Hellman on itself does *not* provide authentication
- Always used along with *digital certificates* for authentication
- Based on PKC (usually RSA)
- Need a trusted third party (CA)



# Public Key Cryptography (PKC)

# Public Key Encryption



# Public Key Encryption

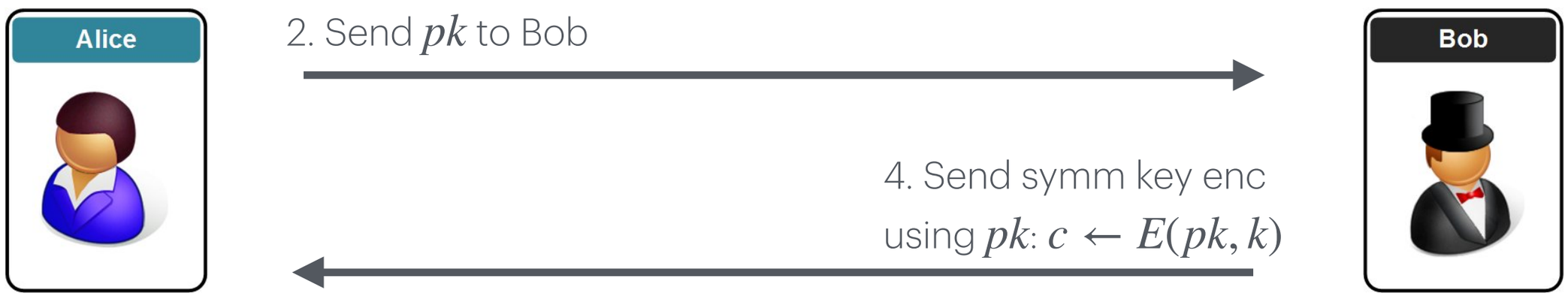
- **Definition**: A triple of algorithms $(G, E, D)$
  - Key generation: *randomized* alg

    $G() \rightarrow$ (public key $pk$, private key $sk$)
  - Enc.: *randomized* $E(pk,$ message $m) \rightarrow$ ciphertext $c$
  - Dec.: *deterministic* $D(sk, c) \rightarrow m$
- Consistency: $\forall (pk, sk)$ output by $G$:
  - $\forall$msg $m$, $D(sk, E(pk, m)) = m$

# Application: Key Exchange

1. Generate key pair
$(pk, sk) \leftarrow G()$

3. generate symm enc key $k \in K$



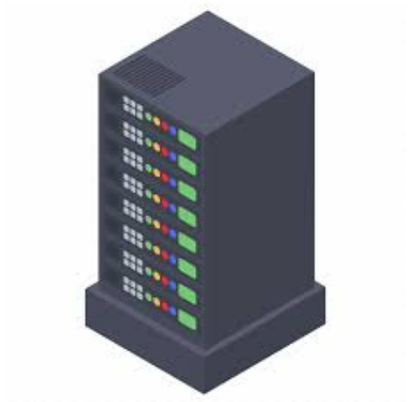

5. decrypt to get shared symm key
$k \leftarrow D(sk, c)$

# PKC Security

- Like for Symm. Enc., we would require
  - **CPA-security**: Attacker provides a pair of eq. len. msgs $(m_0, m_1)$, and the server reply with $E(pk, m_b)$. Security means Attacker *cannot* infer $b$.
    - $E$ must be randomized, otherwise this is trivially broken.

- **CCA-security**:
  - A bit complicated
  - Recall in Symm. Enc., CCA-security is implied by Ciphertext Integrity.
    - I.e. Attacker cannot fabricate new valid ciphertext
  - But in PKC, anyone knowing $pk$ can make new valid ciphertexts

# PKC Security: CCA



Server has
a secret bit
$b \in \{0,1\}$

(3) CCA phase 1:
$c_i \in C$

$m_i = D(k, c_i), i \notin \{1,\ldots,q\}$

(2) Challenge:
$m_0, m_1 \in M : |m_0| = |m_1|$

$c = E(pk, m_b)$

(3) CCA phase 2:
$c_i \in C : c_i \neq c$

$m_i = D(k, c_i), i \notin \{1,\ldots,q\}$

$b$ **?**

**CCA-secure**:
Attacker cannot infer $b$ significantly better than random guess after multiple rounds of interaction.

# PKC Security v.s. Symm Enc Security

- In symm-key settings:
  - **Ciphertext integrity**: Attacker cannot create *new* ciphertexts
  - Implies security against Chosen Ciphertext Attacks.
  - Implies authenticity.
- In public-key settings:
  - Attacker can create new ciphertexts using $pk$ !!
  - So instead: we directly require Chosen Ciphertext Security.
  - But *no* authenticity: Still vulnerable to Man-in-The-Middle, like in the case of Diffie-Hellman.

