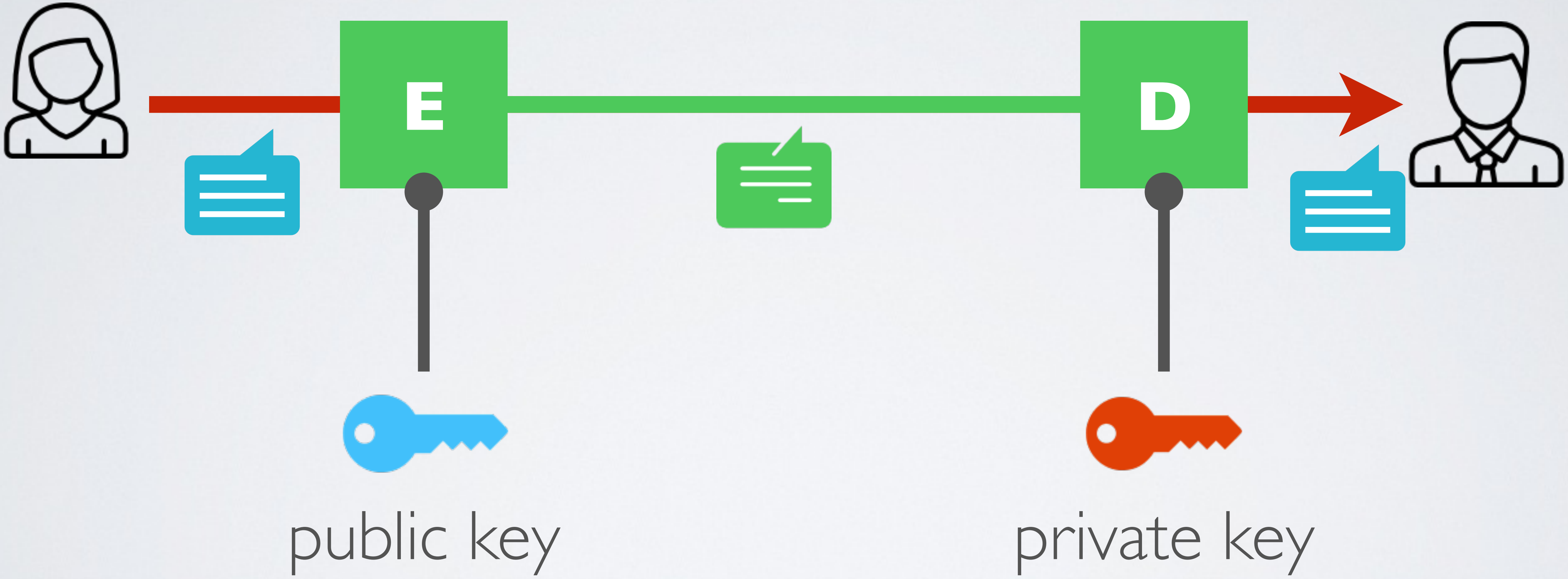
Introductory Cryptography

Asymmetric Encryption

Kc Udonsi

Asymmetric encryption
a.k.a Public Key Cryptography



Asymmetric keys

Ks_A, Kp_A
Kp_A
Kp_A

Alice generates a pair of asymmetric keys

- $\mathbf{Ks_A}$ is the secret key that Alice keeps for herself
- $\mathbf{Kp_A}$ is the public key that Alice gives to everyone (even Mallory)

➡ These two keys $\mathbf{Ks_A}$ and $\mathbf{Kp_A}$ work together

Asymmetric Keys - Functional Requirements

$D_{Ks}(E_{Kp}(m)) = m$ and $D_{Kp}(E_{Ks}(m)) = m$ for every pair (Kp, Ks)

✓ Generating a pair (Kp, Ks) is easy to compute (polynomial)

✓ Encryption is easy to compute (either polynomial or linear)

