CSE551:
Advanced Computer Security

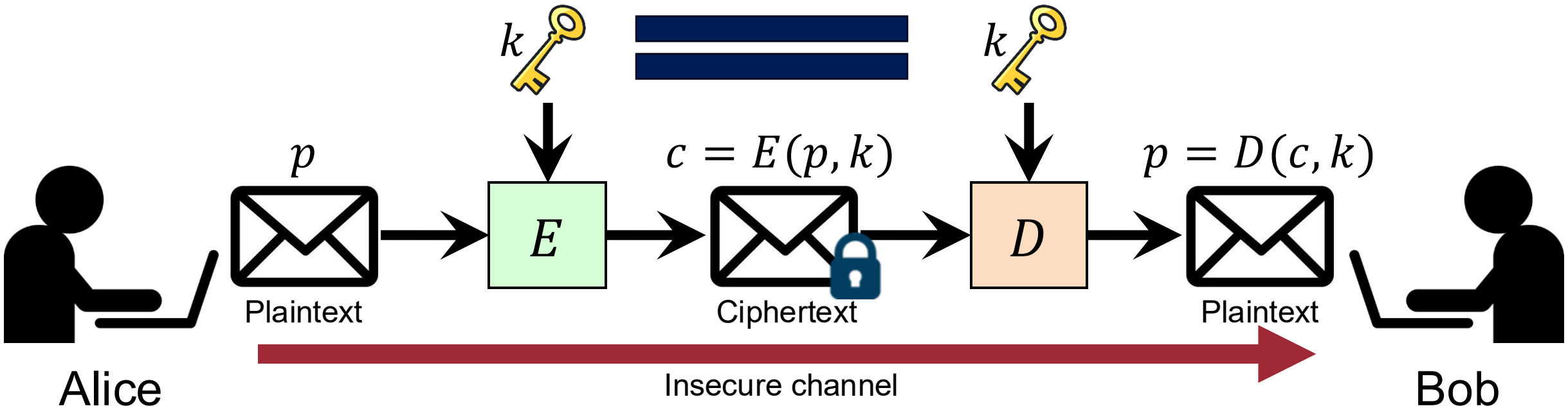
5. Public-Key Infrastructure, Integrity

Seongil Wi

Department of Computer Science and Engineering

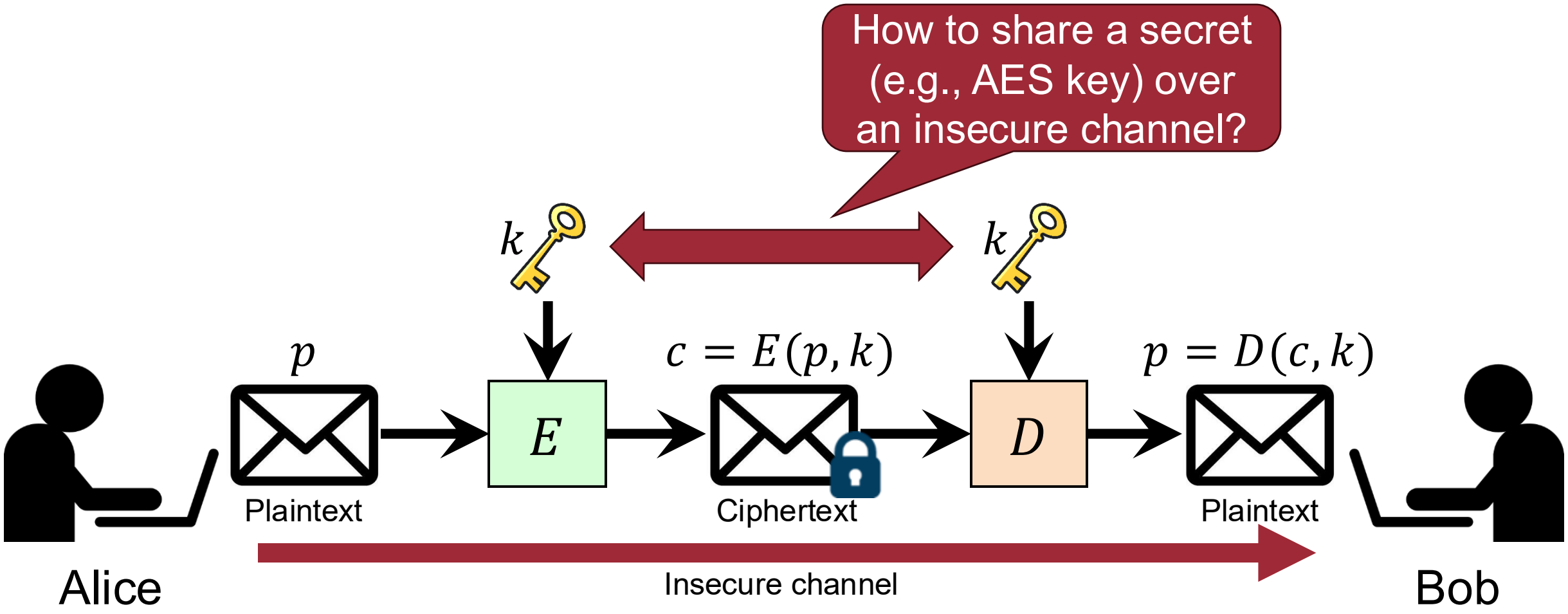
Recap: Symmetric-key Encryption

- **Symmetric**: the encryption and decryption keys *are the same*



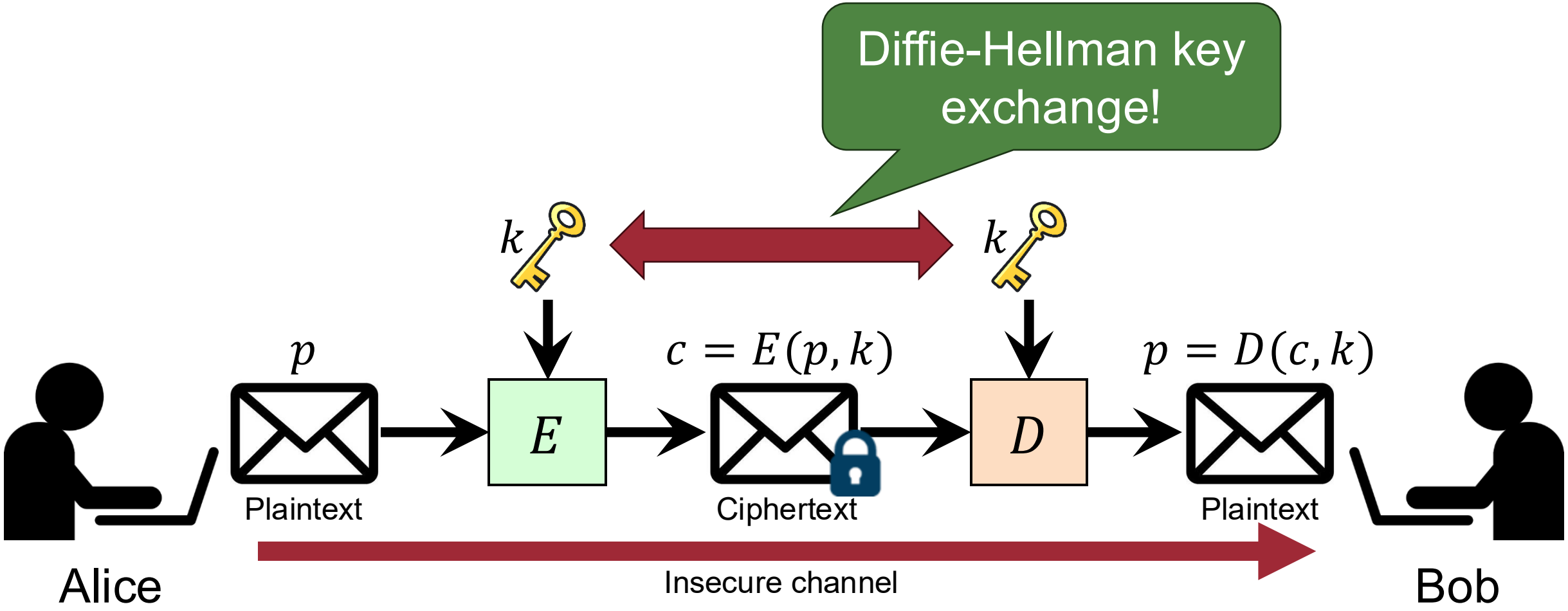
Recap: Symmetric-key Encryption

- **Symmetric**: the encryption and decryption keys *are the same*



Recap: Diffie-Hellman Key Exchange

- **Symmetric**: the encryption and decryption keys *are the same*



Recap: Diffie-Hellman Key Exchange

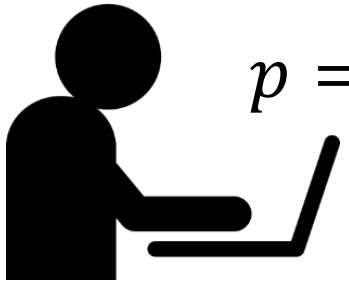
Symmetric key:

$$K = g^{ab} \bmod p$$

$p = 23, g = 9$

$A = (g^a \bmod p) = 6$

$B = (g^b \bmod p) = 16$



$\boldsymbol{a = 4}$

$p = 23, g = 9$

$A = (g^a \bmod p) = 6$

$B = (g^b \bmod p) = 16$

$\boldsymbol{b = 3}$

$p = 23, g = 9$

$A = (g^a \bmod p) = 6$

$B = (g^b \bmod p) = 16$

Alice

Insecure channel

Bob

Recap: Diffie-Hellman Key Exchange

Symmetric key:

$K = g^{ab} \bmod p$

$p = 23, g = 9$

$A = (g^a \bmod p) = 6$

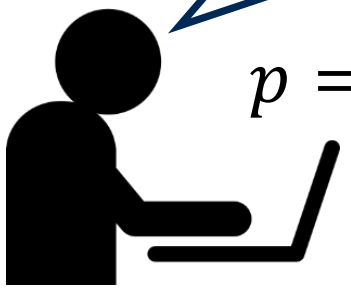
$B = (g^b \bmod p) = 16$

$K = (B^{\boldsymbol{a}} \bmod p) = (g^{ab} \bmod p)$

$= (16^4 \bmod 23) = 9$

$K = (A^{\boldsymbol{b}} \bmod p) = (g^{ab} \bmod p)$

$= (6^3 \bmod 23) = 9$



$\boldsymbol{a = 4}$

$p = 23, g = 9$

$A = (g^a \bmod p) = 6$

$B = (g^b \bmod p) = 16$

$\boldsymbol{b = 3}$

$p = 23, g = 9$

$A = (g^a \bmod p) = 6$

$B = (g^b \bmod p) = 16$

Alice

Insecure channel

Bob

Recap: Security of the Diffie-Hellman Key Exchange

Symmetric key:

$K = g^{ab} \ mod \ p$

$p = 23, g = 9$

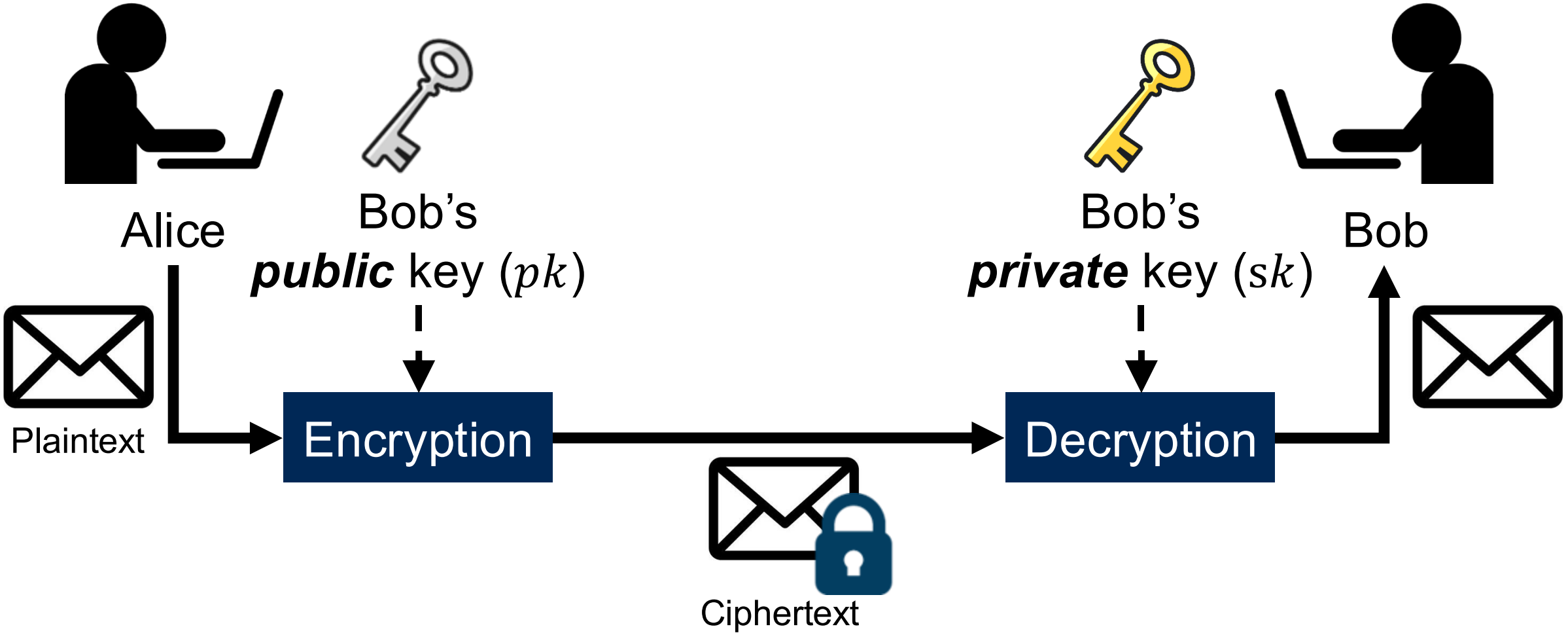
$A = (g^a \ mod \ p) = 6$

$B = (g^b \ mod \ p) = 16$

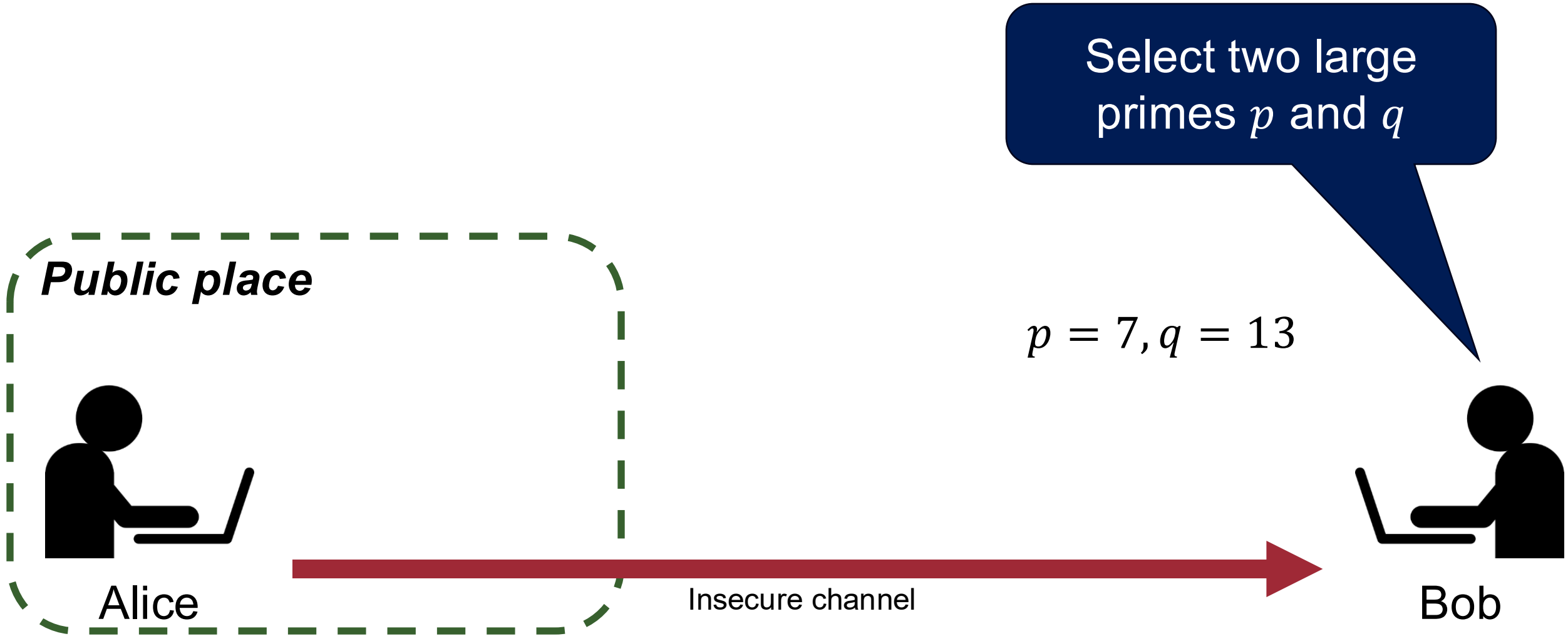
The attacker cannot efficiently compute $(g^{ab} \ mod \ p)$ ***without knowing a and b***

Recap: Asymmetric-key Cryptography

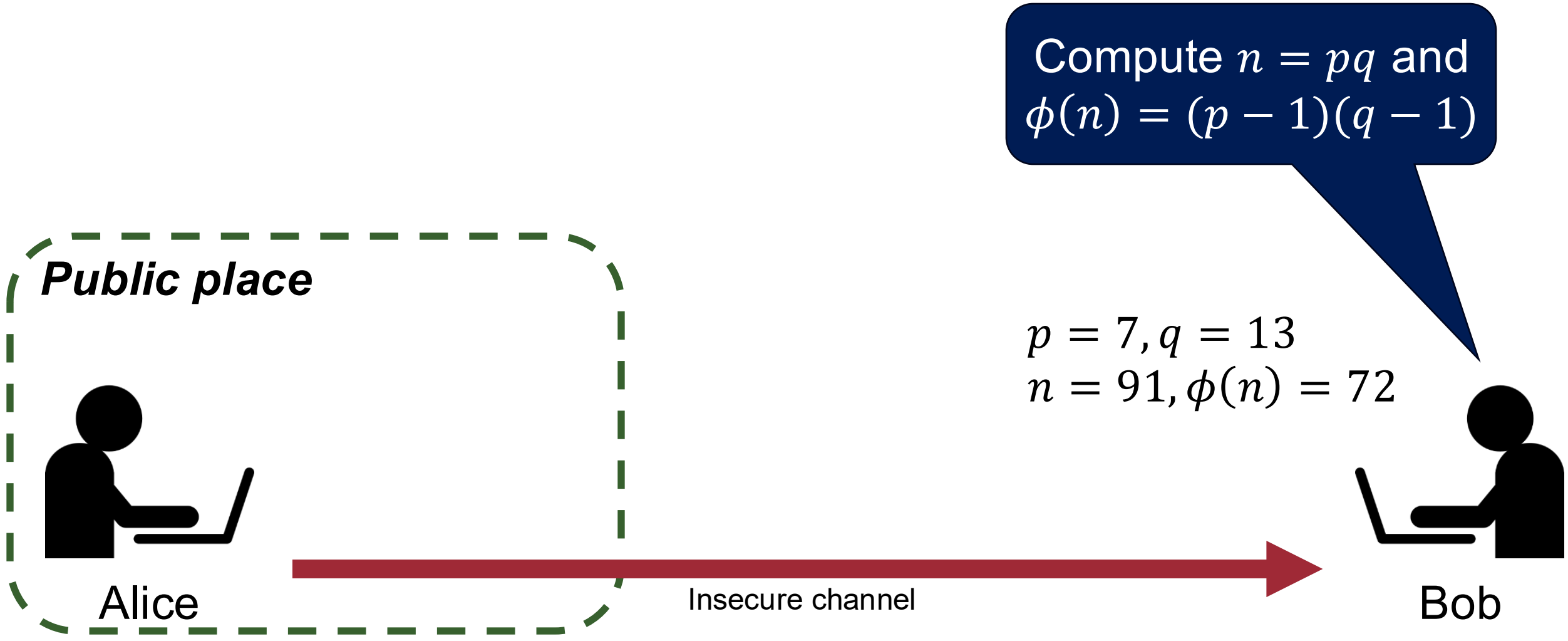
- pk: public key, widely disseminated, used for encryption
- sk: private key kept secretly, used for decryption



Recap: RSA Algorithm

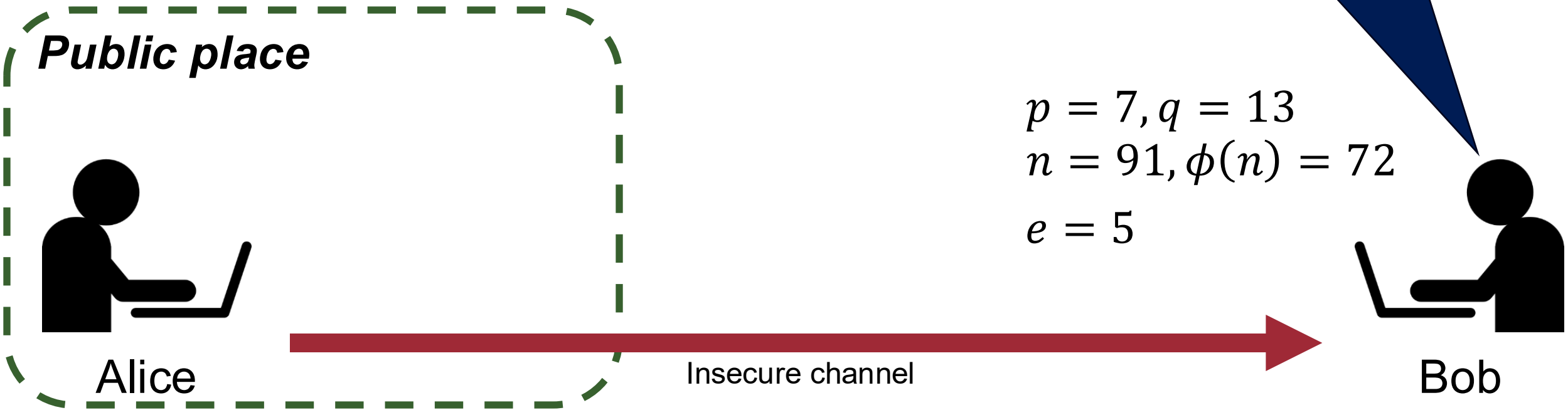


Recap: RSA Algorithm



Recap: RSA Algorithm

Choose e s.t.
- $1 < e < \phi(n)$ **and**
- $\gcd(\phi(n), e) = 1$



$p = 7, q = 13$

$n = 91, \phi(n) = 72$

$e = 5$

Recap: RSA Algorithm

How to find d?
→ Extended Euclidean Algorithm!