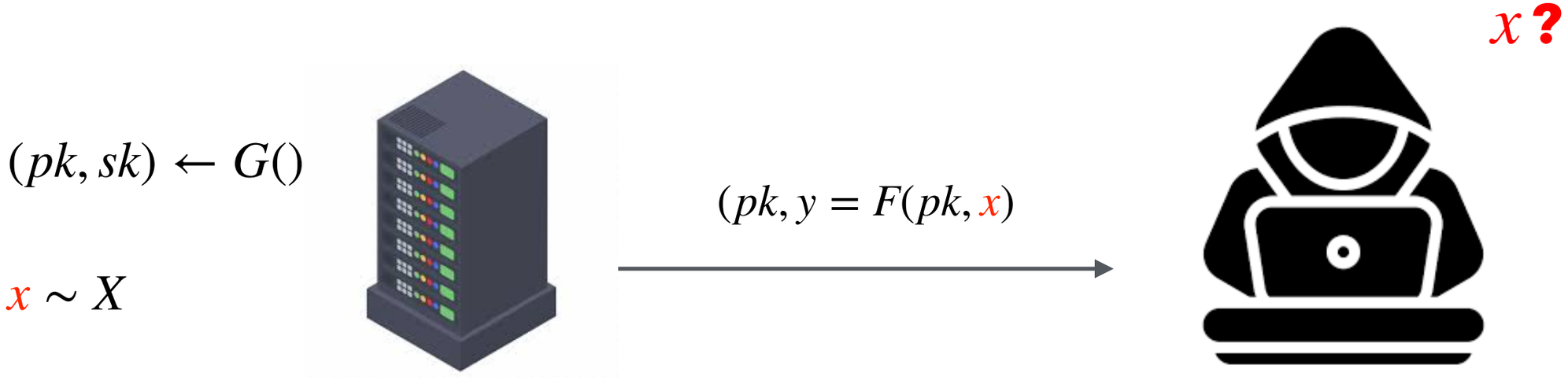
# PKC: Trapdoor Function

# Trapdoor functions (TDF)

- A trapdoor function from $X \mapsto Y$ is the triple $(G, F, F^{-1})$
  - Key generation: *randomized* $G()$ $\rightarrow$ (public key $pk$, private key $sk$)
  - $F(pk, \cdot)$: *deterministic* alg. that defines a function $X \mapsto Y$.
  - $F^{-1}(sk, \cdot)$: a function $Y \mapsto X$ that inverts $F(pk, \cdot)$
    - More specifically: $\forall (pk, sk)$ output by $G$, $\forall x \in X$, $F^{-1}(sk, F(pk, x)) = x$

# Secure Trapdoor functions (TDF)

$(G, F, F^{-1})$ is **secure** if $F(pk, \cdot)$ cannot be inverted without $sk$




The trapdoor is **secure** if any *computationally-bounded* attacker cannot infer $x$ significantly better than random guess.

# PKC Enc System from TDFs

Given

- $(G, F, F^{-1})$: secure TDF $X \mapsto Y$
- $(E_s, D_s)$: symmetric auth. encryption defined over ($K$-key, $M$-msg, $C$-ciphertext)
- $H : X \mapsto Y$ a hash function

We construct a public-key enc. system$(G, E, D)$:

- Key generation $G$: same as $G$ for TDF

# PKC Enc System from TDFs

- Given:
  - Trapdoor function $(G, F, F^{-1})$ from $X \mapsto Y$
  - symmetric cipher $S = (E_S, D_S)$, hash function $H : Y \mapsto$ key space of $S$
- Key generation: $G() \longrightarrow$ public key $pk$, private key $sk$

Encryption $E(pk, m)$:

- Choose a random $x$ from the key space of $S$
- Symm. secret key $h \leftarrow H(x)$
- Mask $x$ as $y \leftarrow F(x)$
- Output $(y, E_S(h, m))$

Decryption $D(sk, (y, c))$:

- $x \leftarrow F^{-1}(y)$
- Symm. secret key $h \leftarrow H(x)$
- Output $D_s\left(h, c\right)$

# PKC Enc System from TDFs


**Theorem**

- If $(G, F, F^{-1})$ is a secure TDF, $(E_s, D_s)$ provides auth. enc.
- and $H : X \mapsto K$ is a "random oracle" (ideal hash func)
- then $(G, E, D)$ is CCA secure (w.r.t. random oracle).

# PKC Enc System from TDFs


Why so complicated?

Why not just send $c = F(m)$ ?

- Not CPA-secure because $F$ is deterministic
- Why hash $x$ as symm. enc. key?
  - $H(\cdot)$ destroys possible undesired structure in $x$
- Efficiency?
  - Can encrypt large message $m$ outside $F$'s domain

# PKC: RSA

# Some basic number theory

Mod-$n$ arithmetic

- $a \equiv_n b$ (or $a \equiv b \mod n$) if the remainders of $a$ and $b$ divided by $n$ are equal.

- Example: $n = 15$
  - $2 \times 8 = 16 = 15 + 1 \equiv_{15} 1$
  - $4 \times 8 = 32 = 2 \times 15 + 2 \equiv_{15} 2$
  - $6 \times 8 \equiv_{15} 3$, $8 \times 8 \equiv_{15} 4$, ...