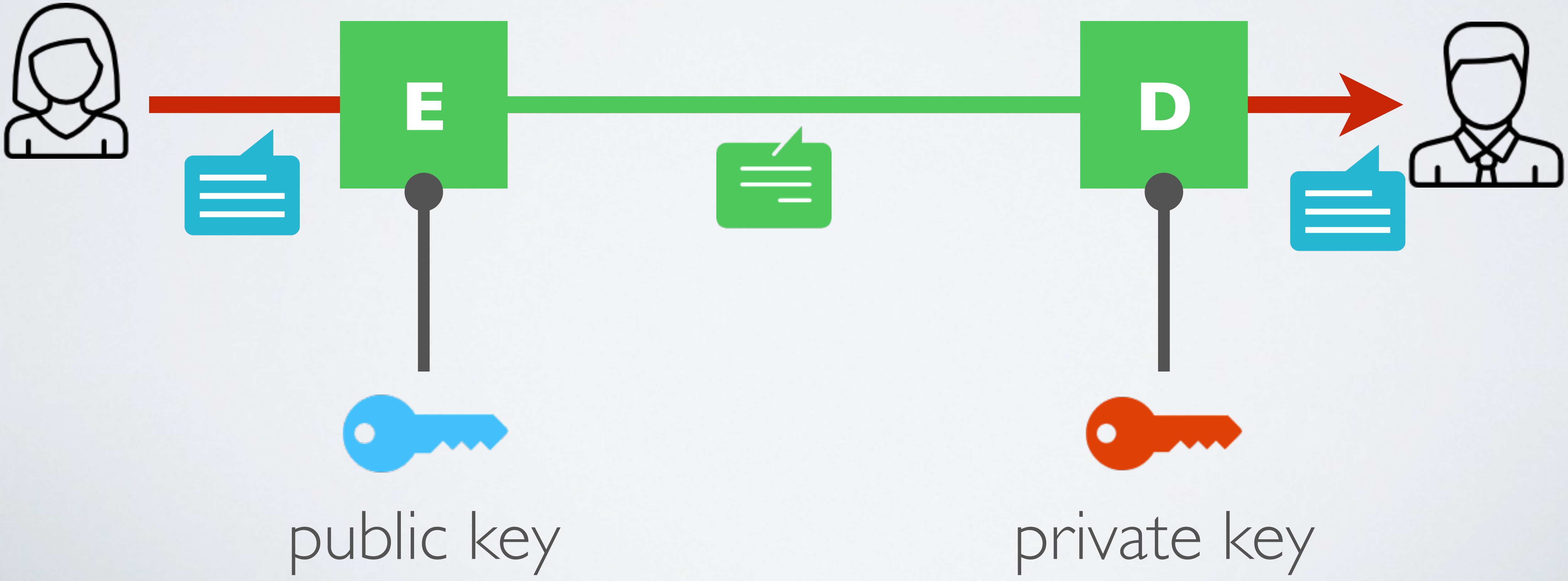
✓ Decryption is easy to compute (either polynomial or linear)

◉ Finding a matching key **Ks** for a given **Kp** is hard (exponential)