Choose d s.t.
- $1 < d < \phi(n)$ **and**
- $(ed \bmod \phi(n)) = 1$

Public place

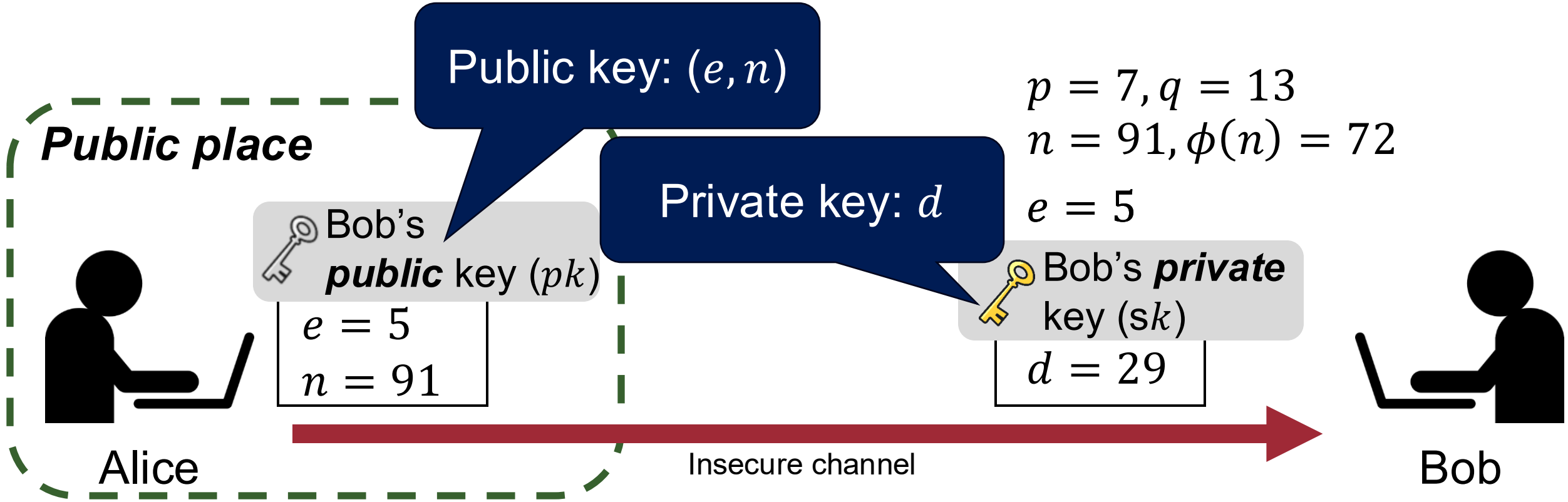
$p = 7, q = 13$
$n = 91, \phi(n) = 72$
$e = 5$
$d = 29$

Alice

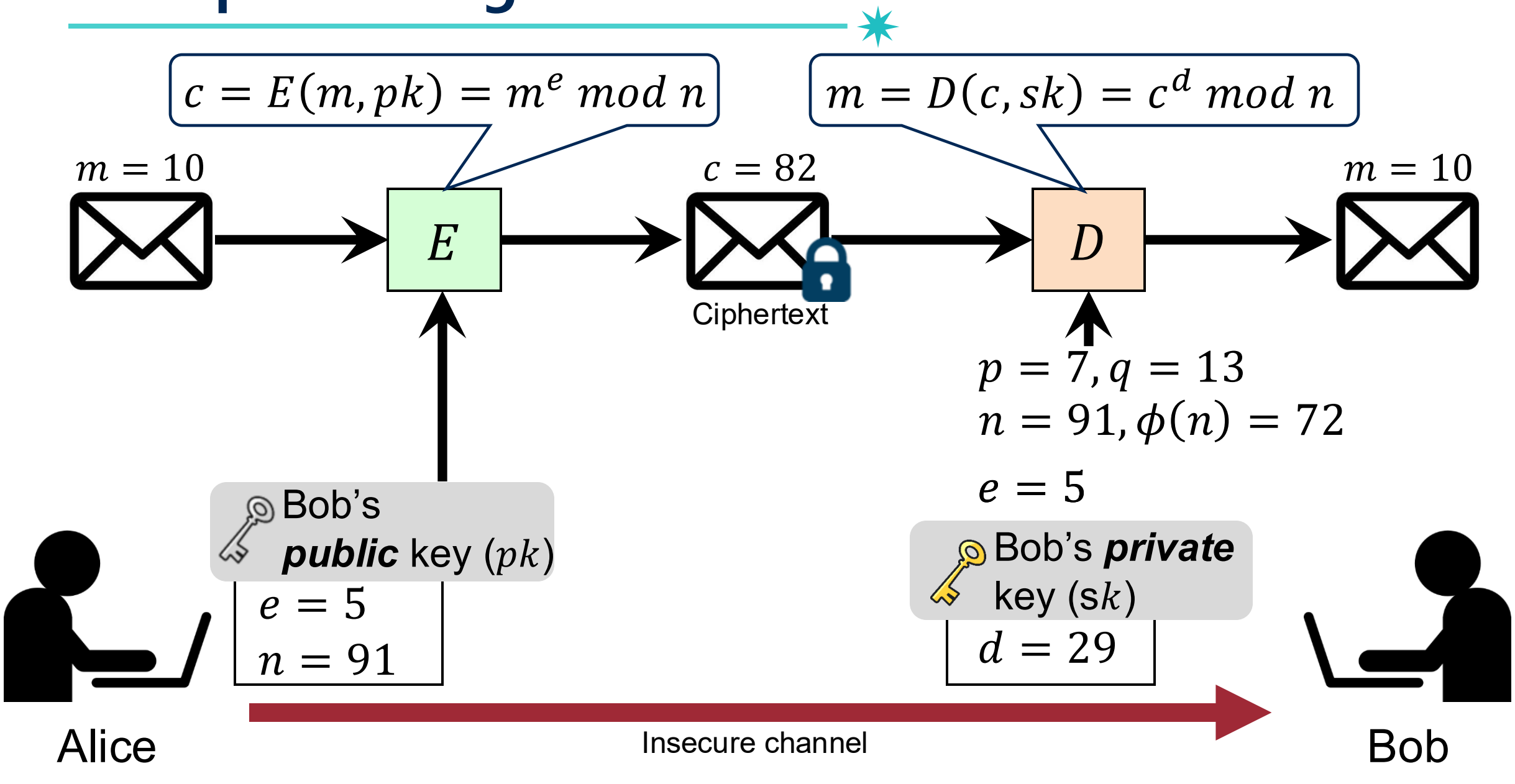
Insecure channel

Bob

Recap: RSA Algorithm



Recap: RSA Algorithm

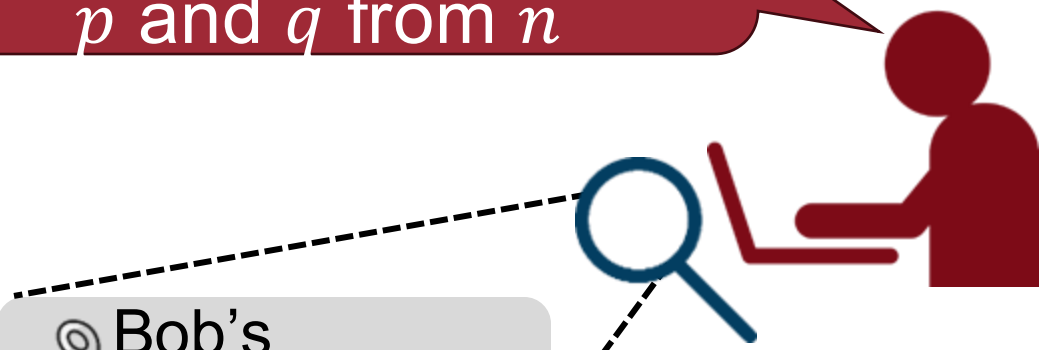


Recap: Security of the RSA Algorithm

$$c = E(m, pk) = m^e \bmod n$$

$$m = D(c, sk) = c^d \bmod n$$

The attacker cannot efficiently compute p and q from n



$p = 7, q = 13$

$n = 91, \phi(n) = 72$

$e = 5$

Bob's ***public*** key (pk)

$e = 5$

$n = 91$

$n = pq$

Bob's ***private*** key (sk)

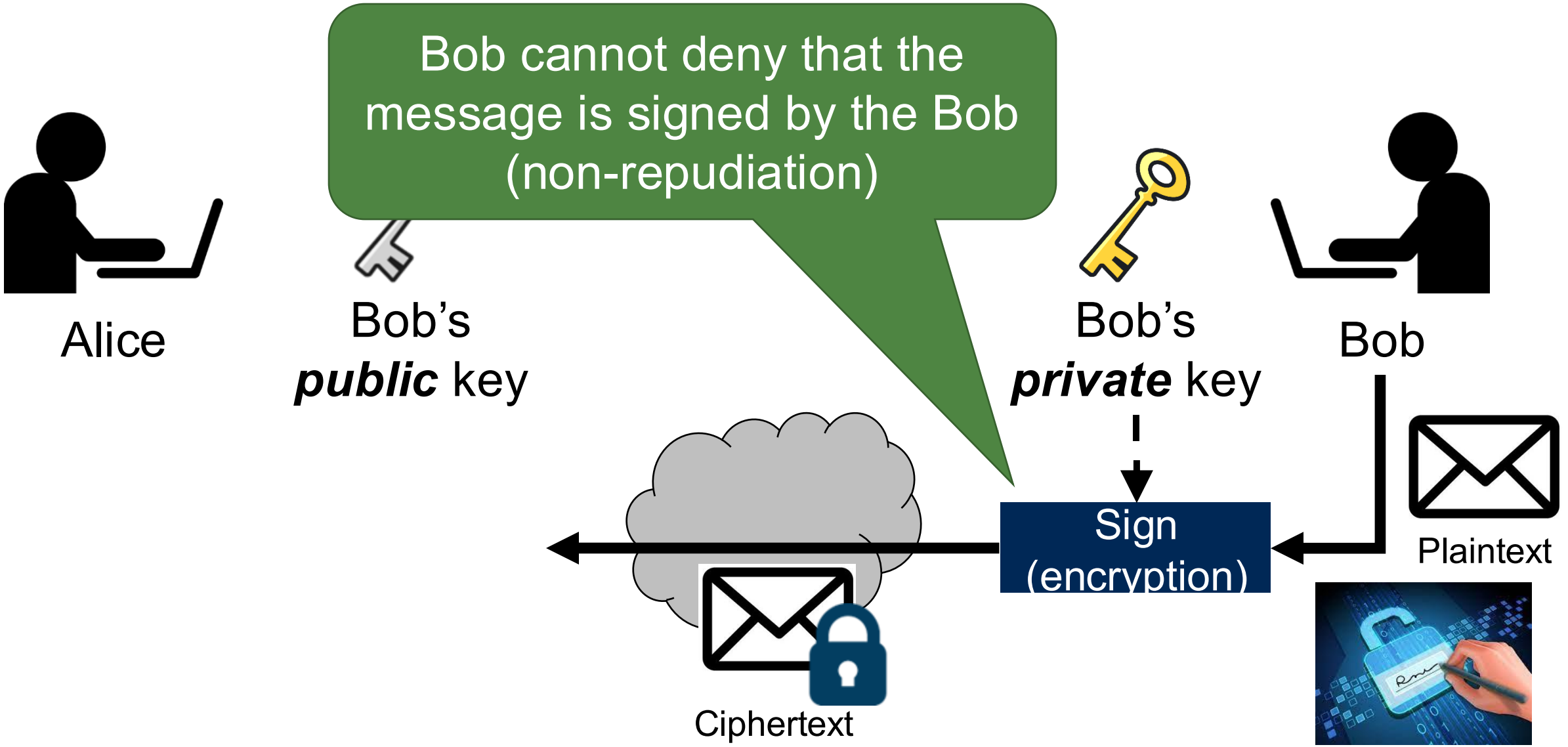
$d = 29$

Alice

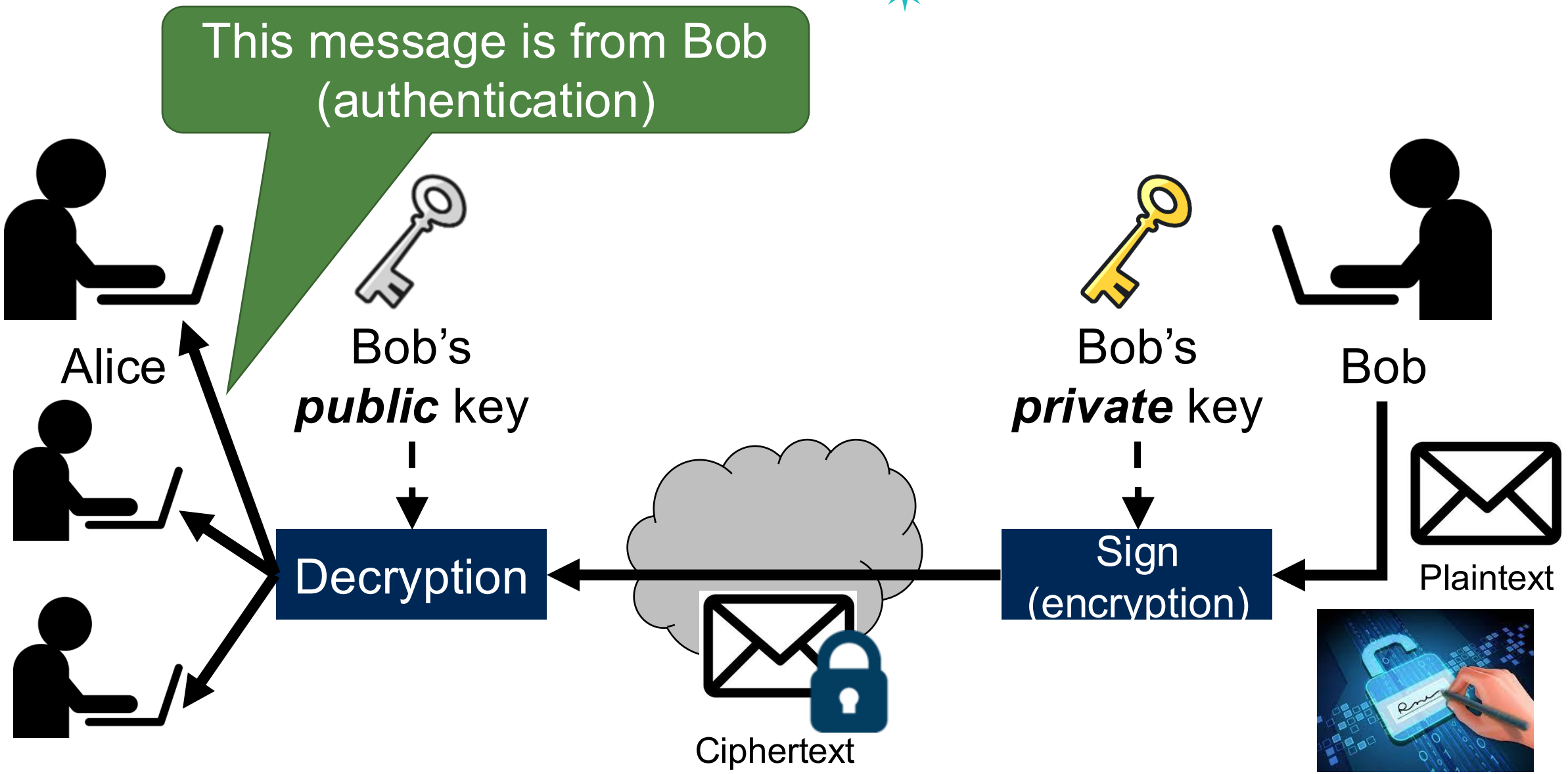
Insecure channel

Bob

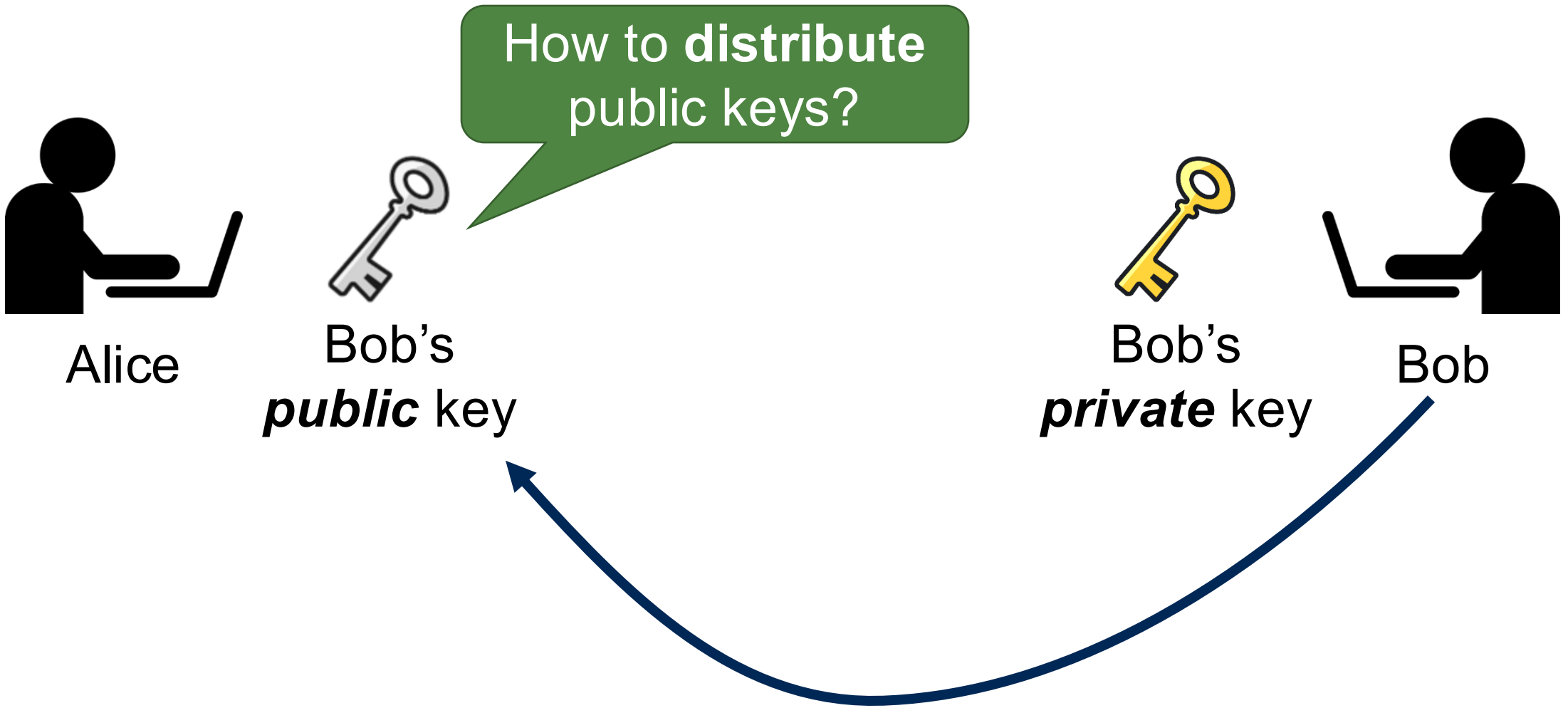
Recap: Digital Signature



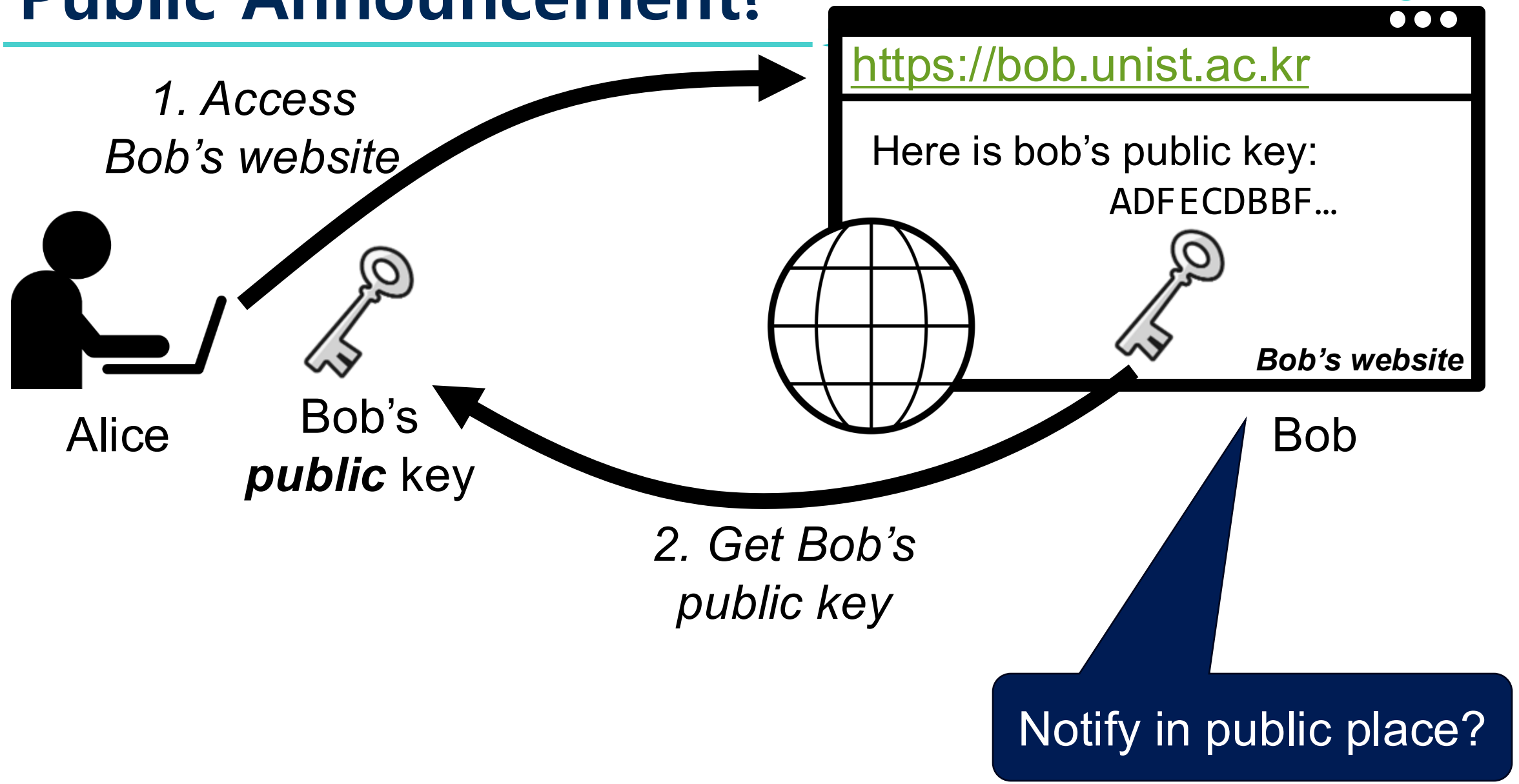
Recap: Digital Signature



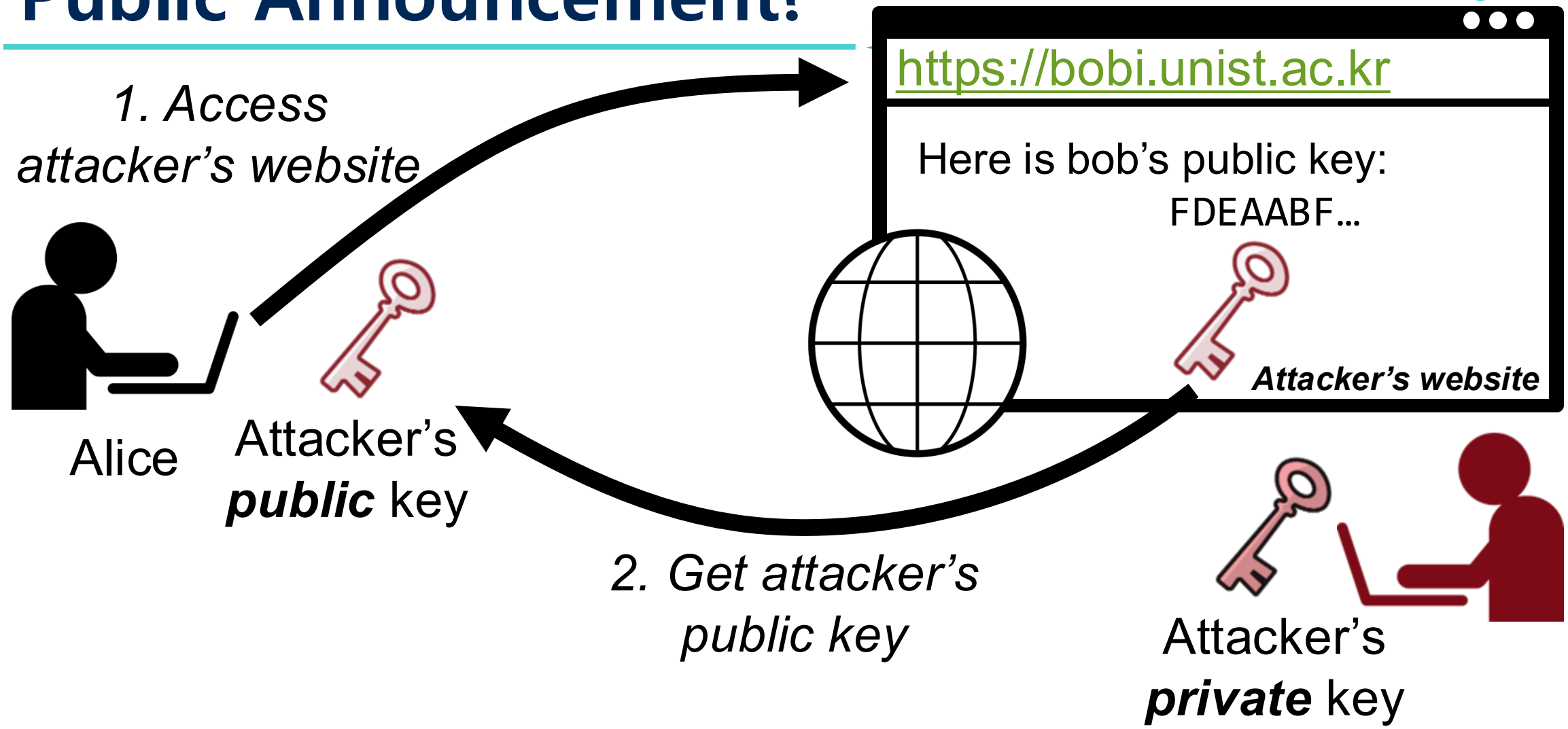
Today's Topic: Distribution of Public Keys