# Some basic number theory

Mod-$n$ arithmetic

- If $ab \equiv_n 1$, then $b$ is the (multiplicative) inverse of $a$ (and vice versa)
- Example: $n = 15$
  - $2 \times 8 \equiv_{15} 1 \implies$ 2 and 8 are inverse of each other
  - $3 \times ? \equiv_{15} 1 \implies$ 3 does not have an inverse!
- Fact: Given $n$, one can find any $a$'s inverse (if exists) efficiently by the Extended Euclidean Algorithm

# Some basic number theory

Mod-$n$ arithmetic

- Let $\mathbb{Z}_n := \{0,1,2,\ldots,n-1\}$, $\mathbb{Z}_n^* := \{x \in \mathbb{Z}_n : x \text{ is invertible}\}$

**Fact**: $x \in \mathbb{Z}_n^*$ if and only if $\gcd(x,n) = 1$

**Theorem (Euler)**: For any two primes $p, q$, let $n := pq$, $\phi(n) := (p-1)(q-1)$, then $\forall\, x \in \mathbb{Z}_n^*, x^{\phi(n)} \equiv_n 1$

**Corollary**: For any two primes $p, q$, let $n := pq$, $\phi(n) := (p-1)(q-1)$, then $\forall\, x \in \mathbb{Z}_n, x^{\phi(n)} \cdot x \equiv_n x$

# Some basic number theory

Mod-$n$ arithmetic

- Let $\mathbb{Z}_n := \{0,1,2,\ldots,n-1\}$, $\mathbb{Z}_n^* := \{x \in \mathbb{Z}_n : x \text{ is invertible}\}$
  - Example: $n = 15 = 3 \times 5$, $\phi(n) = (3-1) \times (5-1) = 8$
    - Euler's Thm: recall $2 \times 8 \equiv 1 \mod 15$
      - $2^8 = 256 = 15 \times 15 + 1 \equiv 1 \mod 15$
      - $8^8 = 15 * 1118481 + 1 \equiv 1 \mod 15$
    - Corollary of Euler's Thm: recall 3 has no multiplicative inverse module 15
      - $3^8 = 437 \times 15 + 6 \not\equiv 1 \mod 15$, but $3^{8+1} = 1312 \times 15 + 3 \equiv 3 \mod 15$

# The RSA Trapdoor Perm.

$(G, \text{RSA}, \text{RSA}^{-1})$

- Intuition: want pub-key $e$ and priv-key $d$ s.t.
  - $\text{RSA}(m) = m^e \mod N$
  - $\text{RSA}^{-1}(c) = c^d = m^{ed} = m \mod N$

# The RSA Trapdoor Perm.

$(G, \text{RSA}, \text{RSA}^{-1})$

- Intuition: want pub-key $e$ and priv-key $d$ s.t.
  - $\text{RSA}(m) = m^e \mod N$

Coro of Euler's Thm: $ed = \alpha\phi(N) + 1$

  - $\text{RSA}^{-1}(c) = c^d = m^{ed} = m \mod N$

- RSA Key generation $G()$:
  - pick two random primes $p, q \approx$ 1024 bits, set $N = pq$.
  - Pick two integers $e, d$ s.t. $ed \equiv_{\phi(N)} 1$
  - Output Pub-key $pk = (N, e)$, private key $sk = (N, d)$

# The RSA Trapdoor Perm.

$(G, \text{RSA}, \text{RSA}^{-1})$

- Why is it a *trapdoor*?
  - To invert RSA$(\cdot)$ (without priv-key $d$), need to solve $x$ from $c = x^e \mod N$

**Best known alg.**

- Factor $N$ to get $p, q$
- $\phi(N) \leftarrow (p-1)(q-1)$
- Find $d$ as $e$'s inverse (mod $\phi(N)$) using Ext. Euclidean Alg
- $x \leftarrow c^d \mod N$

# The RSA Trapdoor Perm.

$(G, \text{RSA}, \text{RSA}^{-1})$

- Why is it a *trapdoor*?
  - To invert RSA$(\cdot)$ (without priv-key $d$), need to solve $x$ from $c = x^e \mod N$ (discrete root)

**Best known alg.**

- Factor $N$ to get $p, q$ ⟵ believed to be hard for large $N$
- $\phi(N) \leftarrow (p-1)(q-1)$
- Find $d$ as $e$'s inverse (mod $\phi(N)$) using Ext. Euclidean Alg
- $x \leftarrow c^d \mod N$

Is there a / no shortcut without factoring $N$?

**Ans**: unknown.

# The RSA Public-key Enc. (ISO)

- Given:
  - symmetric cipher $S = (E_S, D_S)$,
  - hash function $H : \mathbb{Z}_N \mapsto$ key space of $S$
- Key generation $G()$: The RSA trapdoor key generation
  - Output pub-key $pk = (N, e)$, private key $sk = (N, d)$

Encryption $E(pk, m)$:

- Choose random $x$ from $\mathbb{Z}_N$
- $y \leftarrow \text{RSA}(x) = x^e \mod N$
- symm. enc. key $h \leftarrow H(x)$
- Output $(y, E_S(h, m))$

Decryption $D(sk, (y, c))$:

- $x \leftarrow \text{RSA}^{-1}(y)$
- Symm enc key $h \leftarrow H(x)$
- Output $D_s(h, c)$

# PKC: RSA in practice

# Rule of Thumb

- Never use the RSA trapdoor function directly as encryption scheme
  - Easily breakable by CCA (how?)
- Do not implement the alg by yourself.
  - So many ways to go wrong.
  - Use well-established libraries, e.g. OpenSSL, libsodium, etc.

# Parameter choices

- $N$ should be at least 2048 bits, better 4096 bits
  - $p, q \sim$ at least 1024 bits
  - Use small $e$ to speed up encryption: $e = 3, 5, 65537$
  - Never use a small $d$:
    - Very small $d$: attacker can enumerate it
    - Moderatly small: $d < N^{0.292}$ is still insecure [Boneh-Durfee'98]

# Padding

- Recall the RSA PKC encryption:

Encryption $E(pk, m)$:

- Choose random $x$ from $\mathbb{Z}_N$
- $y \leftarrow \text{RSA}(x) = x^e \mod N$
- symm. enc. key $h \leftarrow H(x)$
- Output $(y, E_S(h, m))$

- We *first* sample a $x$ from the RSA TDF domain (e.g. 2048 bits int when $|N| = 2048$).

- *Then* the key $h$ for symm enc is obtained by hashing $x$

# Padding

- In practice the order is often reversed:
  - We first generate the key $h$ for symm. enc. (e.g., 128 bits for AES-128)
  - Then $h$ is masked using TDF and sent along with encrypted msg.

> Encryption $E(pk, m)$:
>
> - Choose random symm enc key
> - $y \leftarrow \text{RSA}(h) = h^e \mod N$
> - Output $(y, E_S(h, m))$

Problem: $h$ is usually too smaller than $N$ (e.g., 128 bits vs 2048 bits)

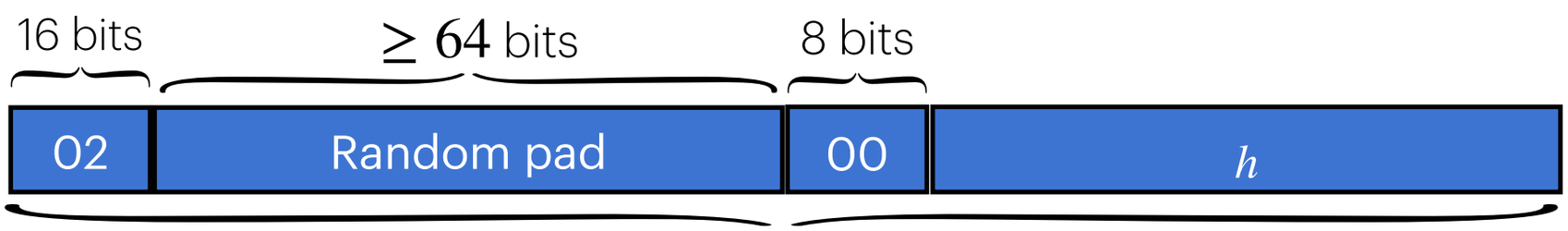
# Padding

- Example:
  - Suppose $e = 3$, then for a small $h$, it's likely that $h^e < N$
  - The attacker can get $h$ by computing $y^{1/3}$ (*without mod* $N$, so this is just a regular cubic root calculation)

# Padding

- Solution: **Pad** some bits to $h$ to make it 2048 bits long

PKCS1 v1.5 Padding scheme:




Bit size of N (e.g. 2048 bits)

- Widely deployed, e.g. in  HTTPS - TLS 1.2
- Unfortunately, weakness known [Bleichenbacher'1998].
- Use RSA-OAEP instead.

# PKC: ElGamal

# Other Public-Key Algorithms

- Many popular public key algorithms rely on the difficulty of *discrete logarithm problem*
  - ElGamal encryption and ElGamal signature
  - Digital Signature Algorithm (DSA)
  - Diffie-Hellman key exchange
  - . . .
- Given an appropriate setup with $g, p$ and $h = g^x \mod p$, it is difficult for someone to compute $x$
  - $x$ is called the discrete logarithm of $h$ to the base $g$
  - groups in which the discrete logarithm problem is hard use prime modulus $p$ (conventional and elliptic curve settings)