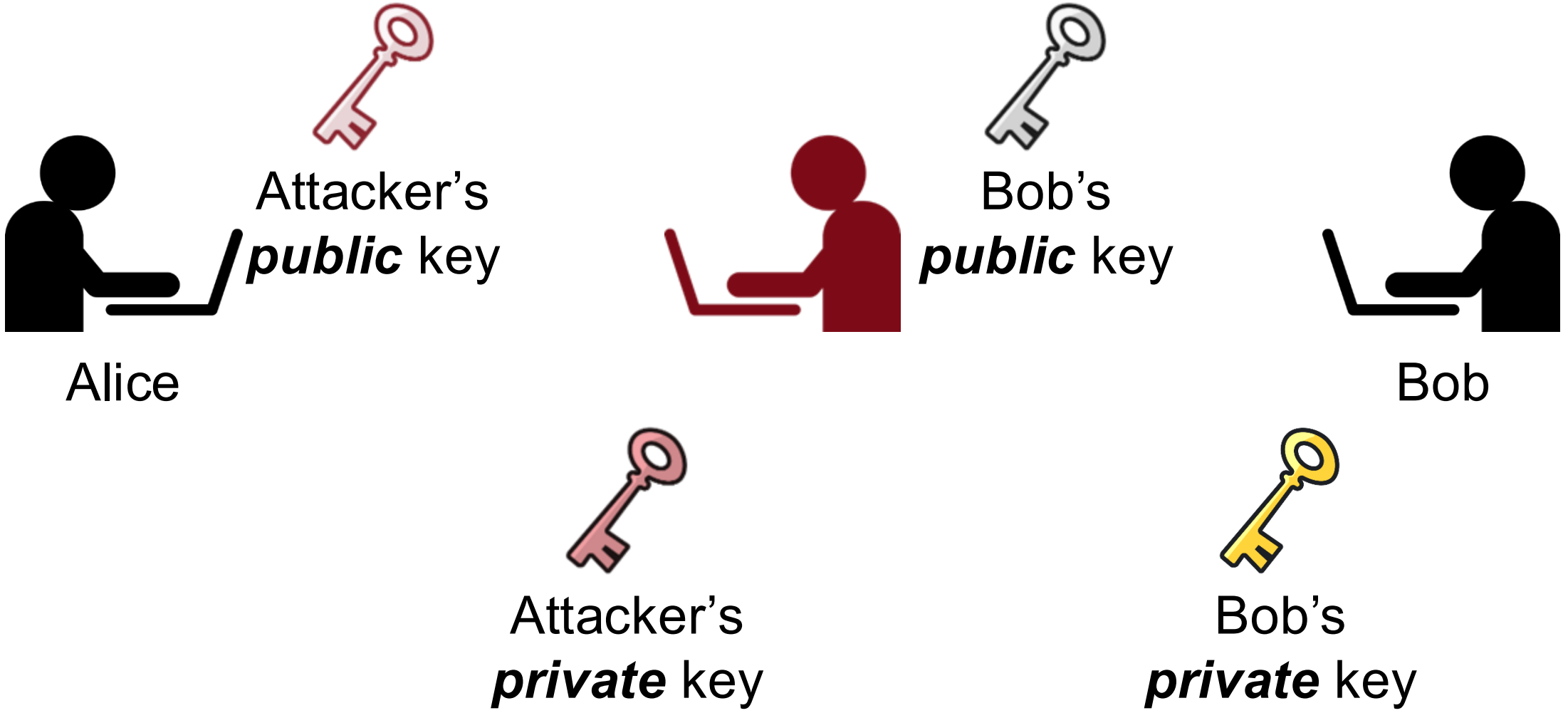
Public Announcement?



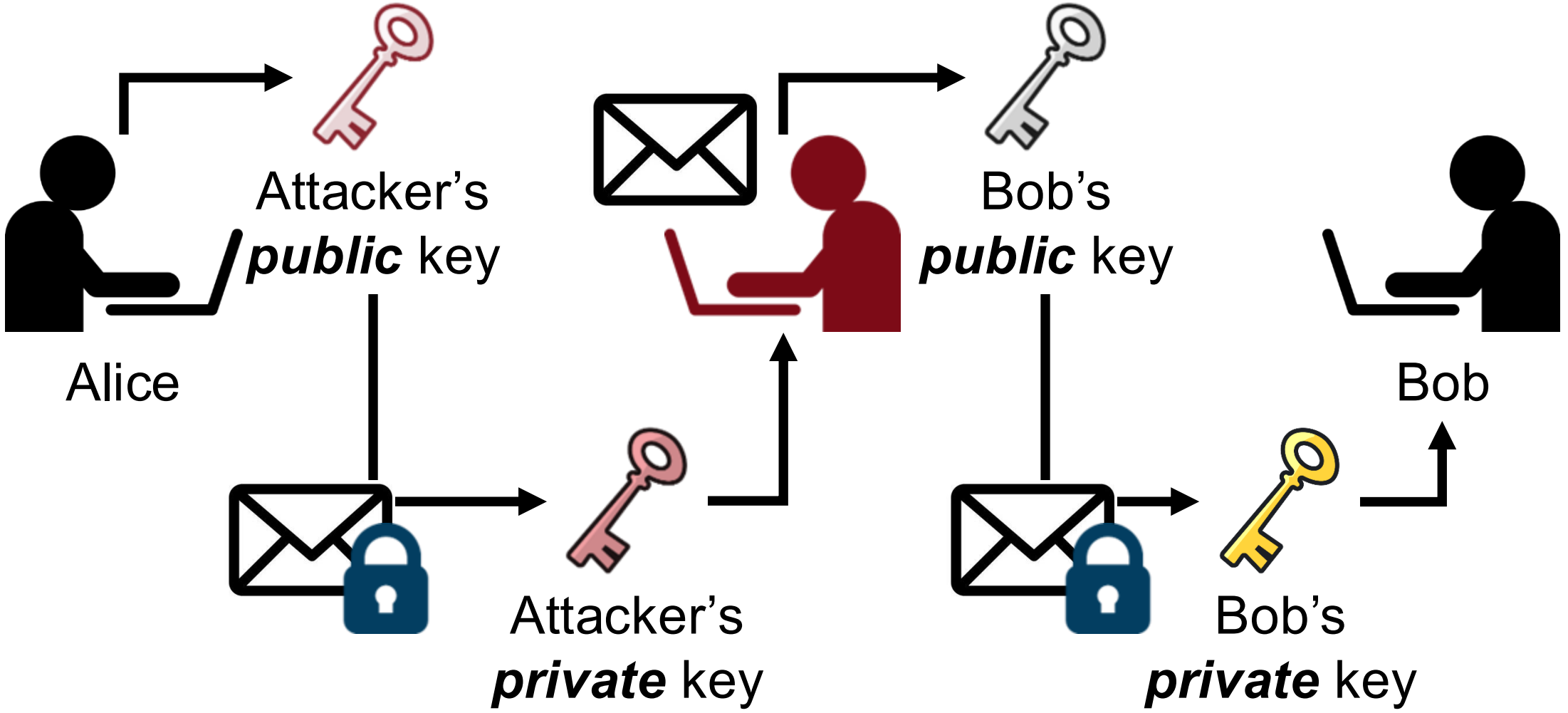
Public Announcement?



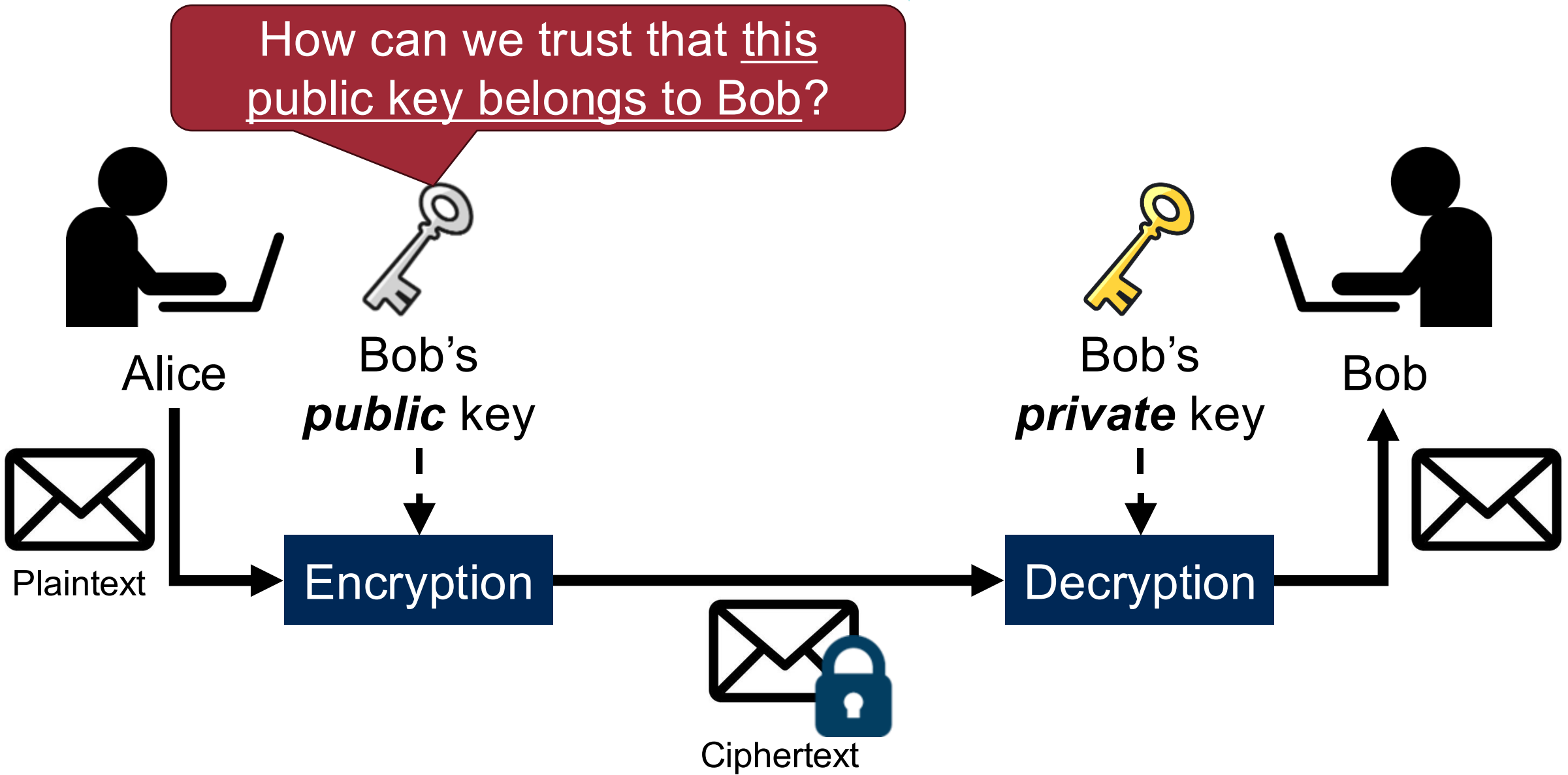
Man-in-the-Middle (MITM) Attack



Man-in-the-Middle (MITM) Attack

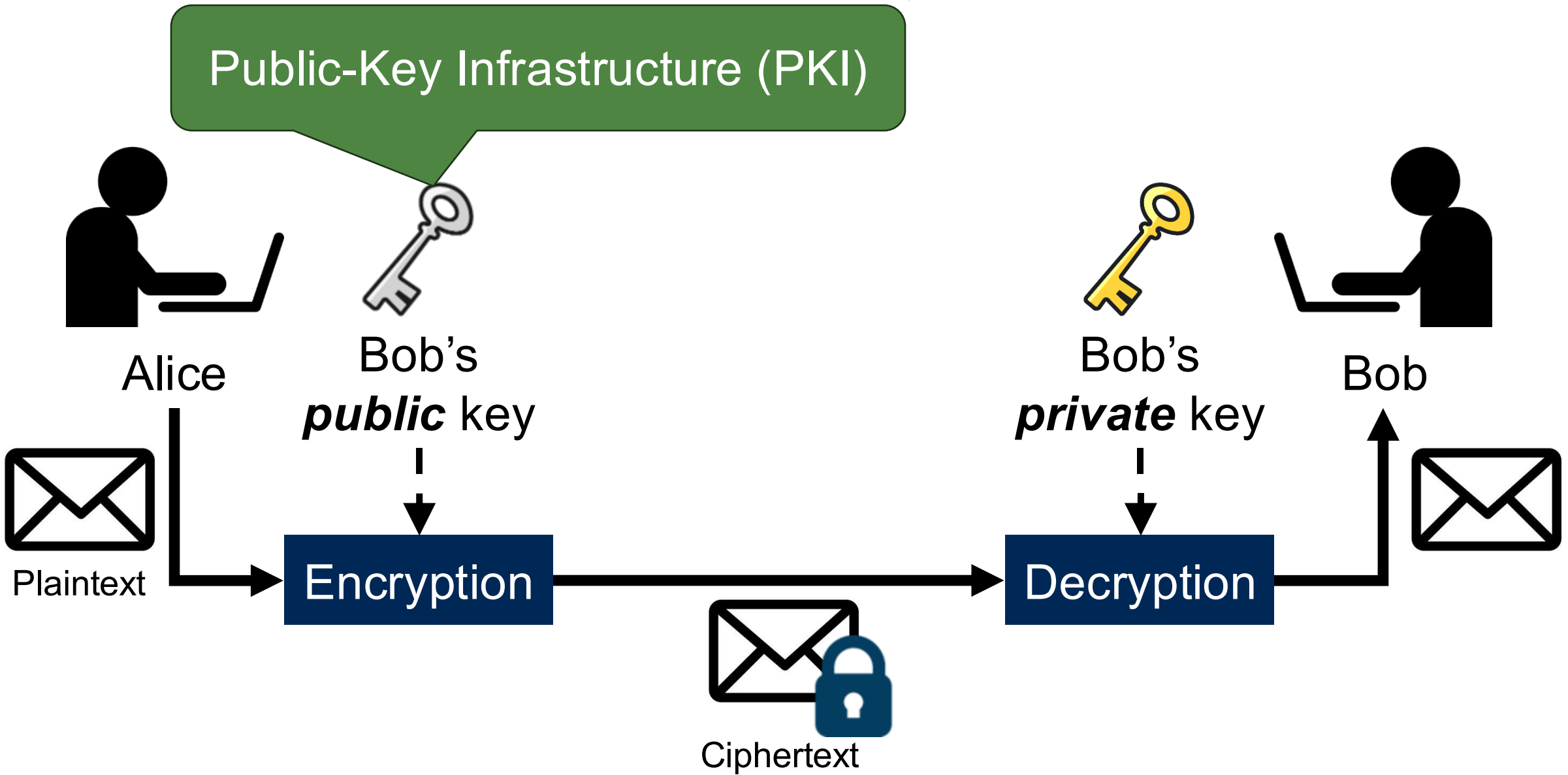


Motivation

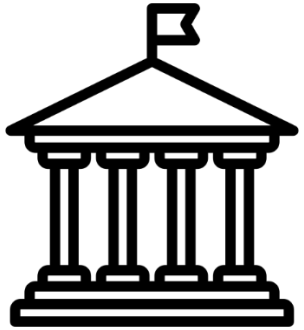

How can we trust that this public key belongs to Bob?
Alice
Bob's *public* key
Bob's *private* key
Bob
Plaintext
Encryption
Decryption
Ciphertext

Public-key Infrastructure (PKI)

Public-Key Infrastructure



Key Idea of Public-Key Infrastructure



Certificate Authority (CA)

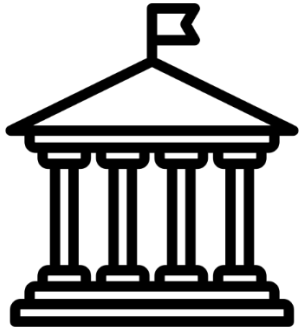
Alice

Bob

Alice

Bob

Key Idea of Public-Key Infrastructure



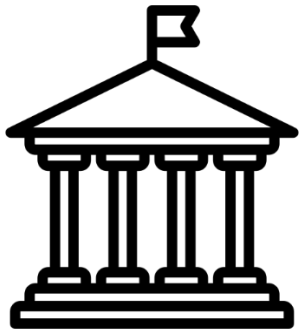
Certificate Authority (CA)

Trusted 3rd-party authority (KISA, yesSign, Verisign …)

Alice

Bob

Key Idea of Public-Key Infrastructure



Certificate Authority (CA)

Manage, distribute, verify **public-keys**

Alice

Bob

Key Idea of Public-Key Infrastructure

Digital Certificate

- ✓ **Subject**: Bob
- ✓ **Expires**: 11/25/2034
- ✓ **Bob's public key**: ADFECDBBF…

Certificate Authority (CA)

Bob's ***public*** key

Bind user's **identity** to **public key**

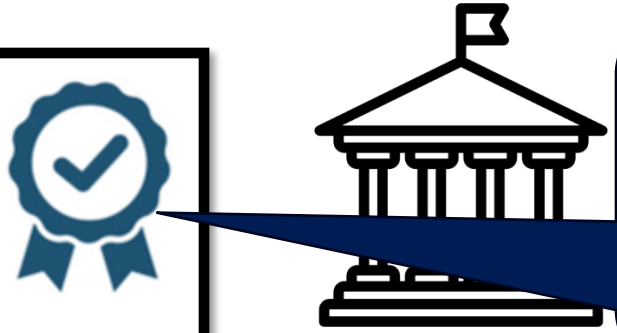
Alice

Bob

Key Idea of Public-Key Infrastructure

Digital Certificate

- ✓ **Subject**: Bob
- ✓ **Expires**: 11/25/2034
- ✓ **Bob's public key**: ADFECDBBF…



Certificate Authority (CA)

Digital signature of CA
(signed with CA's private key)

→ Hash-based digital signature

public key

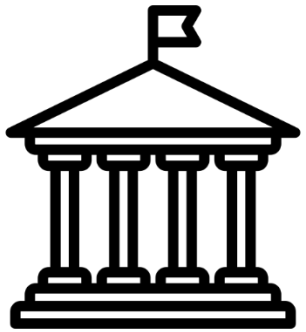
Bind user's **identity** to **public key**

Alice

Bob

Hash-based Digital Signature in PKI

Signing



Certificate Authority (CA)

Digital Certificate

- ✓ **Subject**: Bob
- ✓ **Expires**: 11/25/2034
- ✓ **Bob's public key**: ADFECDBBF…

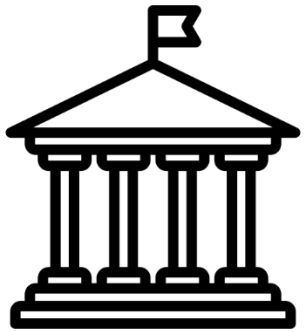
Hash function

0101000010..

Encrypt with CA's ***private key***

Hash-based Digital Signature in PKI

Signing



Certificate Authority (CA)

Digital Certificate

- ✓ **Subject**: Bob
- ✓ **Expires**: 11/25/2024
- ✓ **Bob's public key**: ADFECDBBF…

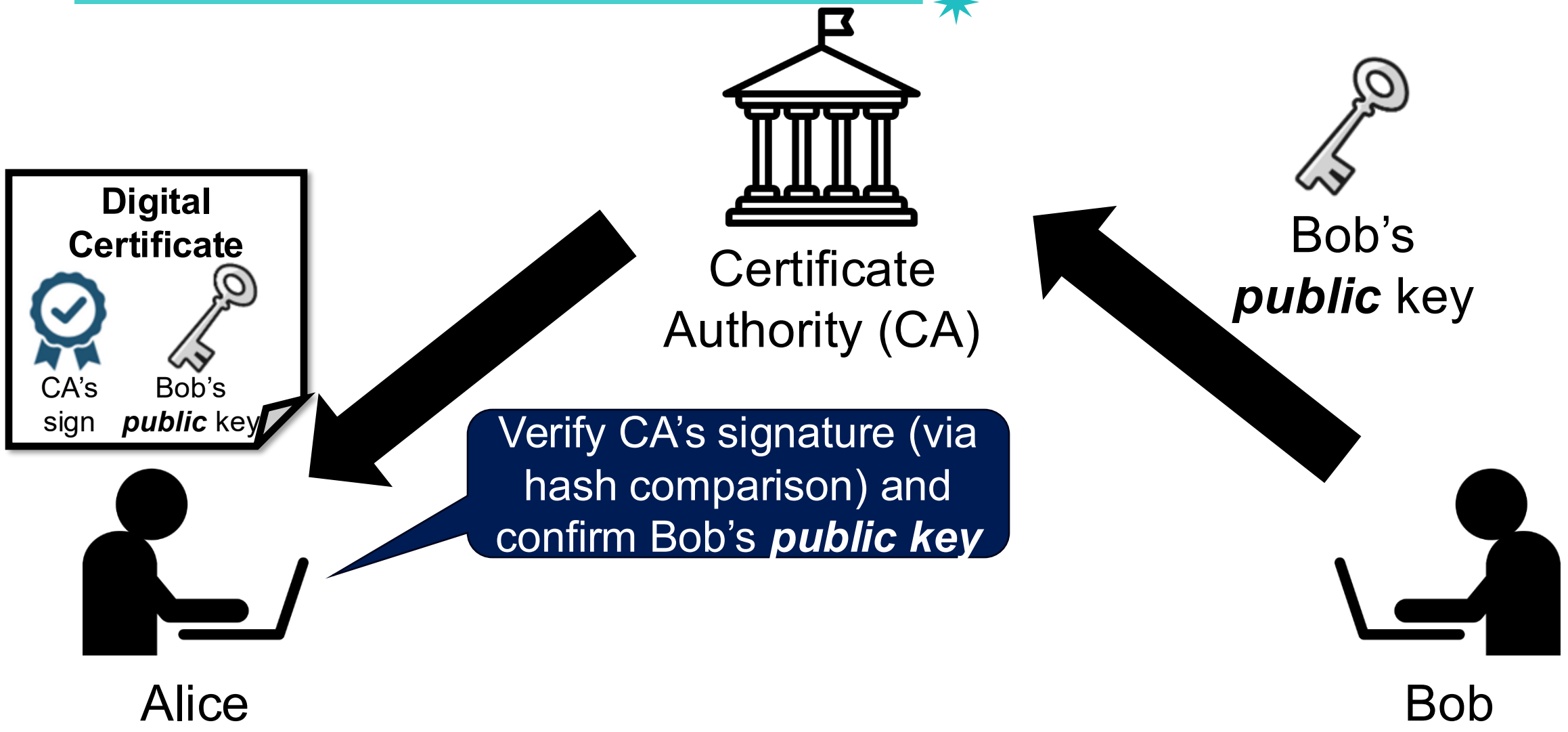
Hash function

0101000010..