# PKC from Discrete Log

## Basic ideas

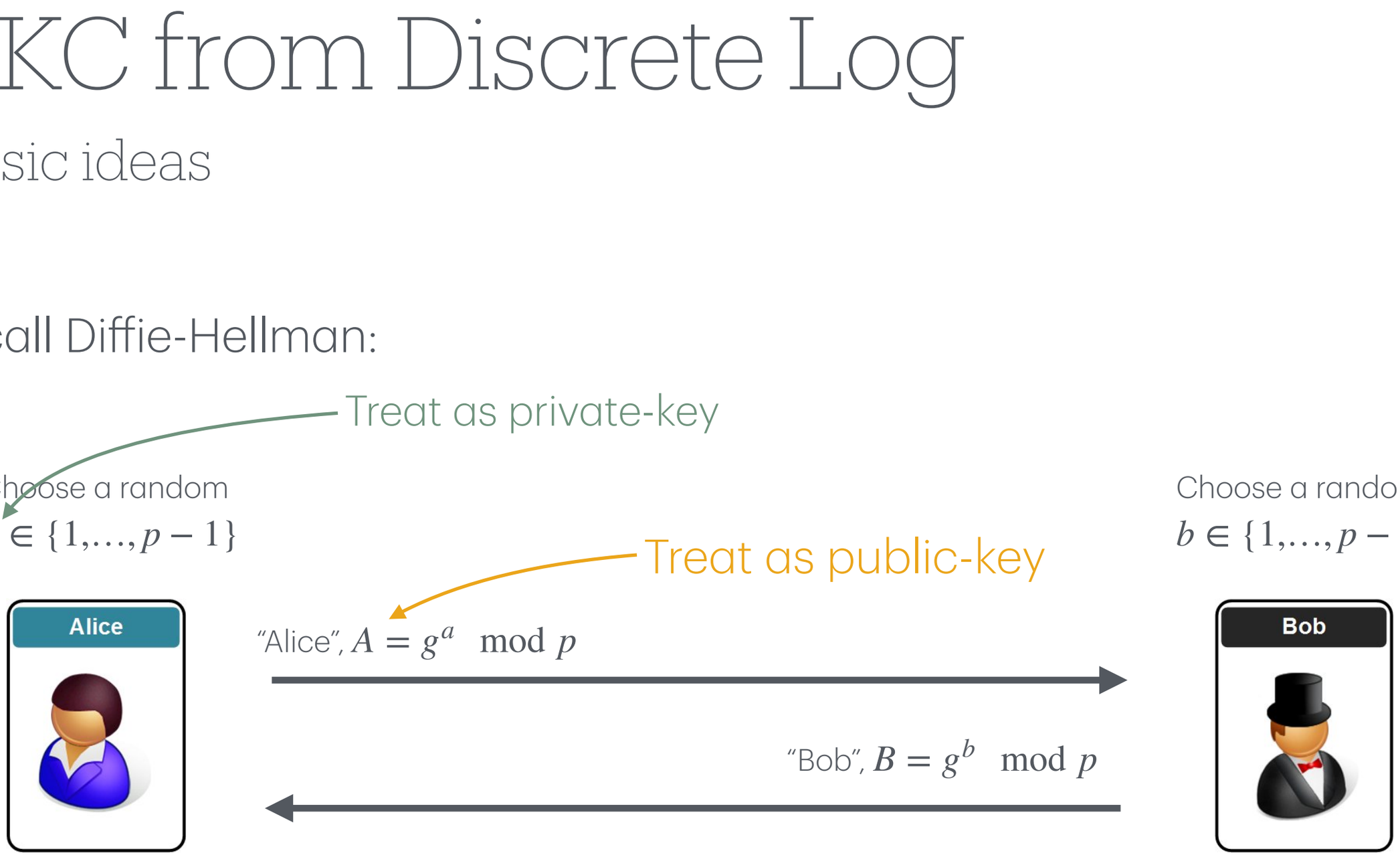
Recall Diffie-Hellman:

Treat as private-key

Choose a random
$a \in \{1,\ldots,p-1\}$

Choose a random
$b \in \{1,\ldots,p-1\}$

Treat as public-key




$$(g^b)^a = B^a \mod p \quad = \quad k_{AB} = g^{ab} \mod p \quad = \quad (g^a)^b = A^b \mod p$$

shared key

# PKC from Discrete Log

## Basic ideas

$(E_s, D_s)$: some symm. auth. enc.

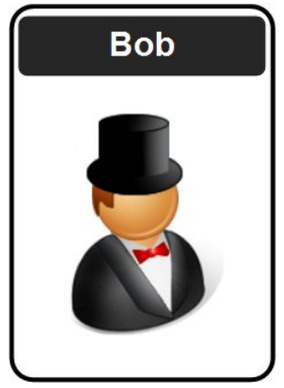
Choose a random
$sk = a \in \{1,\ldots,p-1\}$

Choose a random
$b \in \{1,\ldots,p-1\}$

$pk = A = g^a \mod p$


Compute $g^{ab} = A^b$;

Derive $k$ from $g^{ab}$




$[B = g^b, c = E_s(k, m)]$

Compute $g^{ab} = B^a$;

Derive $k$ from $g^{ab}$;