Encrypt with CA's ***private key***

Append

Key Idea of Public-Key Infrastructure



Hash-based Digital Signature in PKI

Verification

Alice

Digital Certificate

- ✓ **Subject**: Bob
- ✓ **Expires**: 11/25/2034
- ✓ **Bob's public key**: ADFECDBBF…

Hash-based Digital Signature in PKI

Verification

Alice

Digital Certificate

- ✓ **Subject**: Bob
- ✓ **Expires**: 11/25/2034
- ✓ **Bob's public key**: ADFECDBBF…

Hash function

Decrypt with CA's ***public key***

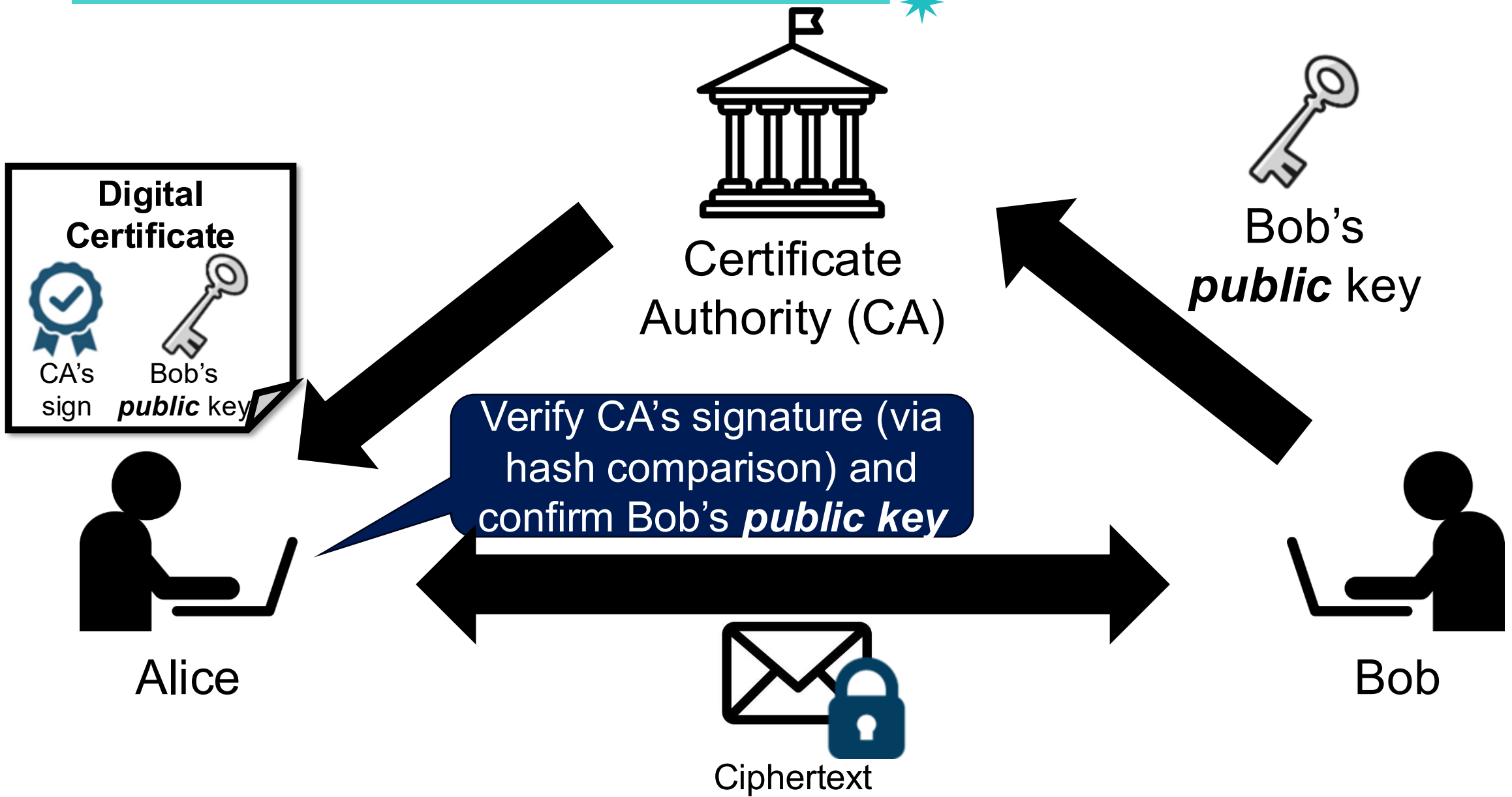
0101000010..

?

0101000010..

1. Confirm Bob's public key
2. Integrity check

Key Idea of Public-Key Infrastructure



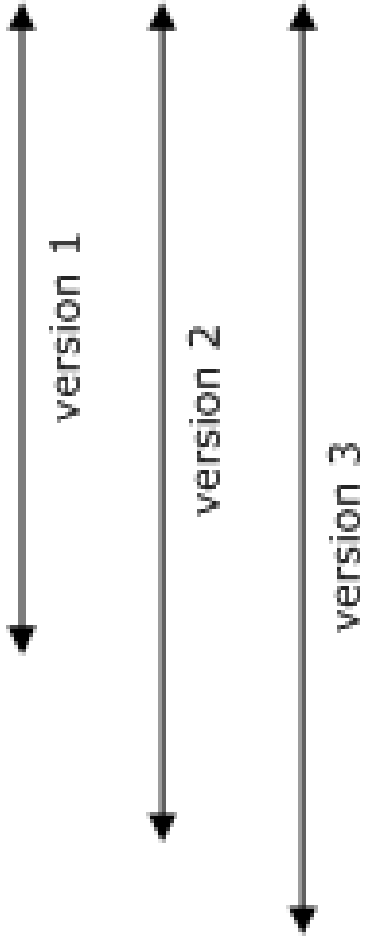
Public-Key Infrastructure (PKI)

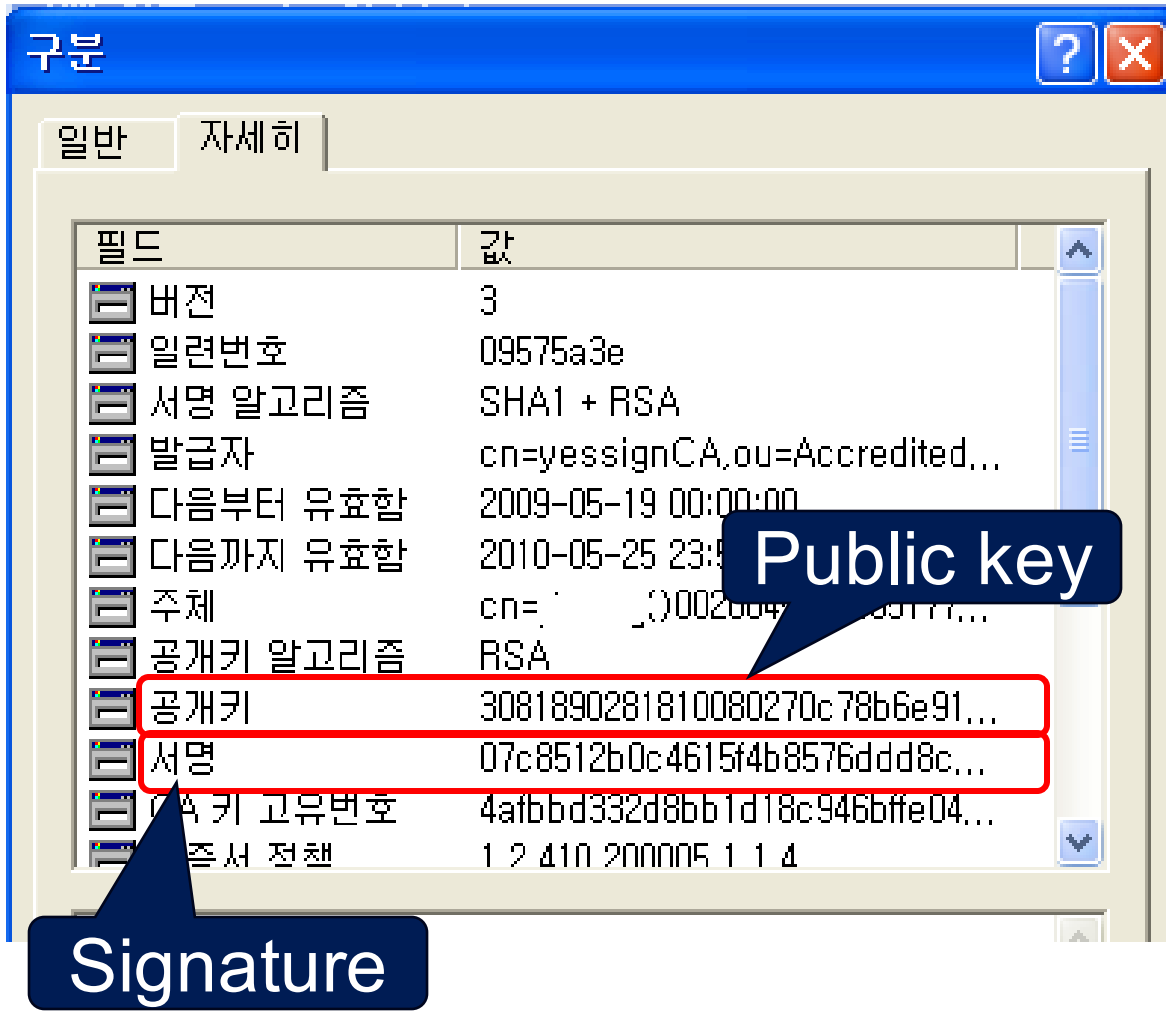
- The set of processes required to create, manage, distribute, use, store, and revoke **digital certificates** and **public-keys**

- Two important components
  - **Certificate Authority (CA):** a **trusted party**, responsible for verifying the identity of users, and then bind the verified identity to a public keys
  - **Digital Certificates**: a document certifying that the public key included inside does belong to the identity described in the document
    - X.509 standard

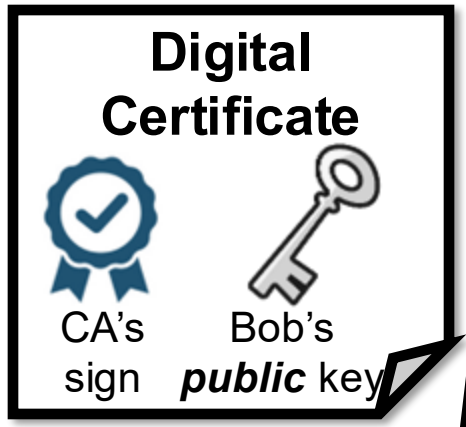
X.509 Certificate

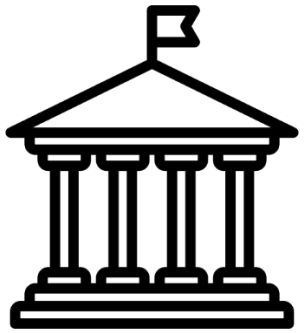
Version
Serial Number
Signature Algorithm Identifier
Issuer Name
Validity Period
Subject Name
Public Key Information
Issuer Unique ID
Subject Unique ID
Extensions





Chain of Trust





Certificate Authority (CA)

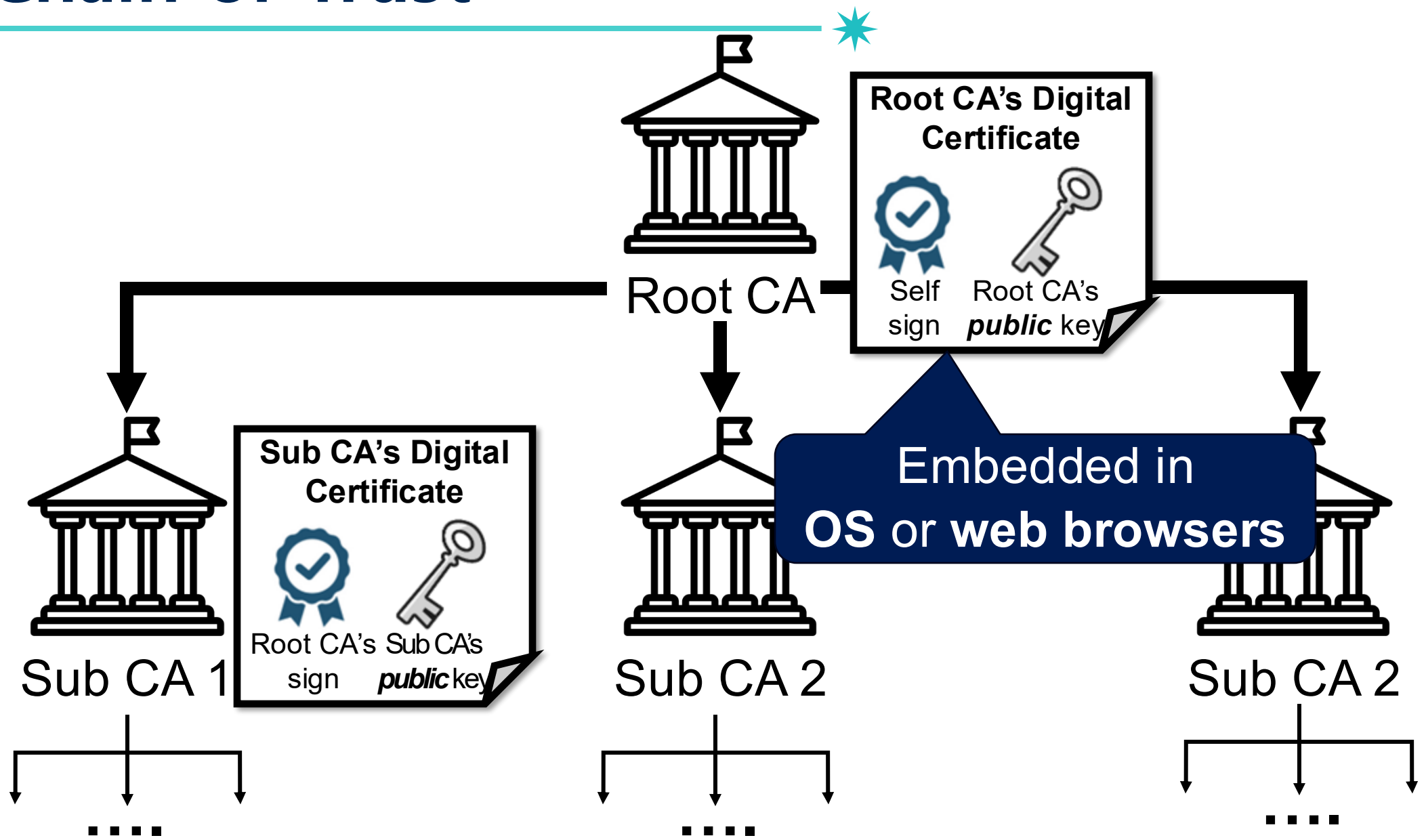
Verify CA's signature (via hash comparison) and confirm Bob's ***public key***

Alice

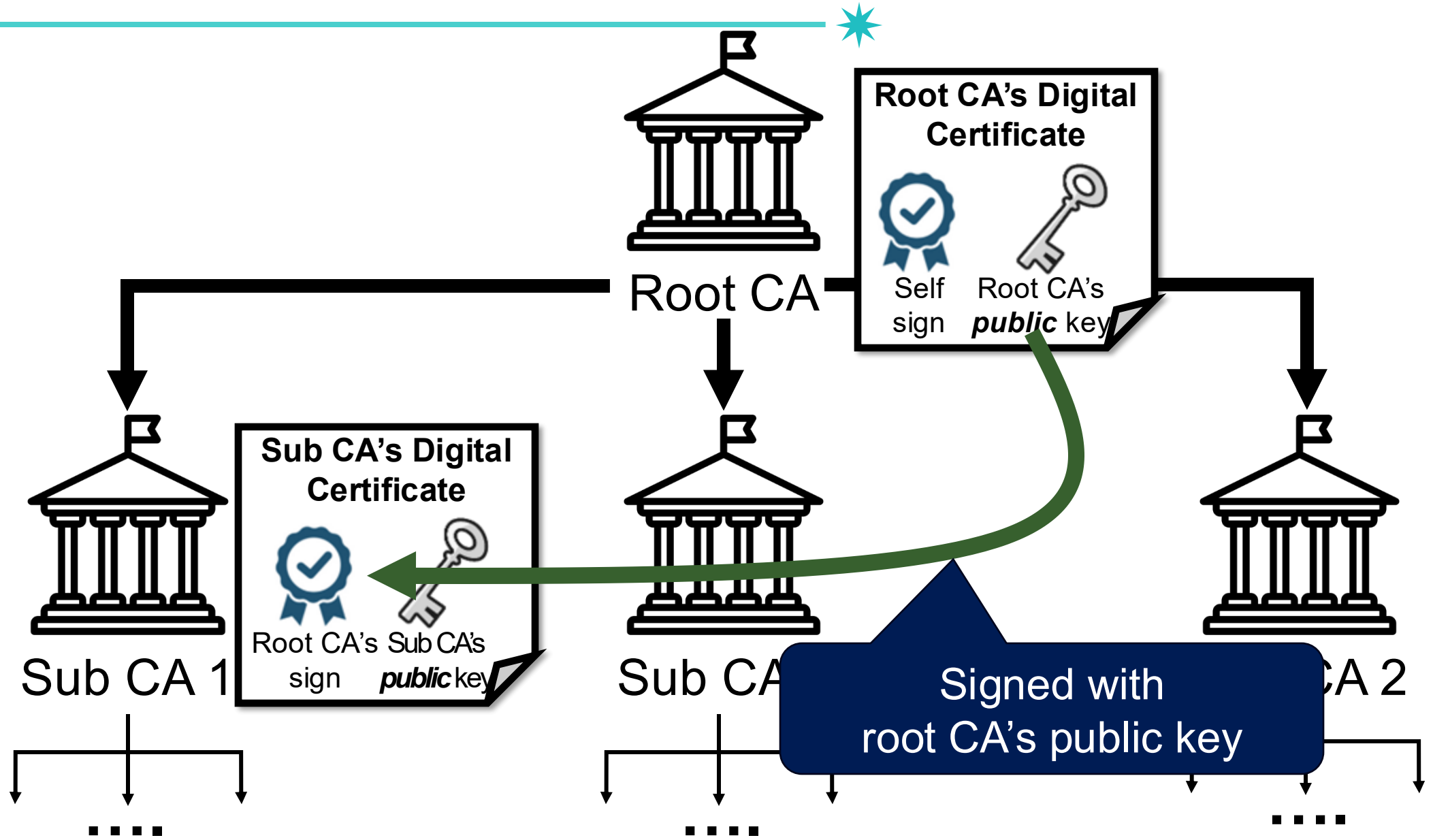
Recursive concern:

How can we trust that the public key belongs to CA?

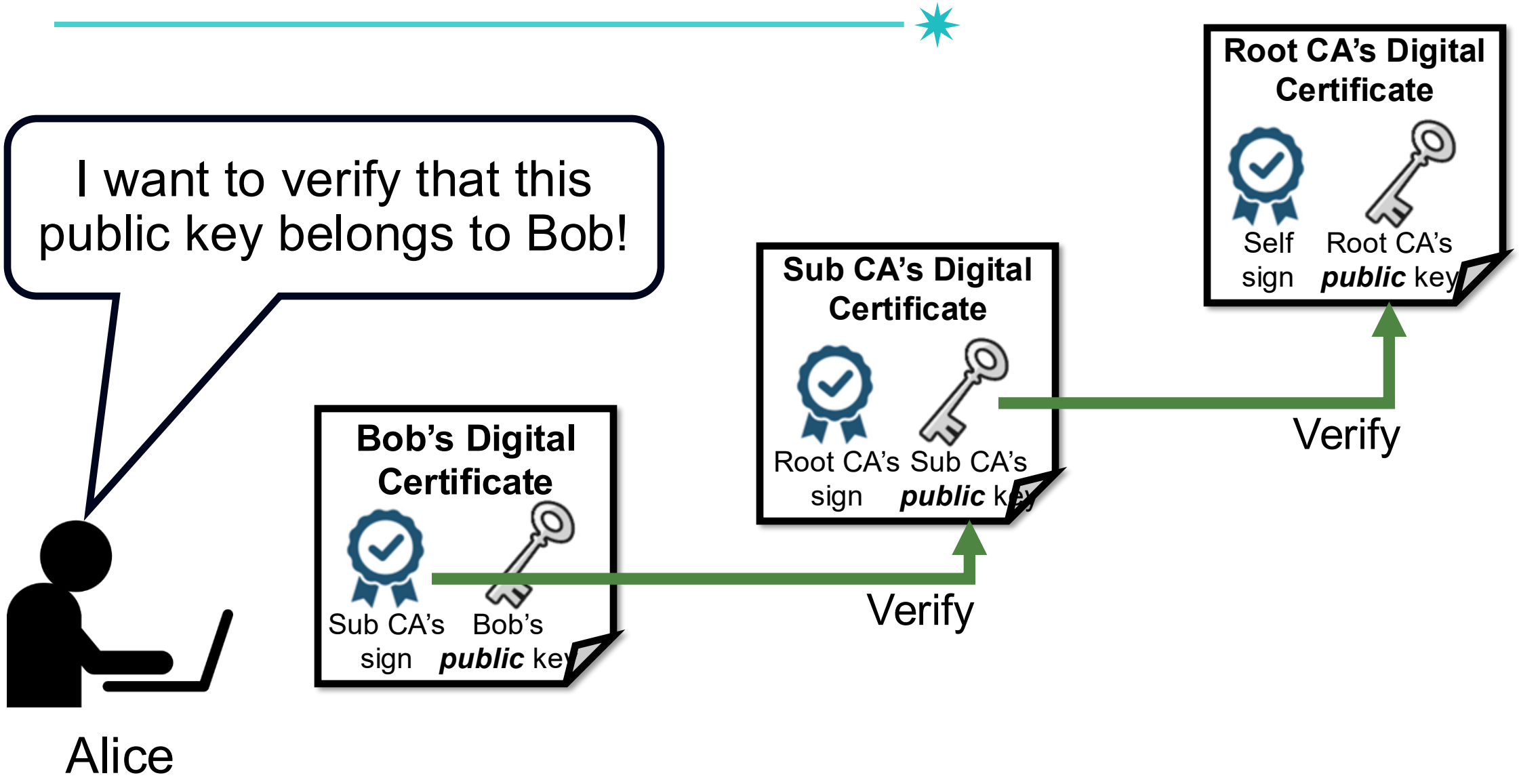
Chain of Trust



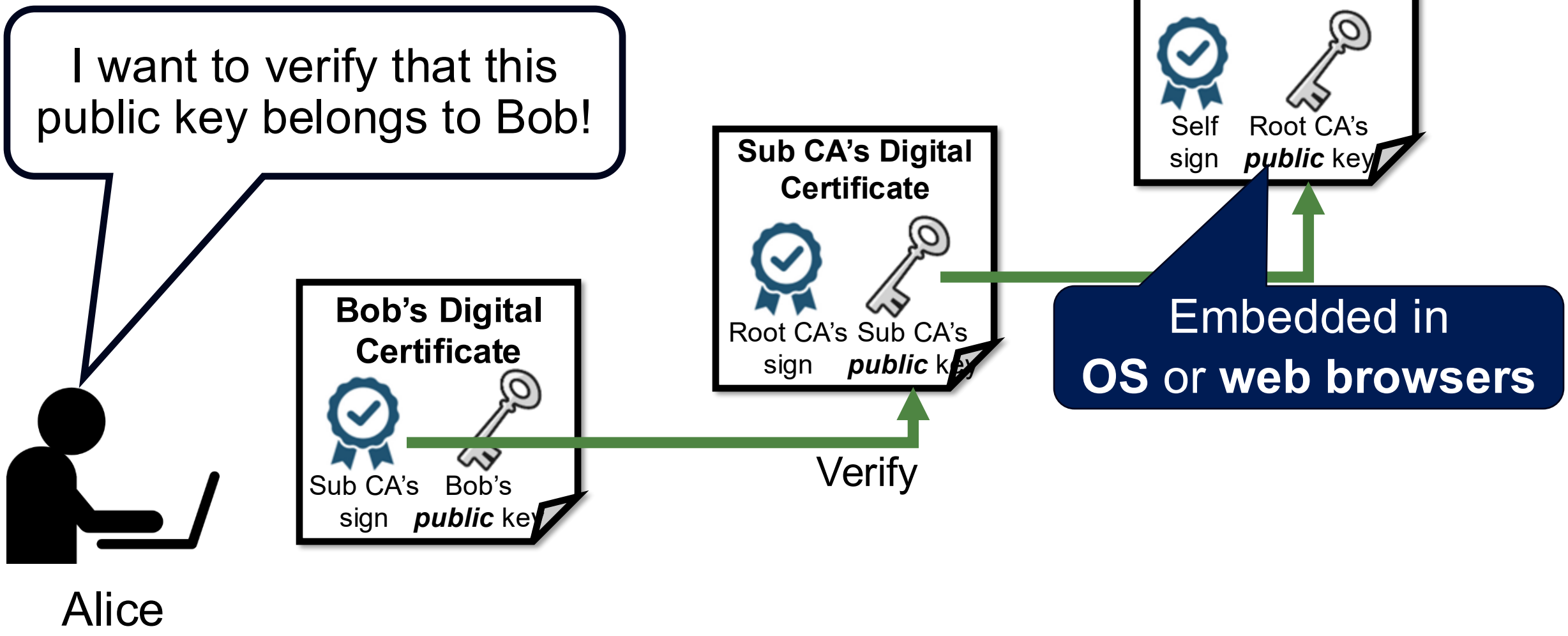
Chain of Trust



Chain of Trust

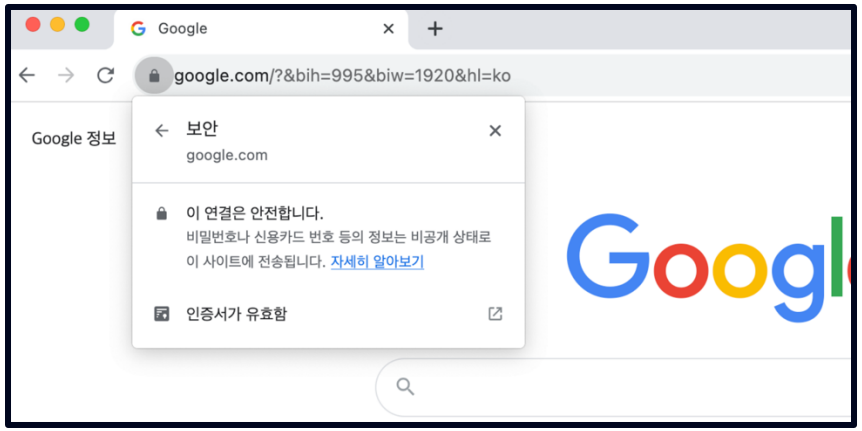


Chain of Trust

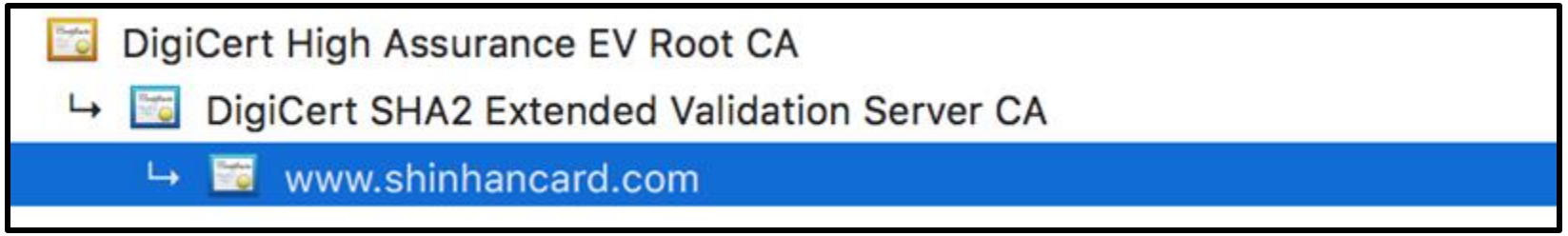


Certificate Authority and Root CA

- Users need some “Root” keys to start with
  - Root CA’s Certificate
  - Embedded in OS or web browsers
    - (Example #1) Root CAs for iOS: https://support.apple.com/en-us/HT208125
    - (Example #2) Chrome



- An example chain of CAs assuring the shinhancard.com:



The Core Functionalities of CA

1. **Verify the subject**
   - Ensure that the person applying for the certificate either owns or represents the identity in the subject field

2. **Signing digital certificates**
   - CA generates a digital signature for the certificate **using its private key**
   - Once the signature is applied, the certificate cannot be modified
   - Signatures can be verified by anyone with the CA's public key

Digital Certificate

- Let's get paypal's certificates

```
$ openssl s_client -showcerts -connect www.paypal.com:443 </dev/null
```

```
-----BEGIN CERTIFICATE-----
MIIHWTCCBkGgAwIBAgIQLNGVEFQ30N5KOSAFavbCfzANBgkqhkiG9w0BAQsFADB3
MQswCQYDVQQGEwJVUzEdMBsGA1UEChMUU3ltYW50ZWMgQ29ycG9yYXRpb24xHzAd
... (omitted) ...
GN/QMQ3a55rjwNQnA3s2WWuHGPaE/jMG17iiL2O/hUdIvLE9+wA+fWrey5//74xl
NeQitYiySDIepHGnng==
-----END CERTIFICATE-----
```

- Save the above data to paypal.pem, and use the following command decode it (see next slide)

```
$ openssl x509 -in paypal.pem -text -noout
```

Example of X.509 Certificate (1st Part)

The CA's identity (Symantec)

The owner of the certificate (paypal)

```
Certificate:
Data:
  Serial Number:
          2c:d1:95:10:54:37:d0:de:4a:39:20:05:6a:f6:c2:7f
  Signature Algorithm: sha256WithRSAEncryption
  Issuer: C=US, O=Symantec Corporation, OU=Symantec Trust Network,
         CN=Symantec Class 3 EV SSL CA - G3
  Validity
     Not Before: Feb  2 00:00:00 2016 GMT
     Not After : Oct 30 23:59:59 2017 GMT
  Subject: 1.3.6.1.4.1.311.60.2.1.3=US/
          1.3.6.1.4.1.311.60.2.1.2=Delaware/
          businessCategory=Private Organization/
          serialNumber=3014267, C=US/
          postalCode=95131-2021, ST=California,
          L=San Jose/street=2211 N 1st St,
          O=PayPal, Inc., OU=CDN Support, CN=www.paypal.com
```

Example of X.509 Certificate (2nd Part)

Public key

CA's signature

```
  Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
         Public-Key: (2048 bit)
         Modulus:
           00:da:43:c8:b3:a6:33:5d:83:c0:63:14:47:fd:6b:22:bd:
           bf:4e:a7:43:11:55:eb:20:8b:e4:61:13:ee:de:fe:c6:e2:
           ... (omitted) ...
           7a:15:00:c5:01:69:b5:10:16:a5:85:f8:fd:07:84:9a:c9:
         Exponent: 65537 (0x10001)
Signature Algorithm: sha256WithRSAEncryption
4b:a9:64:20:cc:77:0b:30:ab:69:50:d3:7f:de:dc:7c:e2:fb:93:84:fd:
78:a7:06:e8:14:03:99:c0:e4:4a:ef:c3:5d:15:2a:81:a1:b9:ff:dc:3a:
... (omitted) ...
fb:00:3e:7d:6a:de:cb:9f:ff:ef:8c:65:35:e4:22:b5:88:b2:48:32:1e:
```

Integrity

Encryption vs Integrity

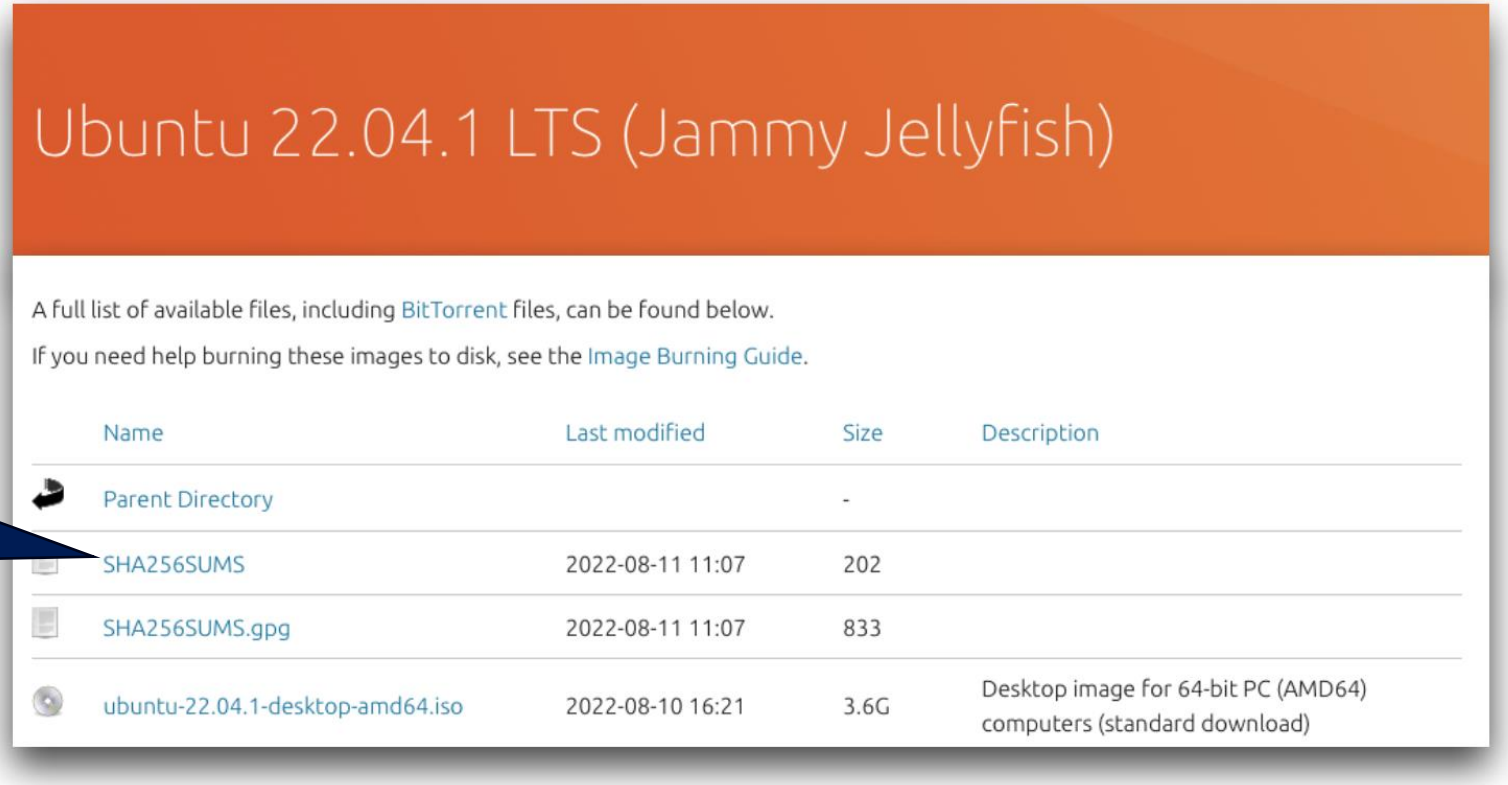
- “Encryption hides message contents and thus adversary cannot modify the encrypted message” [T / F]?

In many cases, message integrity is equally (or more) important

Recap: Integrity

- Information has not been altered in an unauthorized way
- How to ensure the integrity of computer systems?

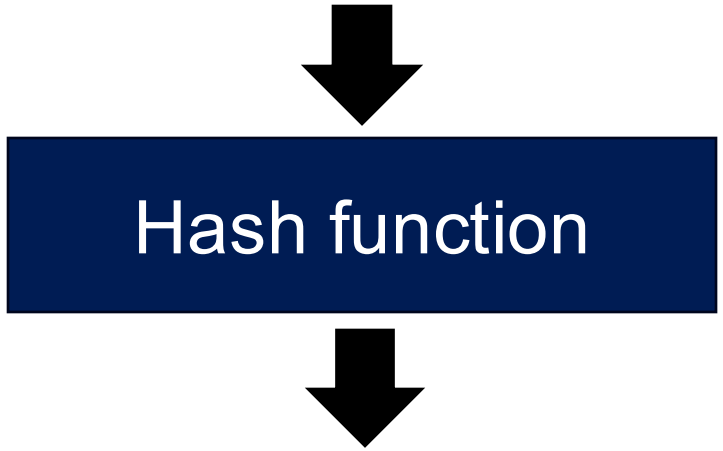
Cryptographic hash function (e.g., SHA256)



Cryptographic Hash Functions

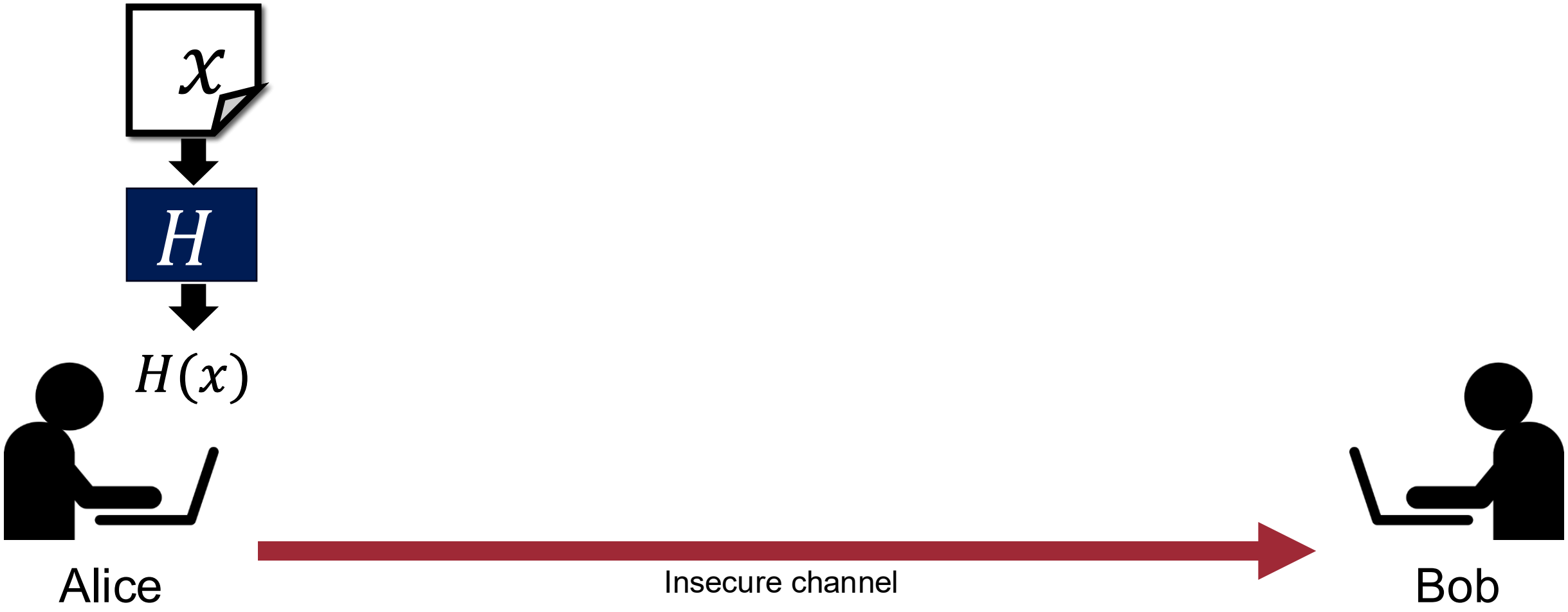
- Condense arbitrary message to fixed size (512 bit…)
- (important!) No **key** for input
- Usually assume hash function is public (e.g., MD5, SHA-512, etc.)

```
01010100100001010100000010000101010110101010010000101010000
00010000101010110101010010011111100100100010000101010110111
11101000101010000111111010010101001010010101011111111...
```