$m \leftarrow D_s(k, c)$

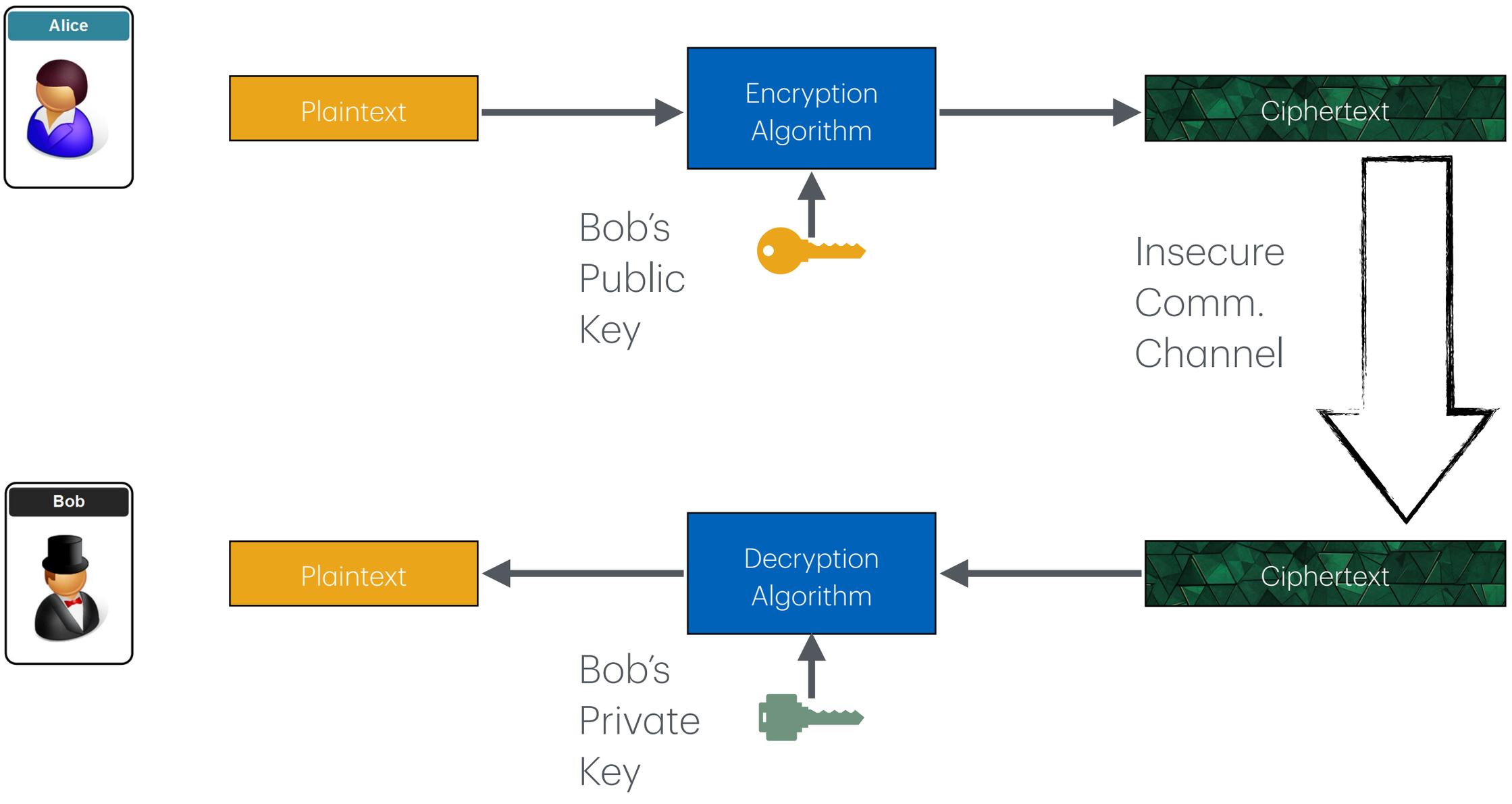
In actual impl. $g$ is also chosen randomly and published as part of the $pk$

# Symmetric vs Public-Key Encryption

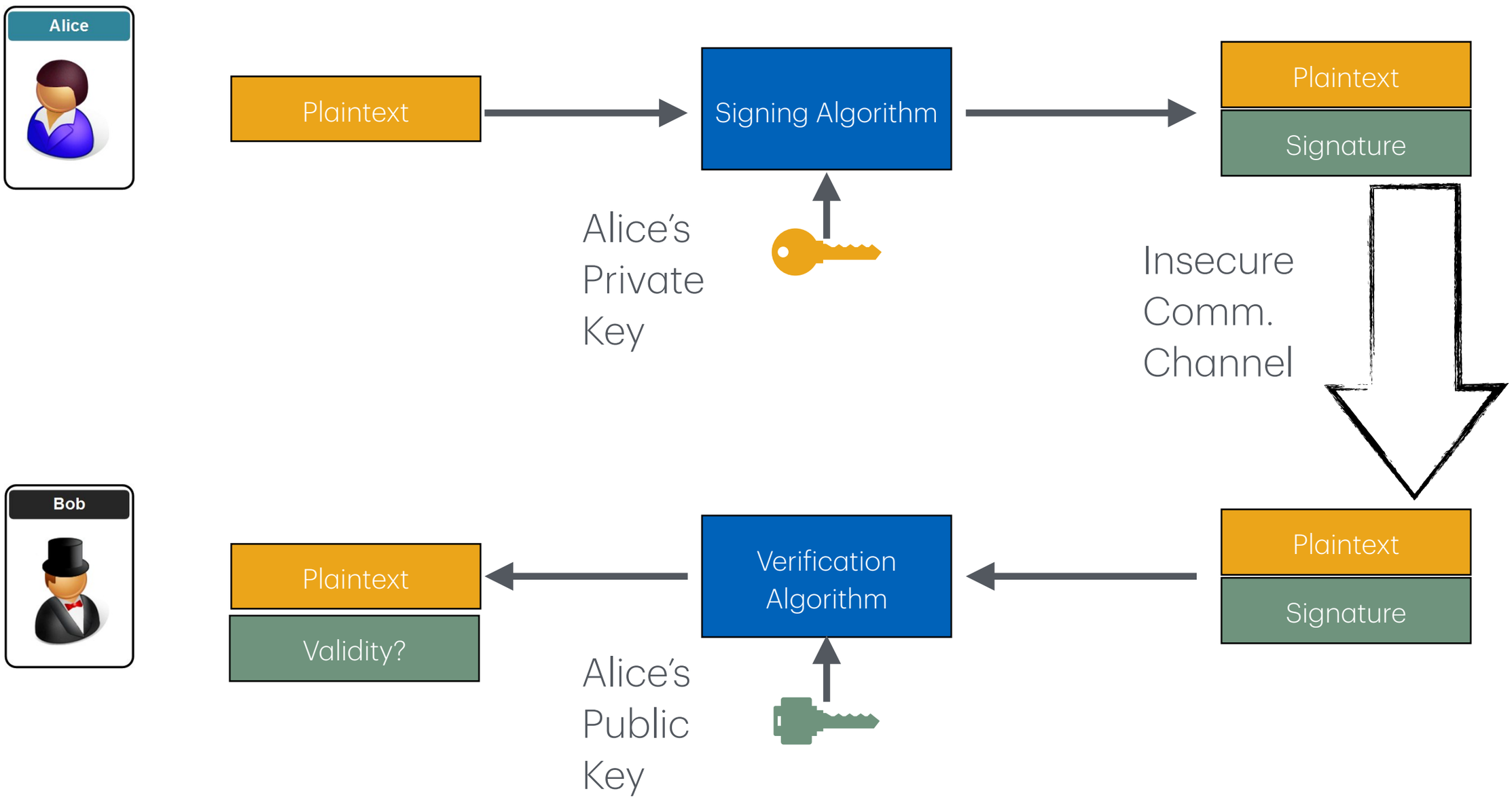
- Public-key operations are orders of magnitude slower than symmetric encryption
  - A multiplication modulo $n$ requires close to $O(\log^2 n)$ work
  - A full-size exponentiation modulo n requires close to $O(\log^3 n)$work
    - It is the cost of multiplication times the exponent size
  - Public-key encryption is typically not used to communicate large volumes of data
    - It is rather used to communicate (or agree on) a symmetric key
    - The data itself is sent encrypted with the symmetric key
- In RSA, decryption is significantly slower than encryption, with key generation being the slowest