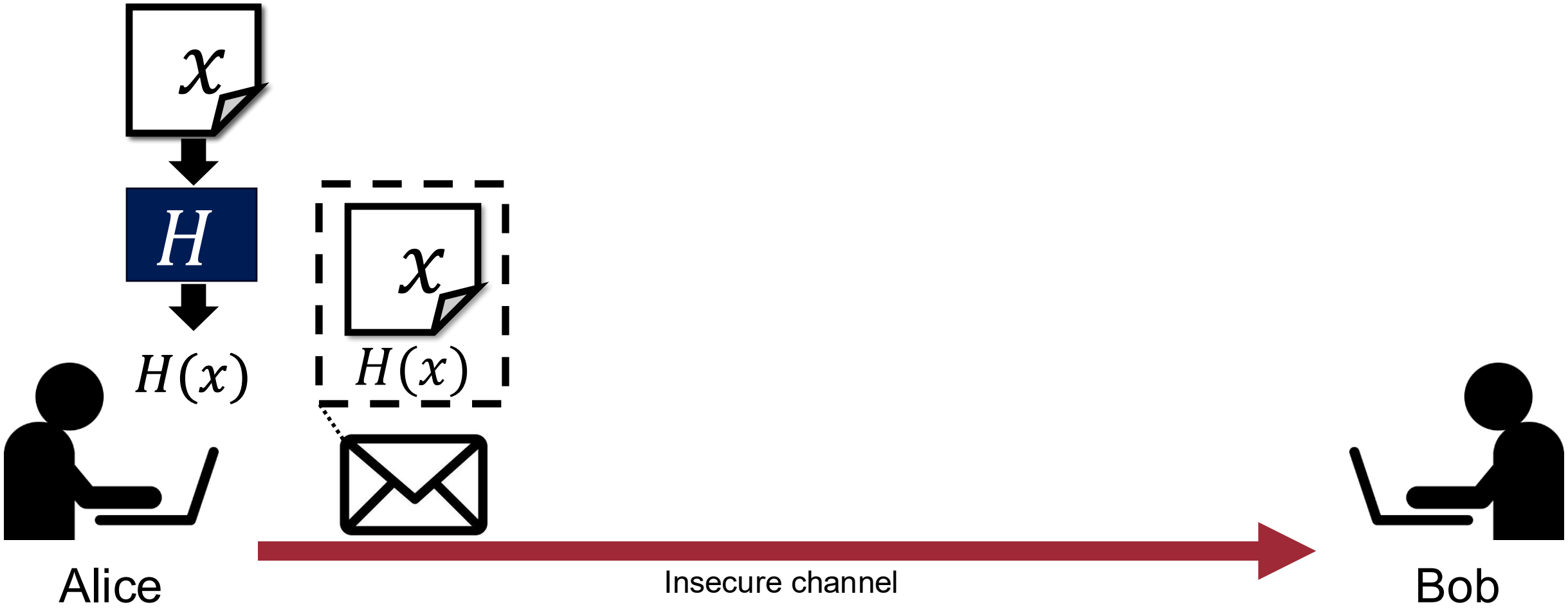



```
01011010101001011111111
```

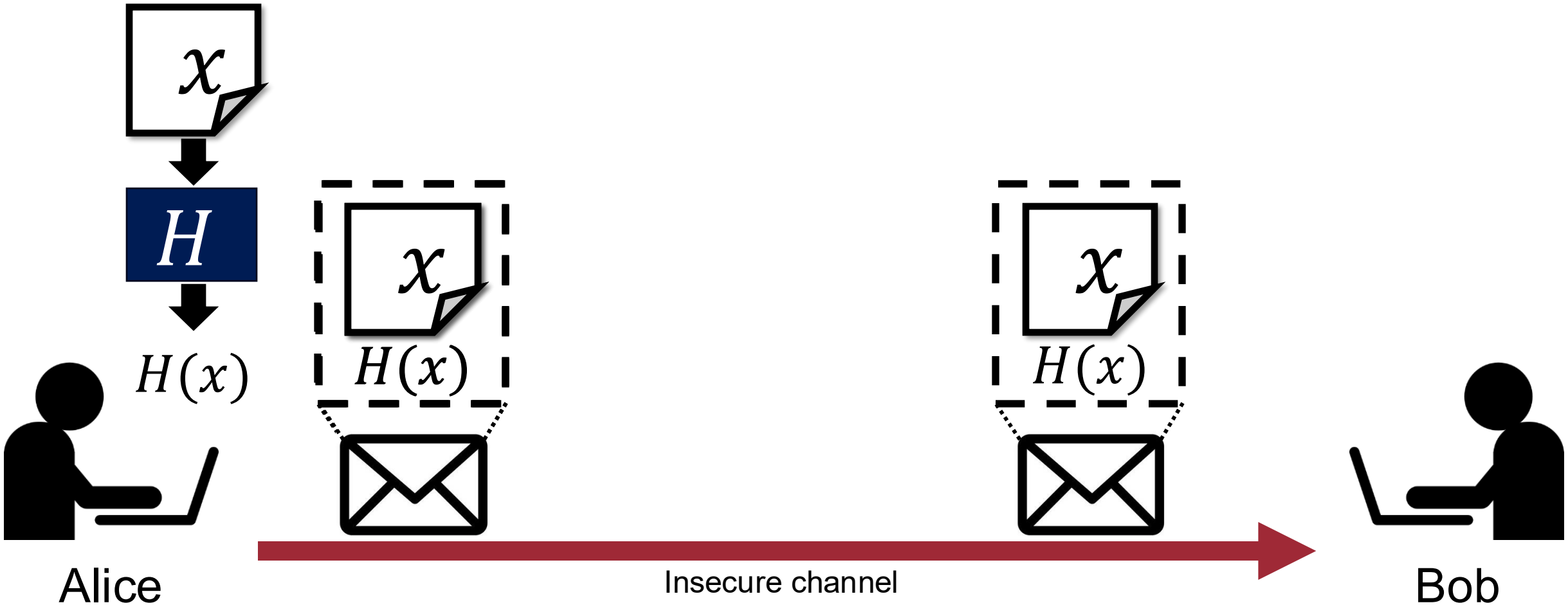
Hash Usage Example



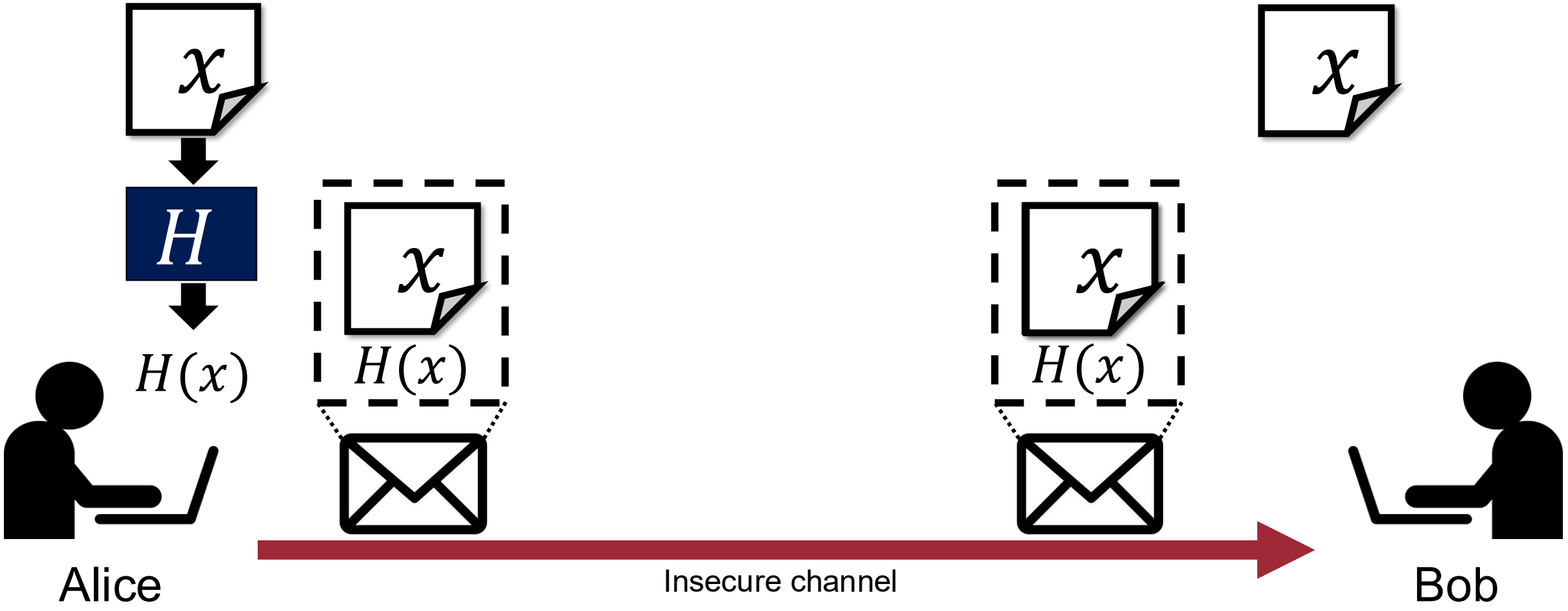
Hash Usage Example



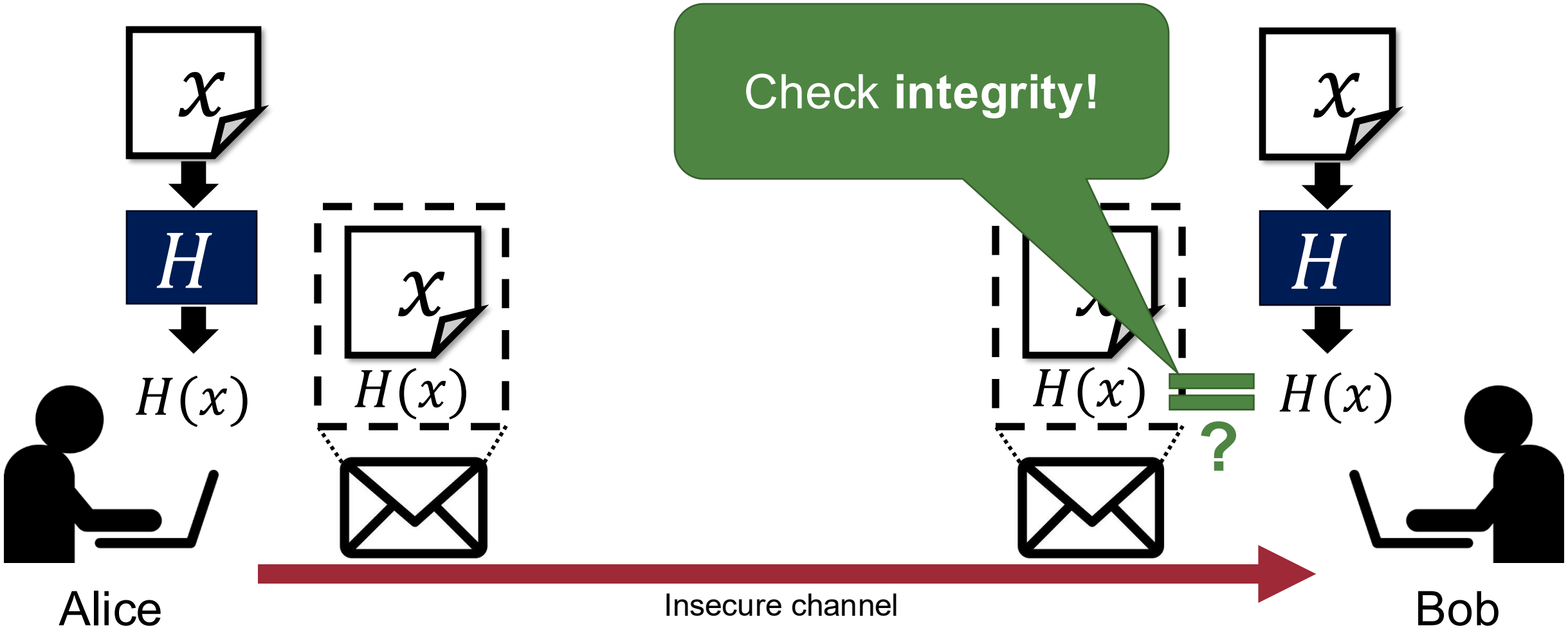
Hash Usage Example



Hash Usage Example



Hash Usage Example



Hash Function Requirements

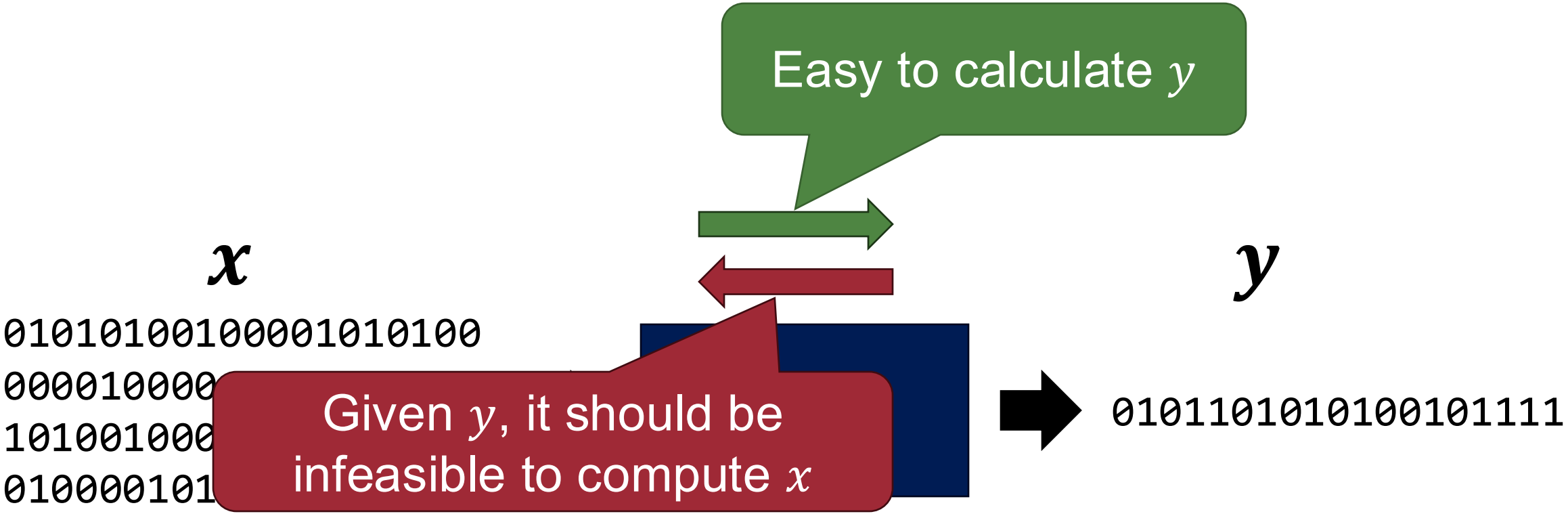
1. **Preimage resistant**
2. **Second preimage resistant**
3. **Collision resistant**
4. **Efficiency**: It is relatively easy to compute for any give input.

Property #1: Preimage Resistant

- Given y, computationally infeasible to find x such that $H(x) = y$
  - So-called one-way property

Property #1: Preimage Resistant

- Given y, computationally infeasible to find x such that $H(x) = y$
  - So-called one-way property



Property #1: Preimage Resistant

- Given y, computationally infeasible to find x such that $H(x) = y$
  - So-called one-way property

- Example:
  - Factoring: $H(x_1, x_2) = x_1 \times x_2$ where x_1, x_2 are prime numbers
  - Discrete logarithm: $H(x) = kx \bmod p$

Application: Password Storage

- Goal: store ID and password pairs to authenticate users
- **Bad approach**: store ID and password pairs in plaintext to a DB

ID	Password
Alice	1234abcd
Bob	verysecure
Charlie	1234abcd

Application: Hash-based Password Storage

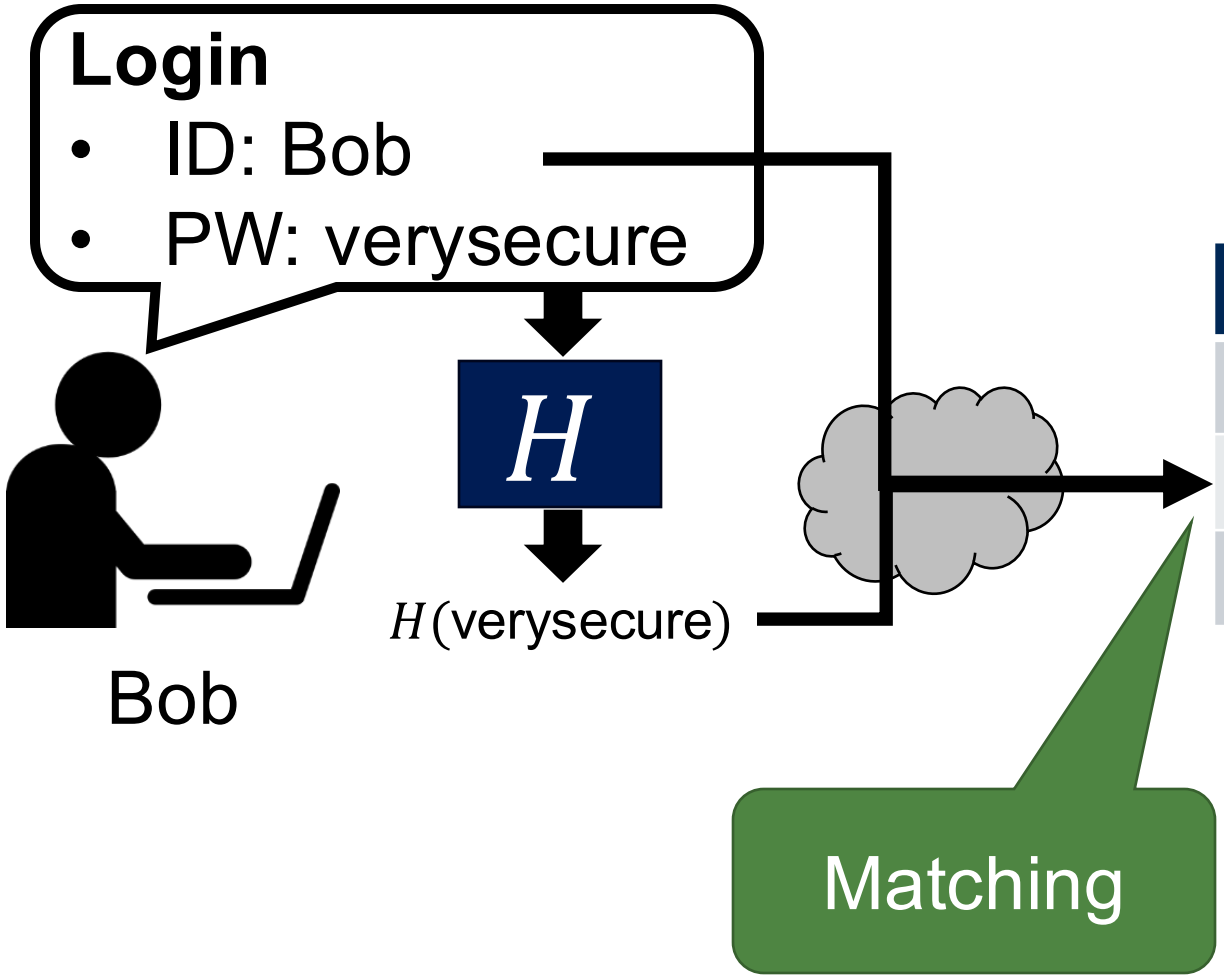
- Hashing passwords

ID	Password
Alice	H(1234abcd)
Bob	H(verysecure)
Charlie	H(1234abcd)

The attacker is not able to calculate "verysecure"

Application: Hash-based Password Storage

- Hashing passwords



ID	Password
Alice	H(1234abcd)
Bob	H(verysecure)
Charlie	H(1234abcd)

Application: Hash-based Password Storage

- Hashing passwords
- BTW, why do we need strong password requirements?

ID	Password
Alice	H(1234abcd)
Bob	H(verysecure)
Charlie	H(1234abcd)

Same password → Same hash value

Application: Salted Hash

- Hashing passwords
- BTW, why do we need strong password requirements?

=> **Salted Hash**: use a randomly generated number (a salt) to make a hash.

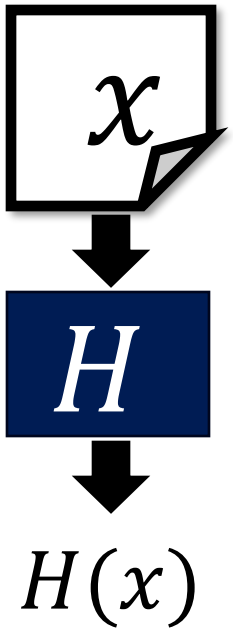
ID	Salt	Password
Alice	$\mathbf{23}$	$H(1234abcd, \mathbf{23})$
Bob	$\mathbf{51}$	$H(verysecure, \mathbf{51})$
Charlie	$\mathbf{97}$	$H(1234abcd, \mathbf{97})$

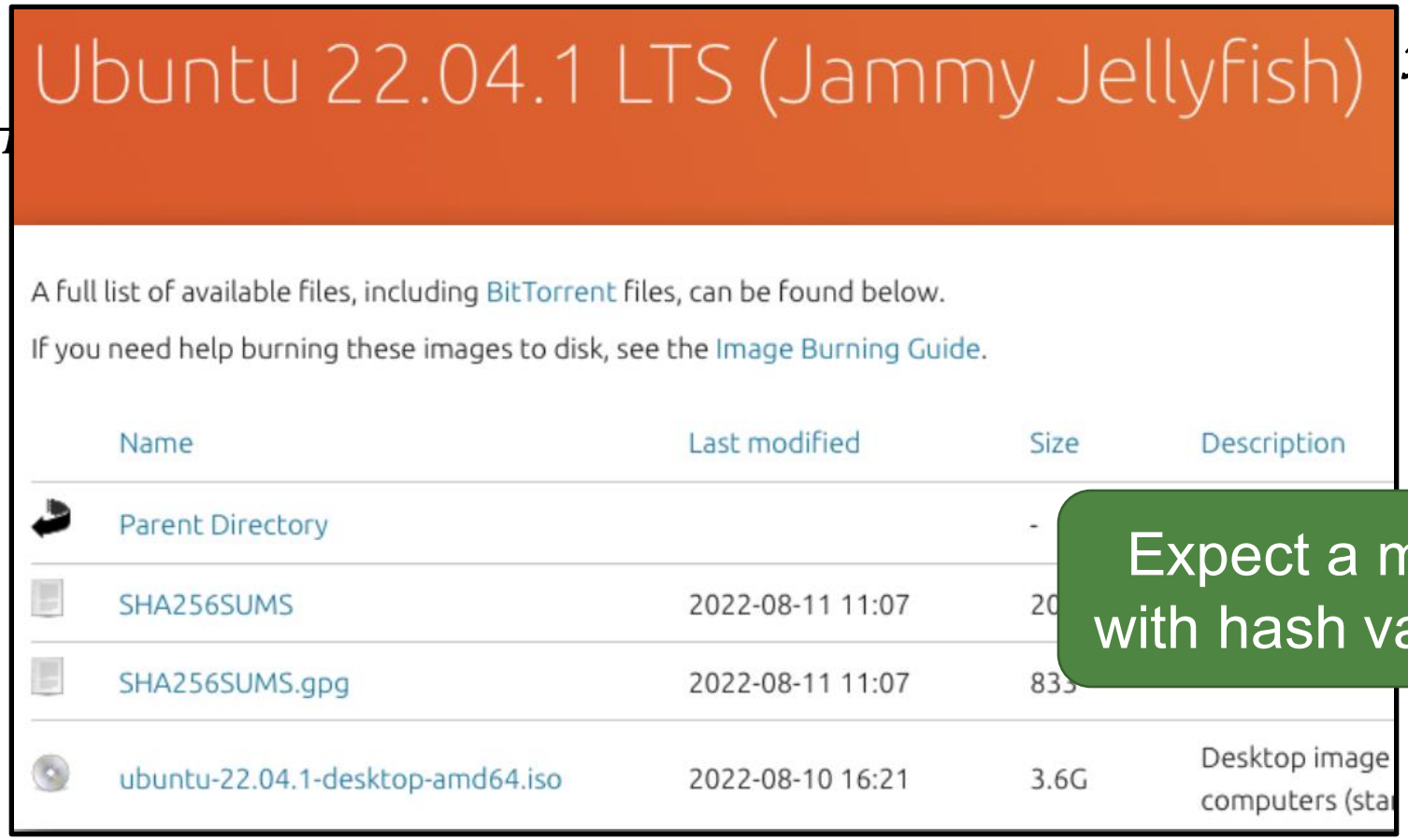
Property #2: Second Preimage Resistant

- Given x, computationally infeasible to find z such that $x \neq z$ and $H(x) = H(z)$

Property #2: Second Preimage Resistant

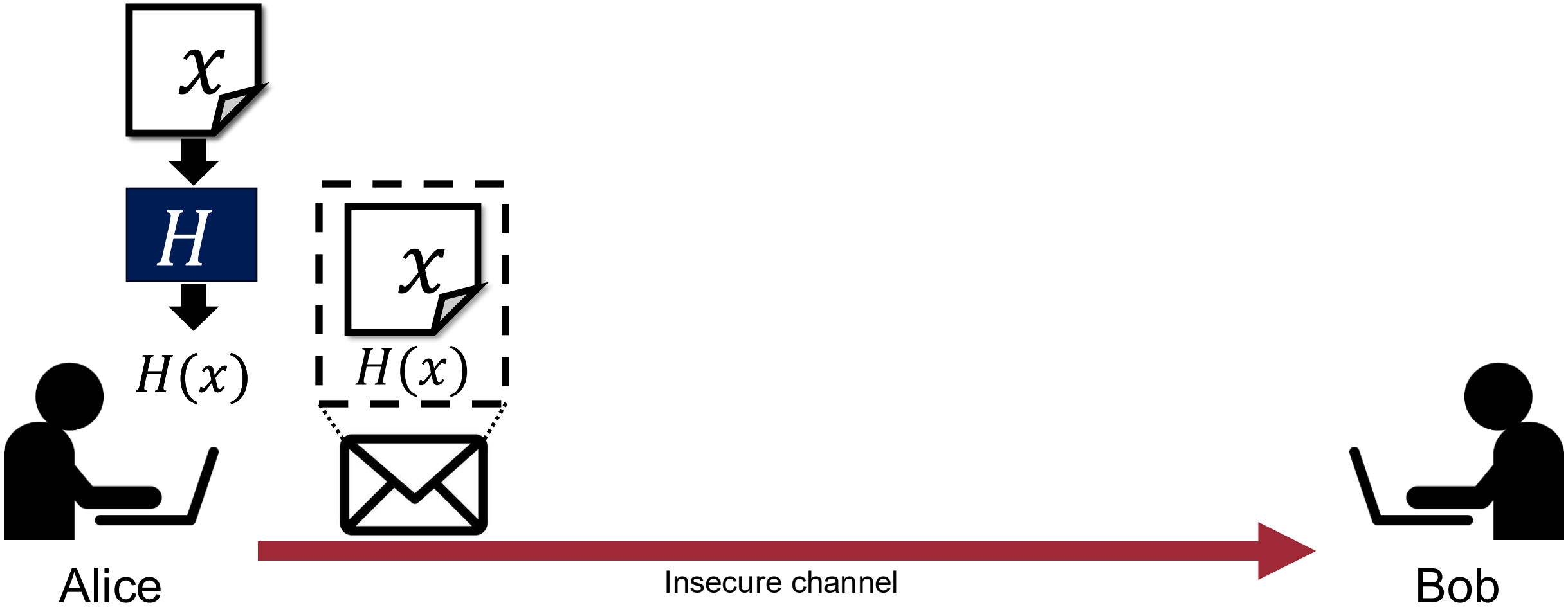
- Given x, $x \neq z$ and $H(x) = H$





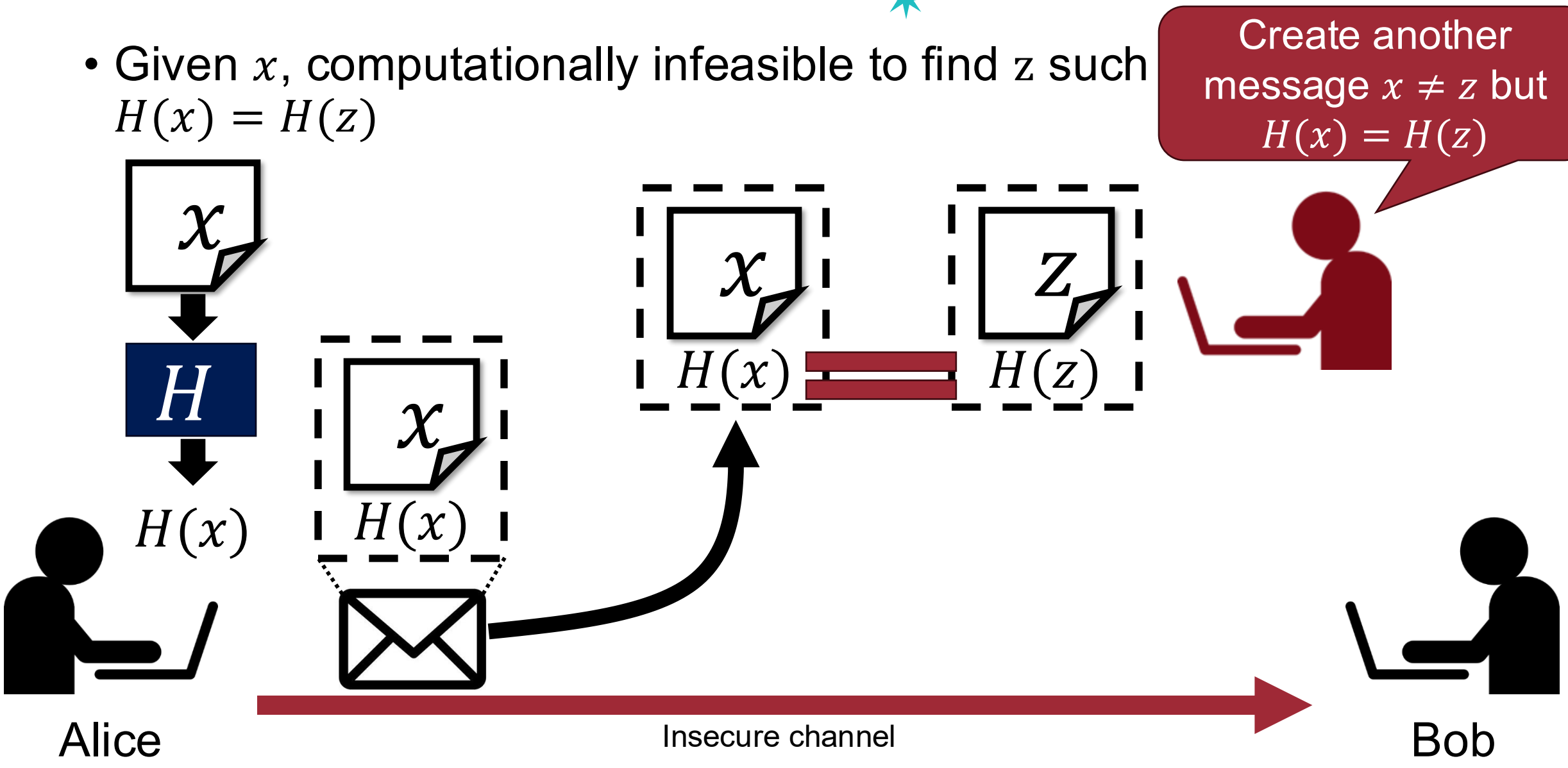
Property #2: Second Preimage Resistant

- Given x, computationally infeasible to find z such that $x \neq z$ and $H(x) = H(z)$



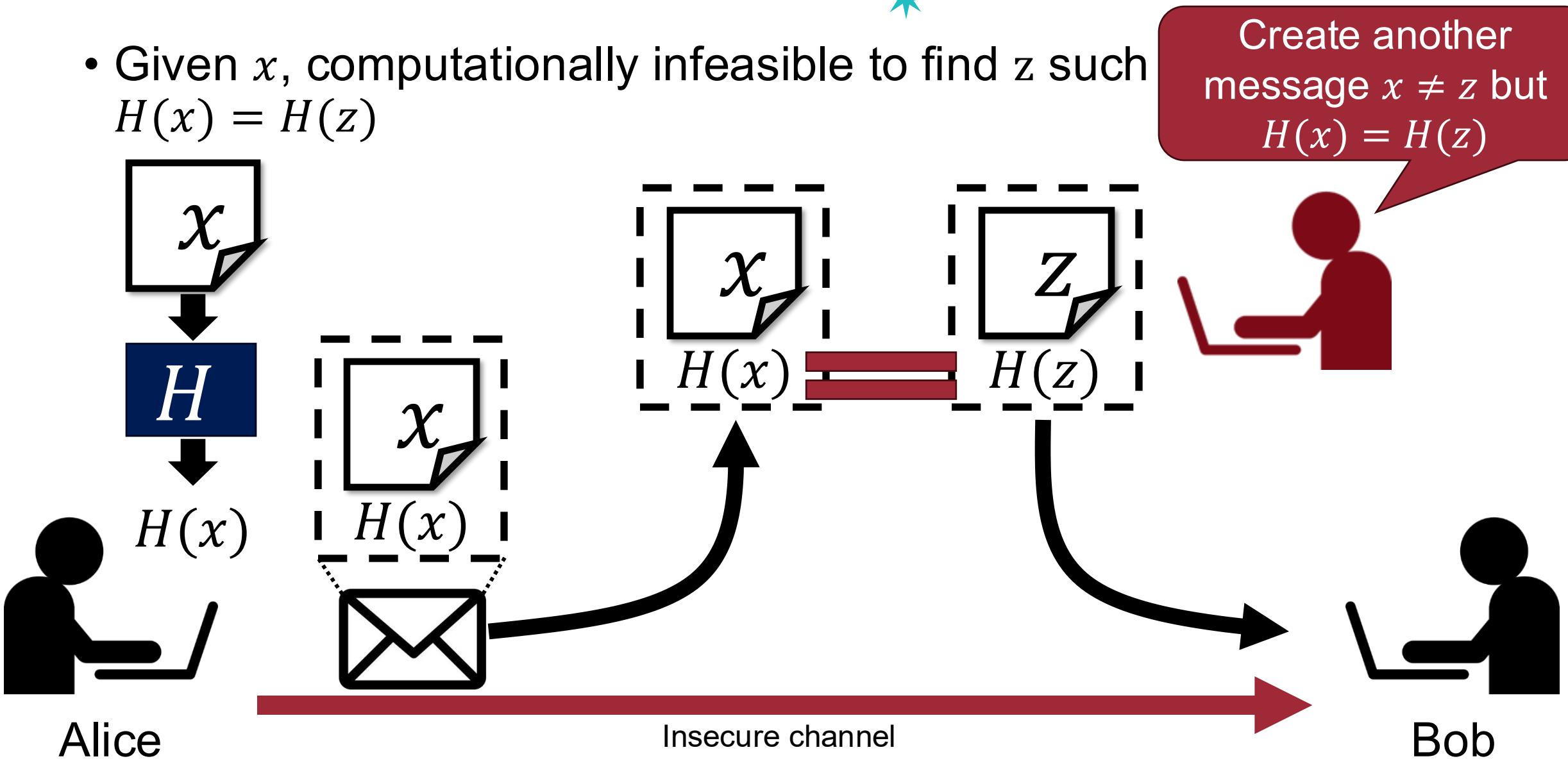
Property #2: Second Preimage Resistant

- Given x, computationally infeasible to find z such $H(x) = H(z)$



Property #2: Second Preimage Resistant

- Given x, computationally infeasible to find z such $H(x) = H(z)$



Property #2: Second Preimage Resistant

- Given x, computationally infeasible to find z such that $x \neq z$ and $H(x) = H(z)$

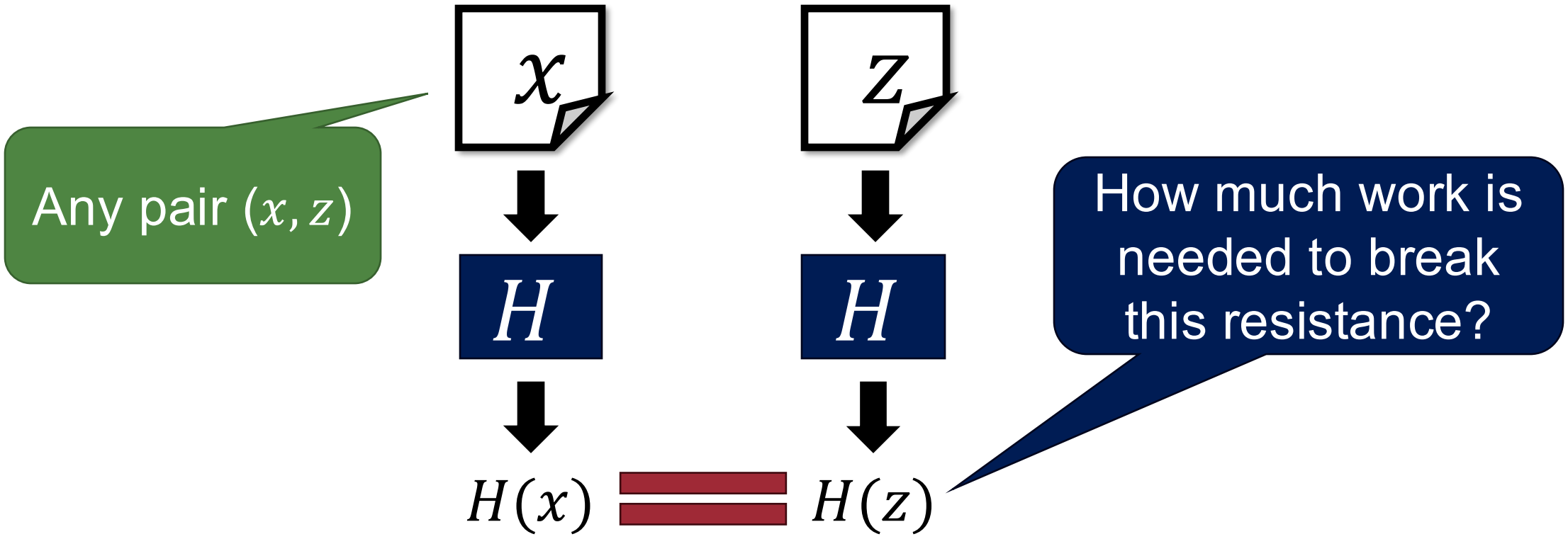
- Example: integrity of software distribution, fingerprinting (e.g., virus, deduplication)

Property #3: Collision Resistant

- Computationally infeasible to find **any pair (x, z)** such that $x \neq z$ and $H(x) = H(z)$

Property #3: Collision Resistant

- Computationally infeasible to find **any pair (x, z)** such that $x \neq z$ and $H(x) = H(z)$



Birthday Paradox

How many people must be in a group, such that there is more than 50% probability that at least two of them have the same birthday?

=> 23 people

(Birthday paradox)

Birthday Paradox

Probability that in a set of n random people, at least two will share a birthday

- Find n such that $p(n) \geq 0.5$
  - # of people in the group: n
  - A year has 365 days
- $p(n) = 1 - \overline{p}(n)$

Probability that all n people have different birthdays

Birthday Paradox

Probability that in a set of n random people, at least two will share a birthday

- Find n such that $p(n) \geq 0.5$
  - # of people in the group: n
  - A year has 365 days
- $p(n) = 1 - \overline{p}(n)$

Probability that all n people have different birthdays

- If $n = 3$,

$$\mathrm{P}(3) = 1 - \left(\frac{365}{365} \times \frac{364}{365} \times \frac{363}{365}\right) = 1 - 0.9917 = 0.0083$$

Birthday Paradox

Probability that in a set of n random people, at least two will share a birthday

- Find n such that $p(n) \geq 0.5$
  - # of people in the group: n
  - A year has 365 days
- $p(n) = 1 - \bar{p}(n)$

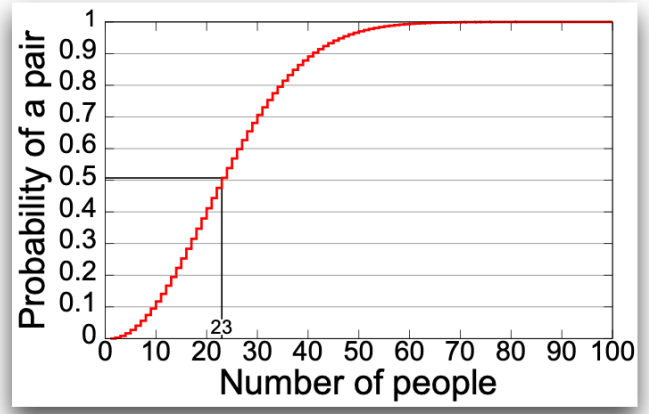
Probability that all n people have different birthdays

- If $n = 3$,

$$\mathrm{P}(3) = 1 - \left(\frac{365}{365} \times \frac{364}{365} \times \frac{363}{365}\right) = 1 - 0.9917 = 0.0083$$

$$\bar{p}(n) = 1 \times \left(1 - \frac{1}{365}\right) \times \left(1 - \frac{2}{365}\right) \times \cdots \times \times \left(1 - \frac{n-1}{365}\right)$$

$$= \frac{365 \times 364 \times \cdots \times (365 - n + 1)}{365^n}$$



Birthday Paradox

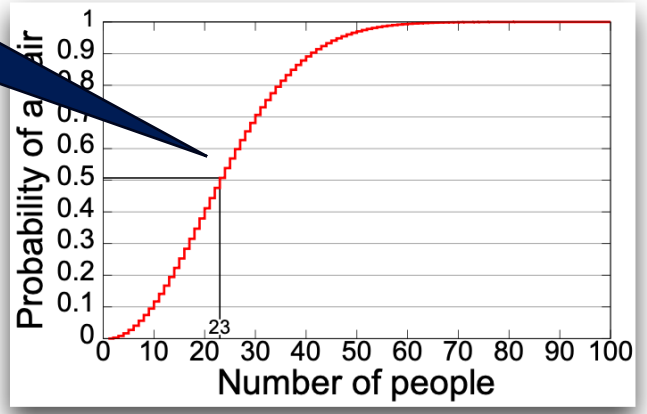
- Find n such that $p(n) \geq 0.5$
  - # of people in the group: n
  - A year has 365 days
- $p(n) = 1 - \bar{p}(n)$
- If $n = 3$,

$$\mathrm{P}(\qquad \qquad) = 1 - 0.9917 = 0.0083$$

If we put $n = 23$, the probability is 50.7%

$$\bar{p}(n) = 1 \times \left(1 - \frac{\quad}{365}\right) \times \left(1 - \frac{\quad}{365}\right) \times \cdots \times \times \left(1 - \frac{\quad}{365}\right)$$

$$= \frac{365 \times 364 \times \cdots \times (365 - n + 1)}{365^n}$$



Property #3: Collision Resistant

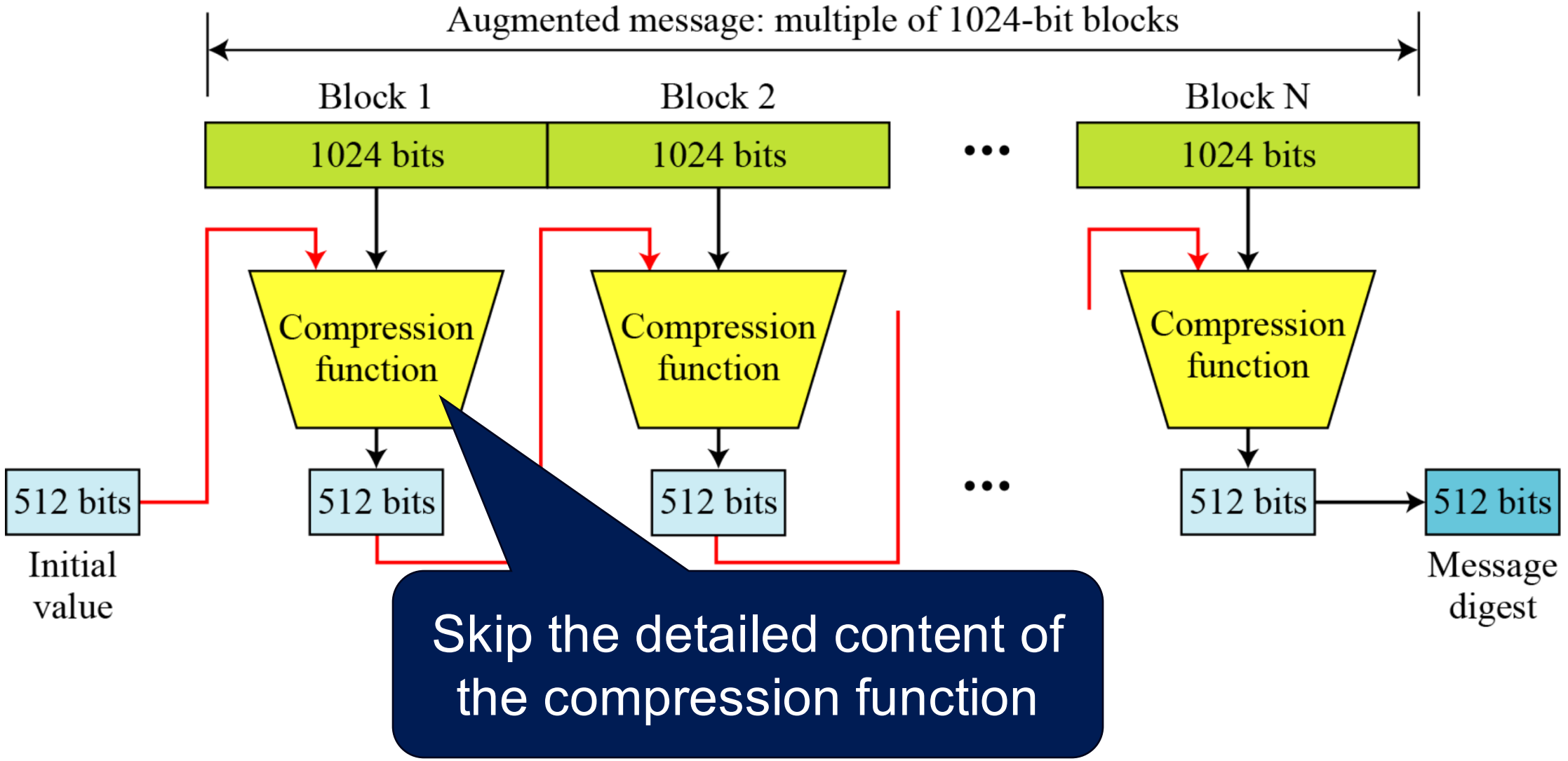
- Computationally infeasible to find **any pair (x, z)** such that $x \neq z$ and $H(x) = H(z)$

- ***Birthday attack***: If we have an m bit hash value, $\mathbf{2^{m/2}}$ **work** is needed to break collision resistant (not 2^m, birthday paradox)
  - To ensure security against 2^n attacks, the hash output length must be $2n$-bits

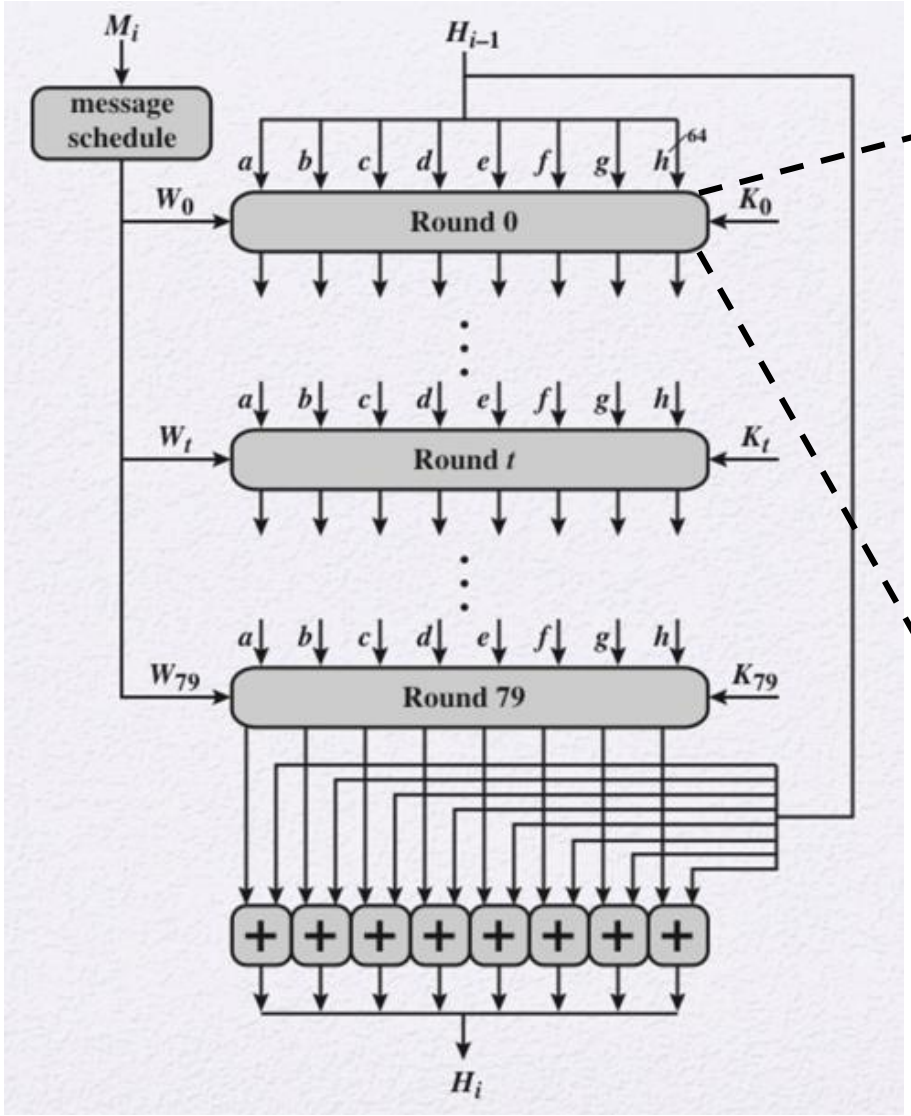
Hash Function Standards

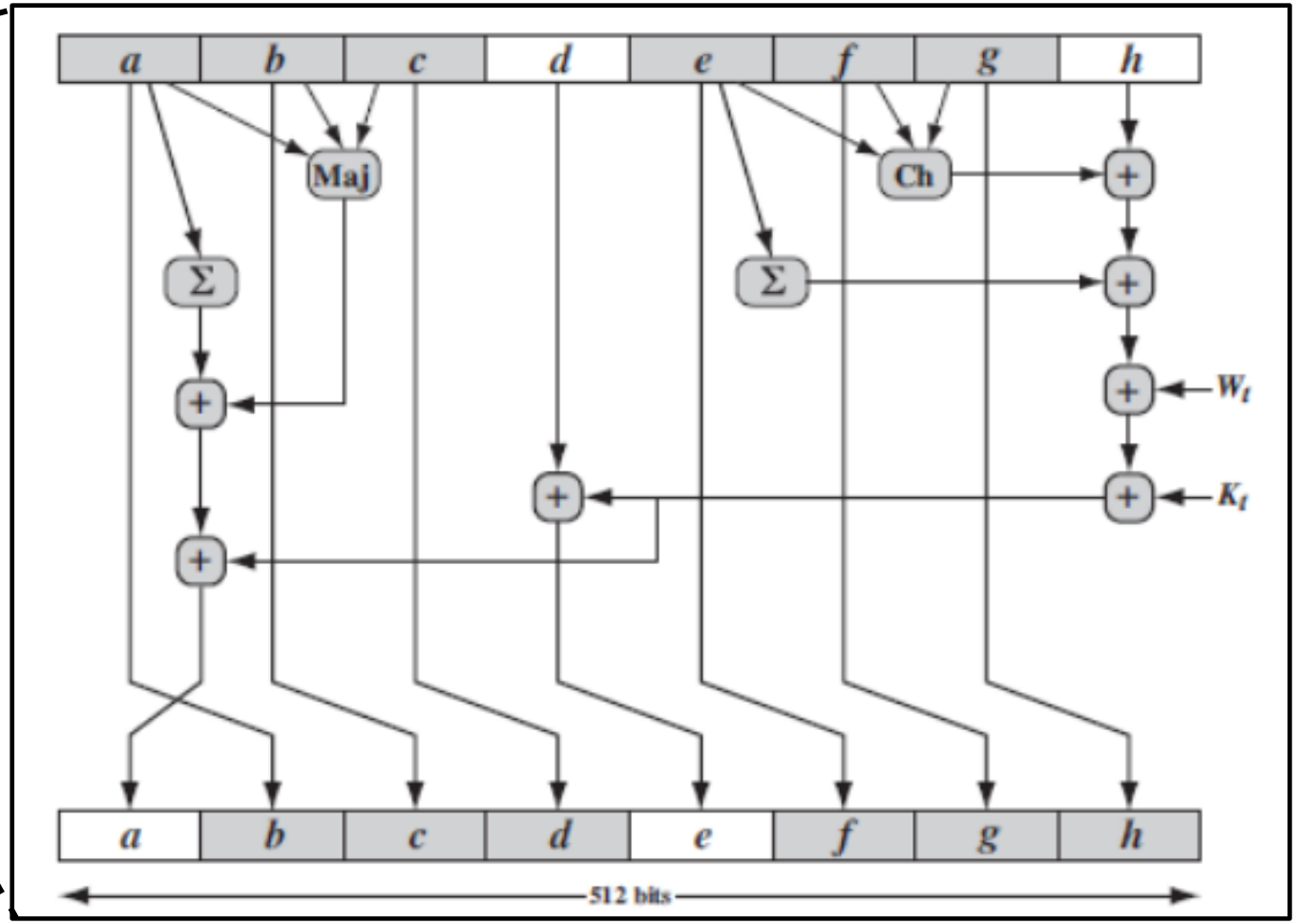
- MD5
  - Pairs of collisions reported
  - Still used for simple data diffing
- SHA-1
  - Pairs of collisions reported
  - Broken
- SHA-256, SHA-384, SHA-512 (message digest size)

SHA-512 Overview

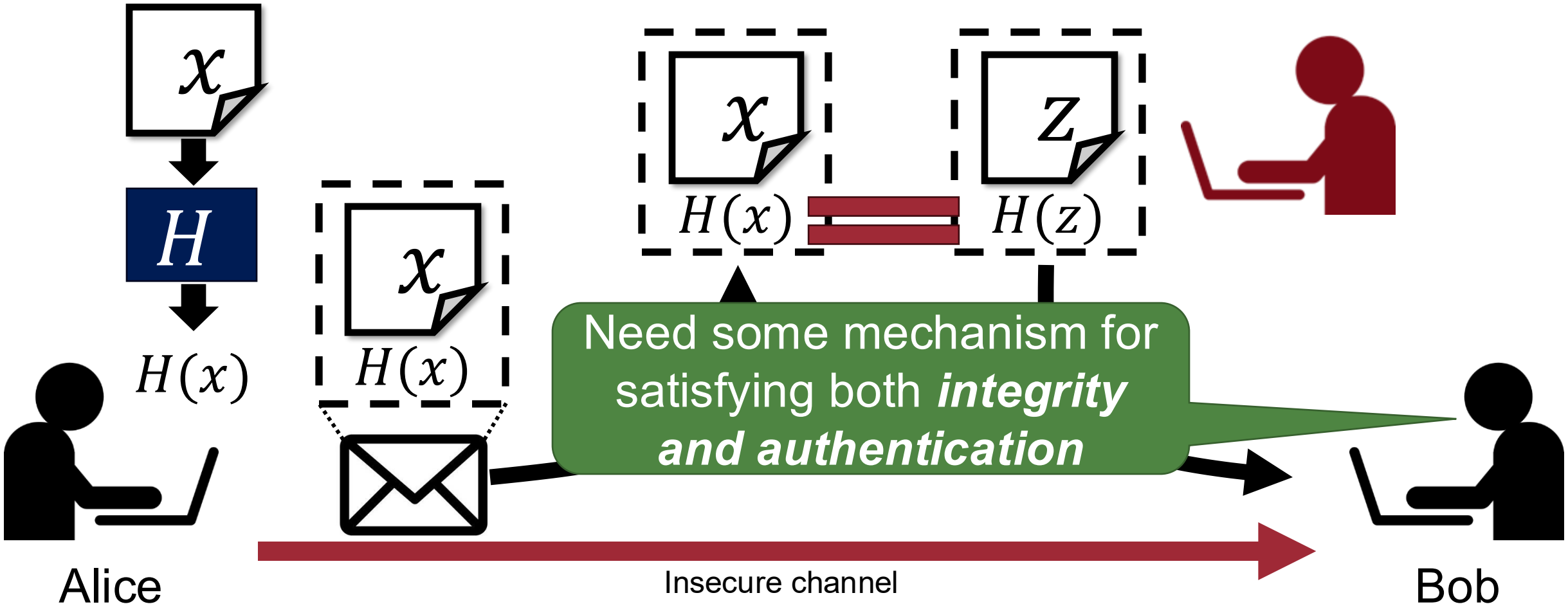


(Skip) Compression Function





Recap: Second Preimage Resistant



Motivation

- In the case of *asymmetric cryptography*, integrity and authentication can be ensured through hash-based digital signatures

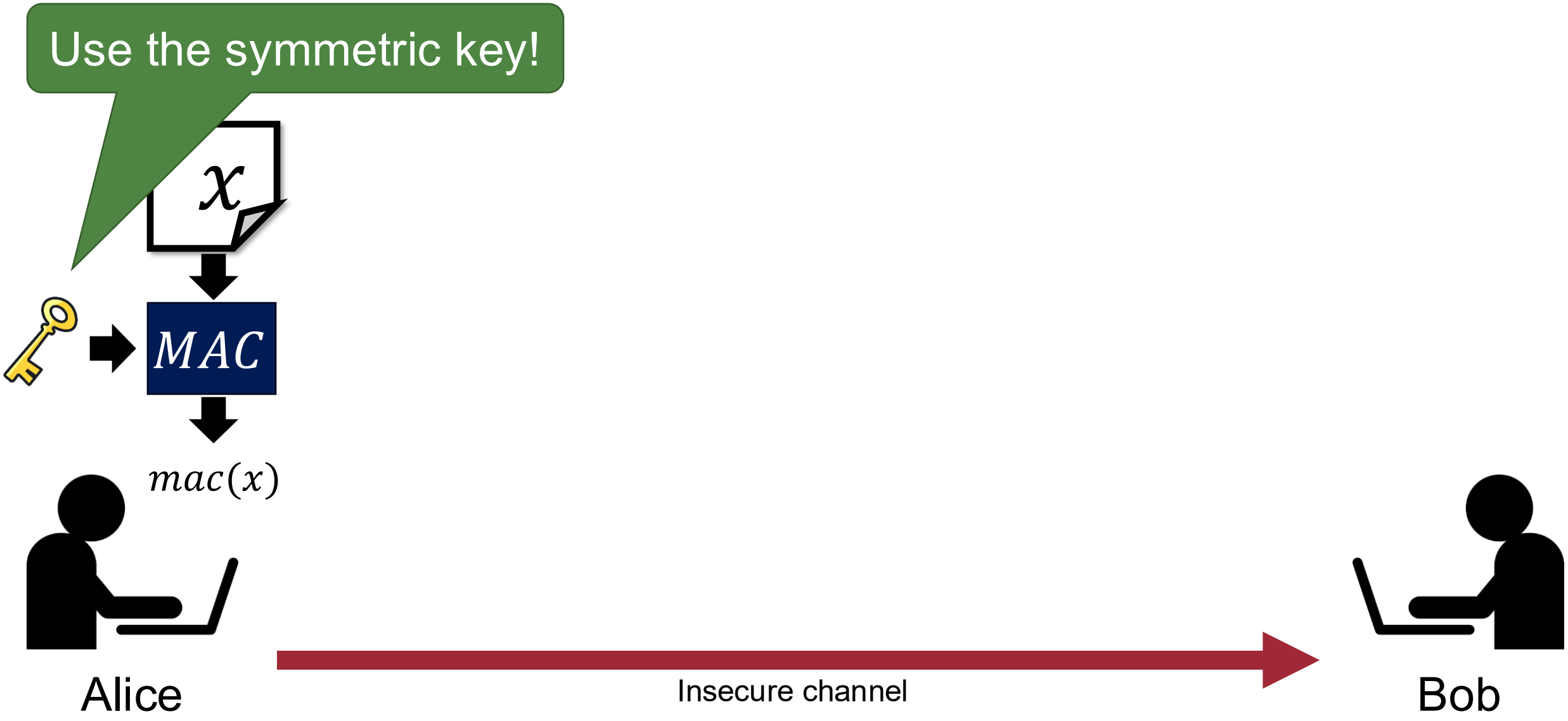
- Q. In the case of *symmetric cryptography*, how can both integrity and authentication be ensured?