# PKC: Digital Signature

# Public-Key Encryption



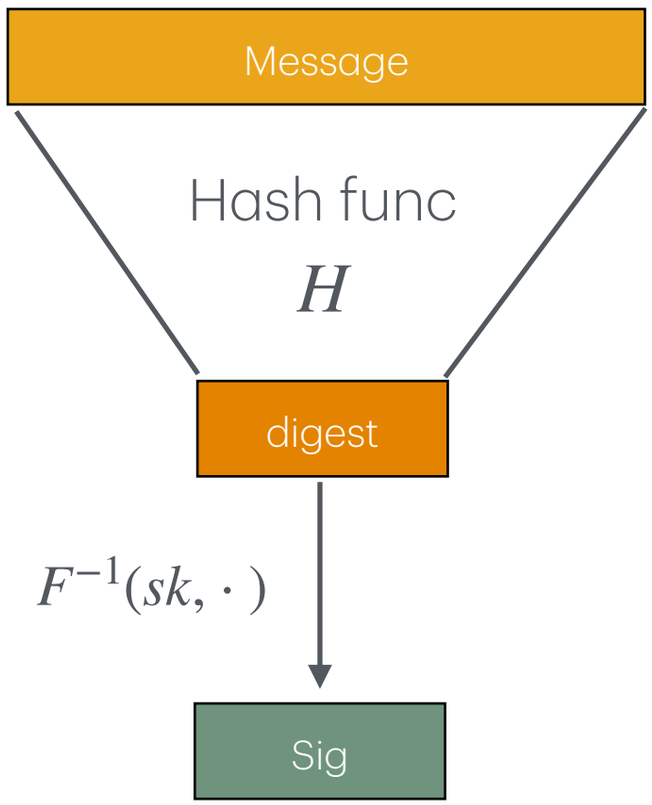
# (Public-Key) Digital Signature



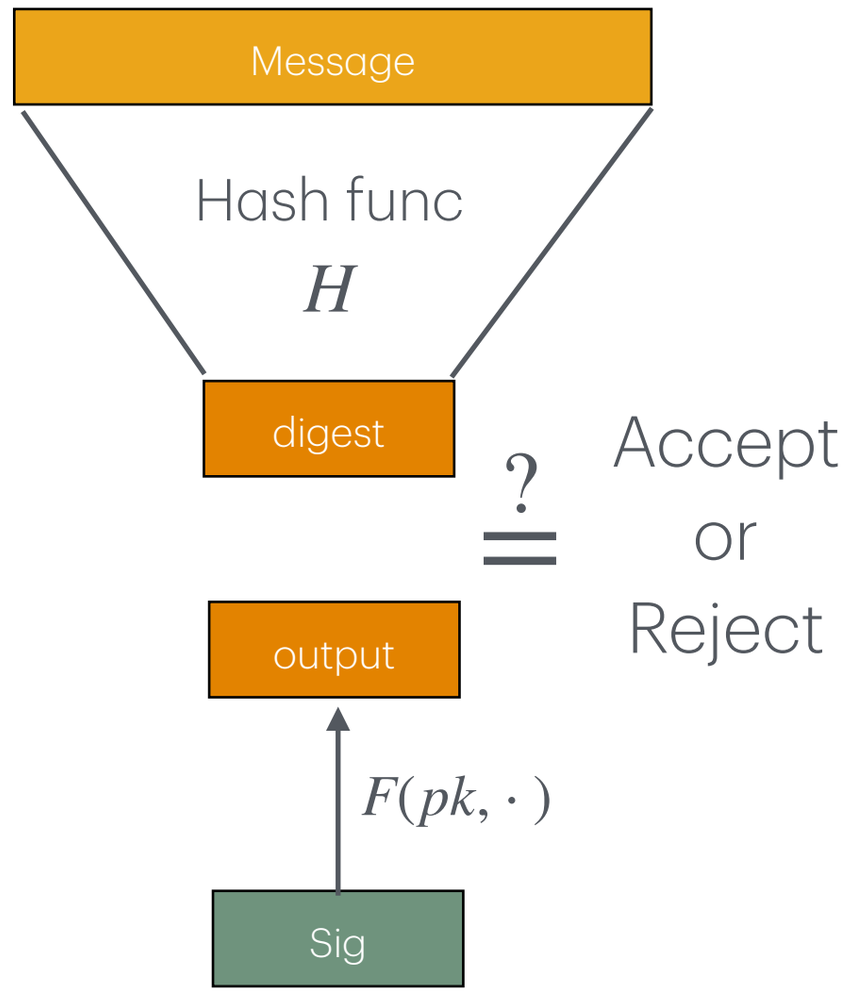
# Example: Digital Signature from TDF

Given: a secure trapdoor function $(G, F, F^{-1})$ and $(pk, sk) \leftarrow G()$

Signing alg. $S(sk, \text{msg})$




Verify alg. $V(pk, \text{msg}, \text{sig})$



# Example: Digital Signature from RSA TDF

- Given:
  - RSA trapdoor function $(G, \text{RSA}, \text{RSA}^{-1})$,
  - collision-resistant hash function $H : \mathbb{Z}_N \mapsto$ key space of $S$
- Key generation $G()$: The RSA trapdoor key generation
  - Output pub-key $pk = (N, e)$, private key $sk = (N, d)$

Signing $S(sk, m)$:

- $y \leftarrow \text{RSA}^{-1}(H(m)) = H(m)^d \mod N$

Verifying $V(pk, m, \sigma)$:

- Check if $H(m) = \text{RSA}(\sigma) = \sigma^e \mod N$

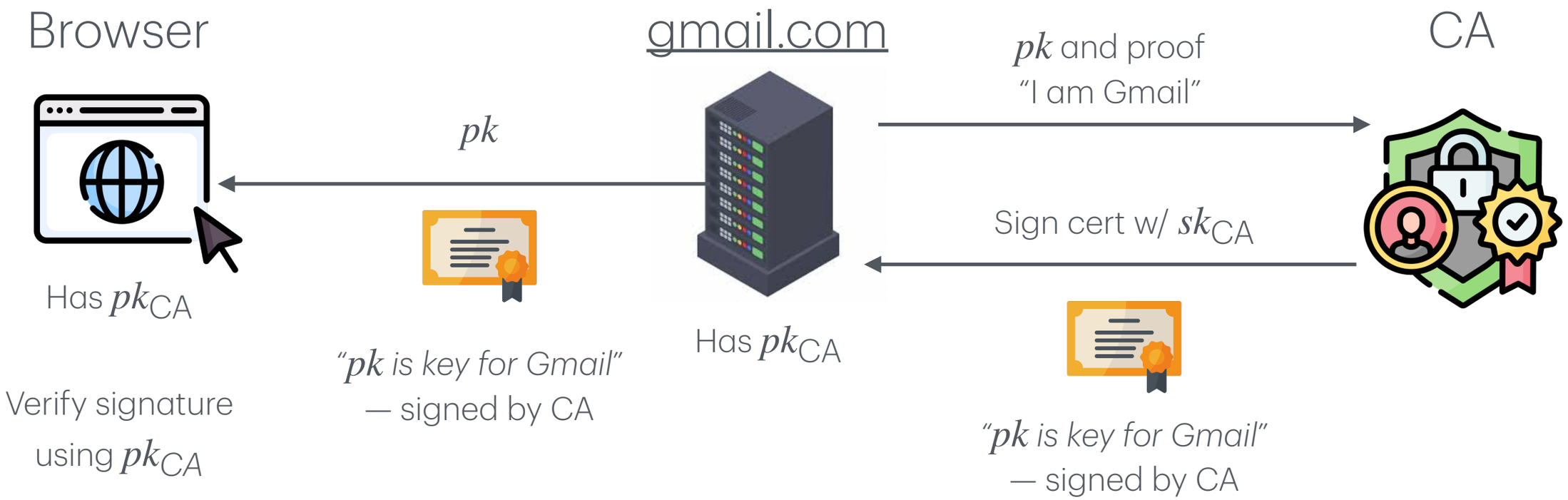
In practice $H(m)$ will be padded before usage.

# When to use signatures

- If **one** party signs and **one** party verifies: use a *MAC*
  - Often requires interaction to generate a shared key
  - Recipient can modify the data and re-sign it before passing the data to a 3rd party
- If **one** party signs and **many** parties verify: use a *signature*
  - Recipients cannot modify received data before passing data to a 3rd party (non-repudiation)

# Application: Certificate

- Problem: browser needs server's public-key to setup a session key
- Solution: server asks trusted 3rd party (CA) to sign its public-key $pk$.



# The Big Picture

| Security Goal | Symmetric Key | | Public Key |
|---|---|---|---|
| Secrecy / Confidentiality | Block ciphers with encryption modes (AES); Stream ciphers | Authenticated Encryption | Public key encryption (RSA, ElGamal, etc) |
| Authenticity / Integrity | Message Authentication Codes (CBC-MAC, HMAC) | | Digital signatures (RSA, DSA, etc) |

# Acknowledgement


- The slides of this lecture is developed heavily based on
  - Slides from Prof Dan Boneh's lecture on Cryptography (https://crypto.stanford.edu/~dabo/courses/OnlineCrypto/)
  - Slides from Prof Ziming Zhao's lecture on Computer Security (https://zzm7000.github.io/teaching/2023springcse410565/index.html)

# Questions?