→ Message Authentication Codes (MAC)

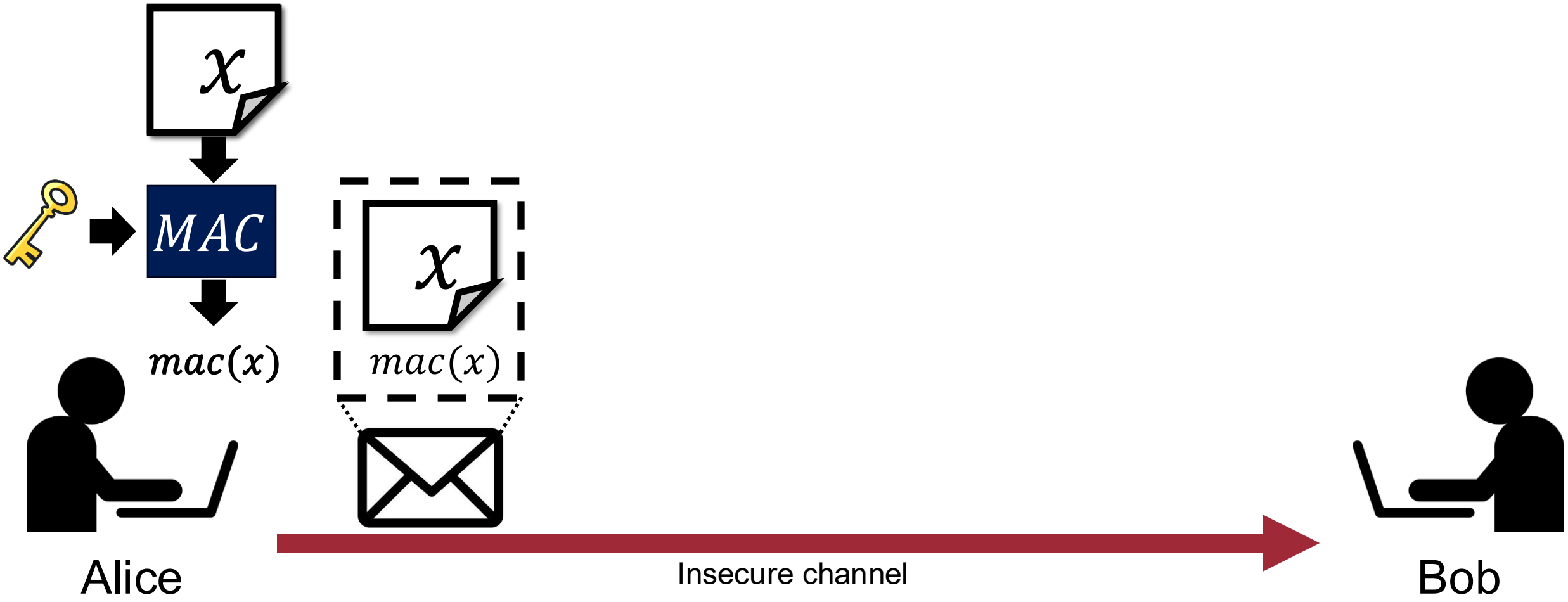
Message Authentication Codes (MAC)

“Cryptographic checksum” to ensure the **integrity** of the message and the data origin **authentication** (in symmetric-key cryptography)

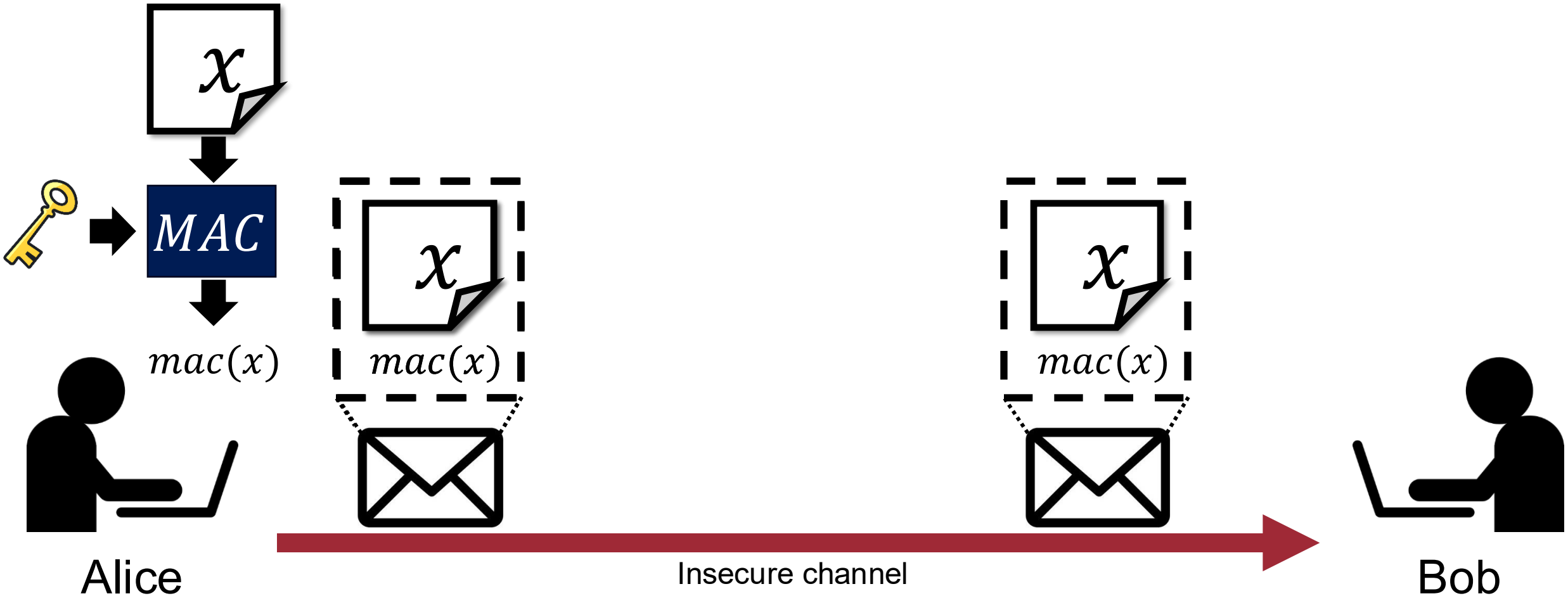
Message Authentication Codes (MAC)



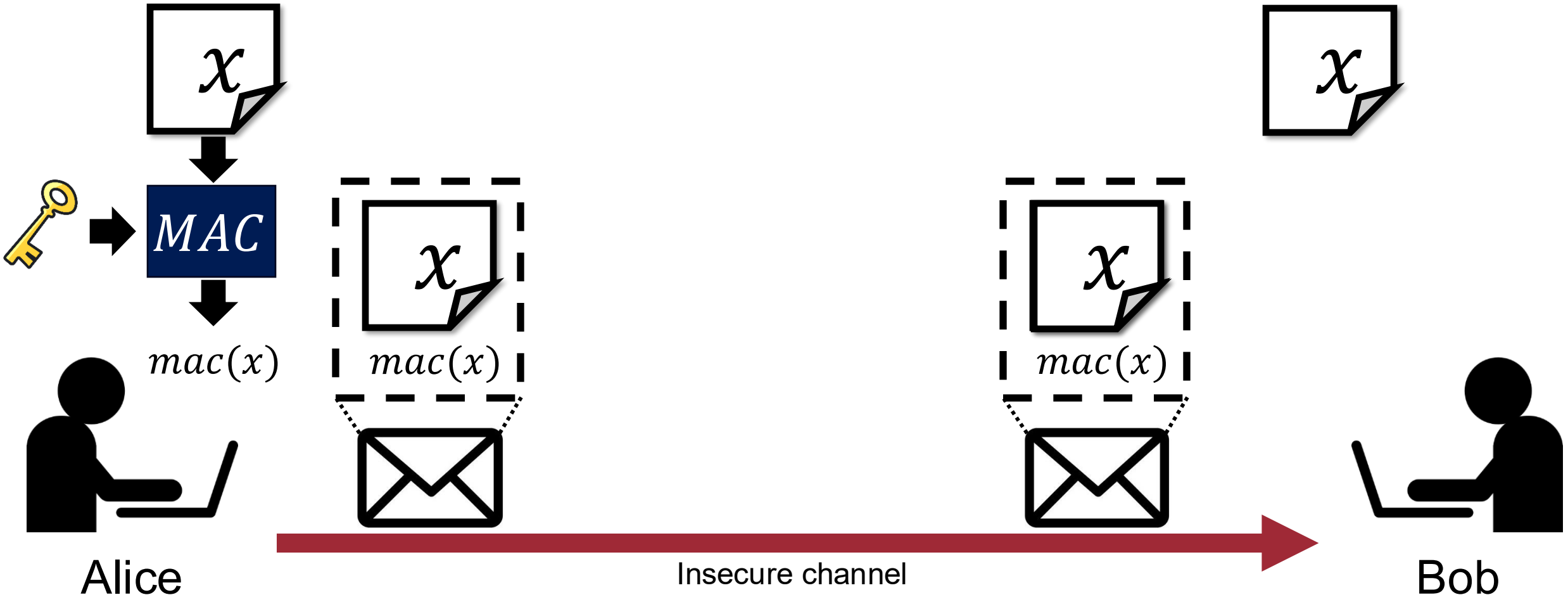
Message Authentication Codes (MAC)



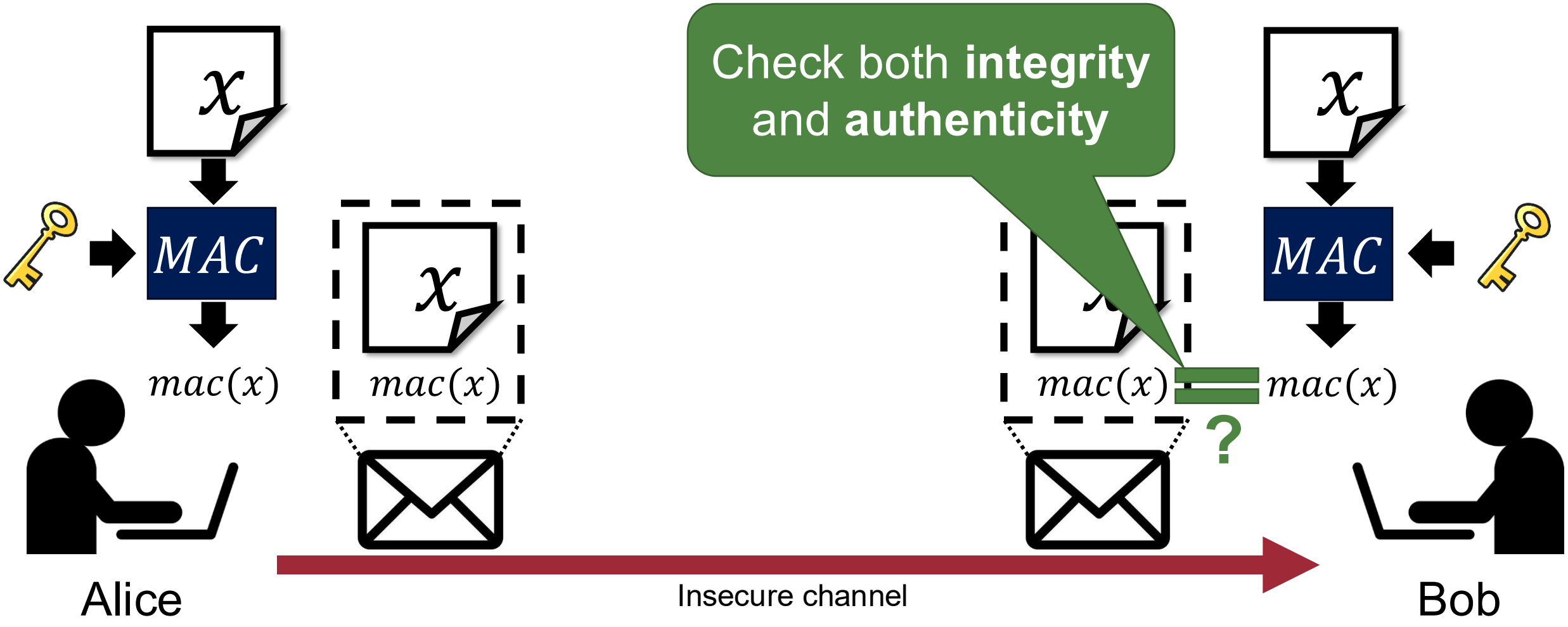
Message Authentication Codes (MAC)



Message Authentication Codes (MAC)



Message Authentication Codes (MAC)



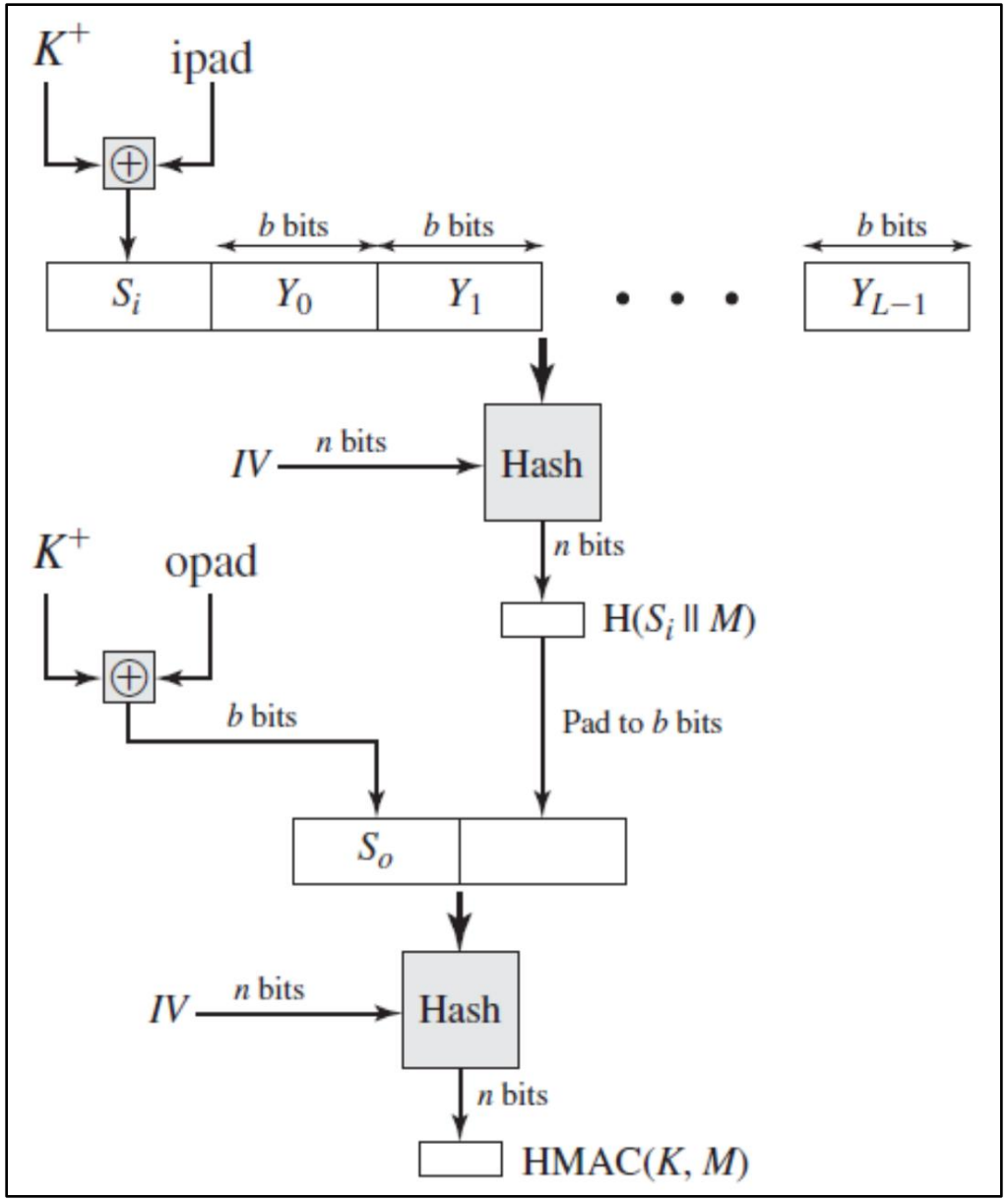
Message Authentication Codes (MAC)

“Cryptographic checksum” to ensure the **integrity** of the message and the data origin **authentication** (in symmetric-key cryptography)

- CBC-MAC, CMAC, OMAC, HMAC, …

MAC Algorithm Example: HMAC

- For your information ☺



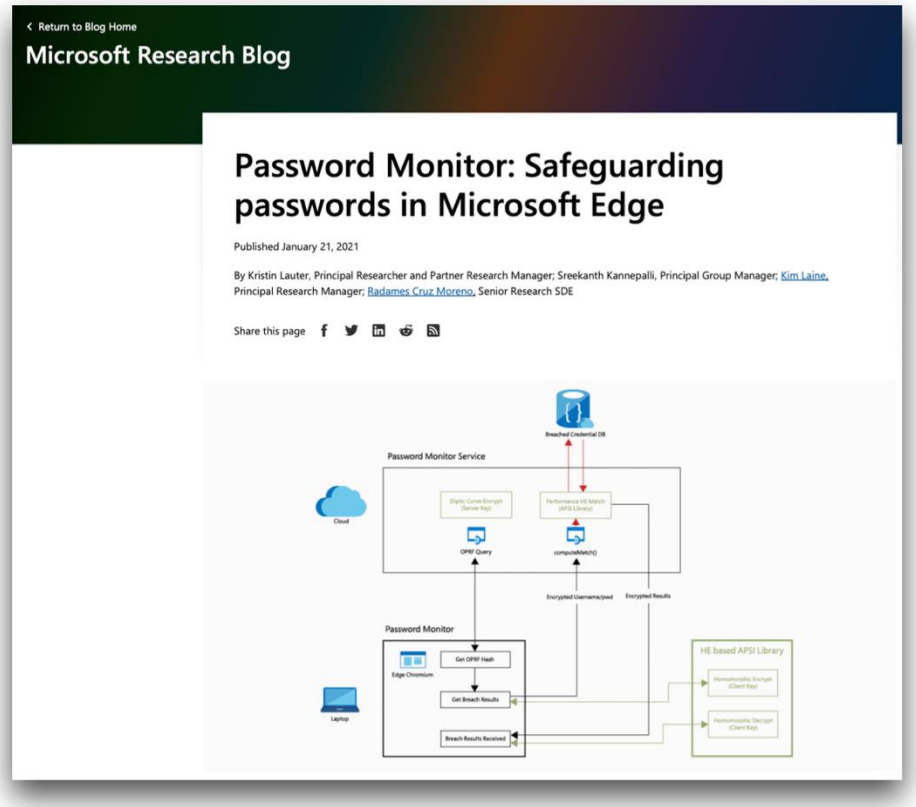
$$HMAC_K(m) = H((K' \oplus opad)||H((K' \oplus ipad)||m))$$

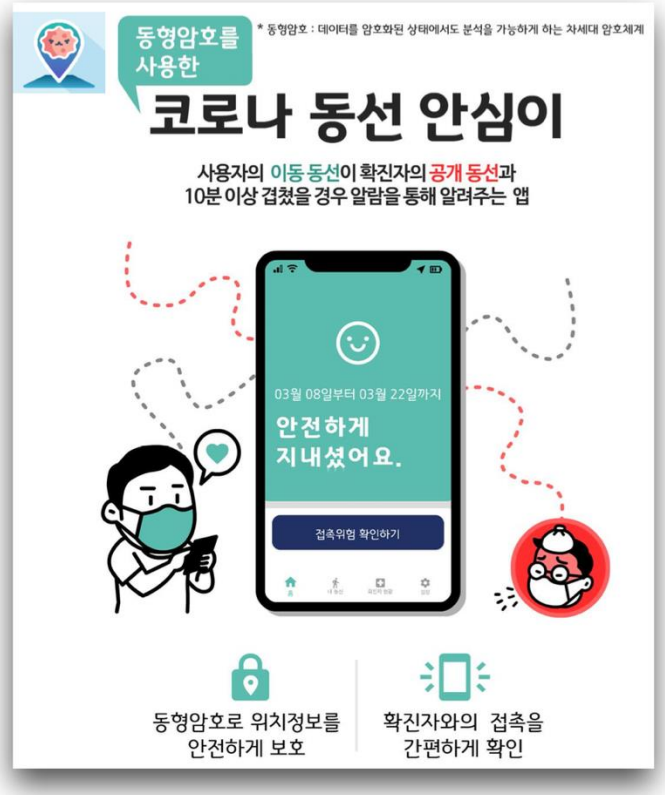
Holy Grail of Cryptography

- Is it possible to provide a secure public service?
  - i.e., computations on encrypted data
- Example
  - Average GPA in the class with encrypted individual GPAs
  - Covid-19 alert with encrypted location information
  - Election with encrypted votes
- Necessary property: **homomorphism**
  - $Dec(c1 \oplus c2) = Dec(c1) \oplus Dec(c2)$

Homomorphic Encryption (동형 암호)

- Allows computations on encrypted data
- "*A Fully Homomorphic Encryption Scheme*", C. Gentry, 2009
- Applications:





A Simplified Symmetric Homomorphic Encryption

- **Plaintext space:** $\{0,1\}$
- **Secret key:** p
- **Random numbers:** q and ϵ
- **Encryption:** $Enc(m) = m + pq + 2\epsilon$
- **Decryption**: $Dec(c) = (c \bmod p) \bmod 2$
- **Homomorphism**
  - $Dec\big(Enc(m1) + Enc(m2)\big) = Dec\big(Enc(m1 + m2)\big) = m1 + m2$
  - $Dec(Enc(m1) \times Enc(m2)) = Dec(Enc(m1 \times m2)) = m1 \times m2$

Summary

- Public-Key Infrastructure
  - Certificate Authority (CA)
  - Digital Certificate
  - Chain of trust
- Cryptographic Hash Functions
  - Preimage resistant
  - Second preimage resistant
  - Collision resistant
- Message Authentication Codes (MAC)
  - Check both integrity and authenticity for symmetric key environment
- Homomorphic Encryption

Question?