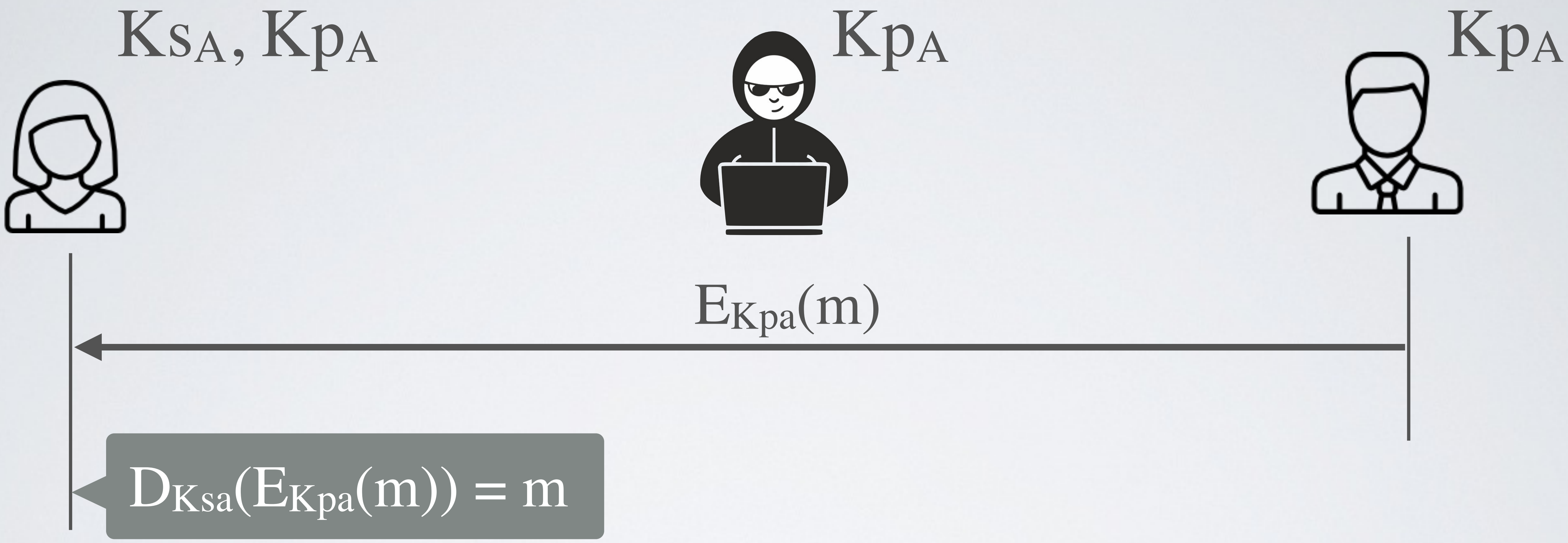
◉ Decryption without knowing the corresponding key is hard (exponential)

Asymmetric encryption
a.k.a Public Key Cryptography

- ➡ The public key for encryption
- ➡ The private key for decryption



Asymmetric encryption for **confidentiality**



Ks_A, Kp_A

Kp_A

Kp_A

$E_{Kpa}(m)$

$D_{Ksa}(E_{Kpa}(m)) = m$

Bob encrypts a message **m** with Alice's public key **Kp_A**

➡ Nobody can decrypt **m**, except Alice with her private key **Ks_A**

✓ Confidentiality without the need to exchange a secret key

RSA - Rivest, Shamir and Alderman

Key Size	1024 - 4096
Speed	~ factor of 10^6 cycles / operation
Mathematical Foundation	Prime number theory

Most widely used to secure network traffic

Adopted in 1977

Computational Complexity

Easy problems with prime numbers

- Generating a prime number p
- Addition, multiplication, exponentiation
- Inversion, solving linear equations

Hard problem with prime numbers

- Factoring primes
  e.g. given $\mathbf{n}$ find $\mathbf{p}$ and $\mathbf{q}$ such that $\mathbf{n = p \,.\, q}$

RSA - generating the key pair

1. Pick p and q two large prime numbers and calculate $n = p . q$ (see primality tests)
2. Compute $z = (p-1).(q-1)$
3. Pick a prime number $e < z$ such that e and z are relative primes

➡ (e,n) is the **public key**

4. Solve the linear equation $e * d = 1 \ (\bmod z)$ to find d

➡ (d,n) is the **private key**
however p and q must be kept secret too

RSA - encryption and decryption

Given $\mathbf{Kp} = (e, n)$ and $\mathbf{Ks} = (d,n)$

➡ Encryption : $E_{kp}(m) = m^e \bmod n = c$

➡ Decryption : $D_{ks}(c) = c^d \bmod n = m$

➡ $\mathbf{(m^e)^d \bmod n = (m^d)^e \bmod n = m}$

Other asymmetric cryptography schemes

Diffie-Hellman (precursor)

➡ No Authentication but good for key-exchange

El-Gamal

➡ Good properties for homomorphic encryption

Elliptic Curve Cryptography (widely used nowadays)

➡ Fast and small keys (190 bits equivalent to 1024 bits RSA)

Asymmetric vs Symmetric

	Symmetric	Asymmetric
pro	Fast	No key agreement
cons	Key agreement	Very slow

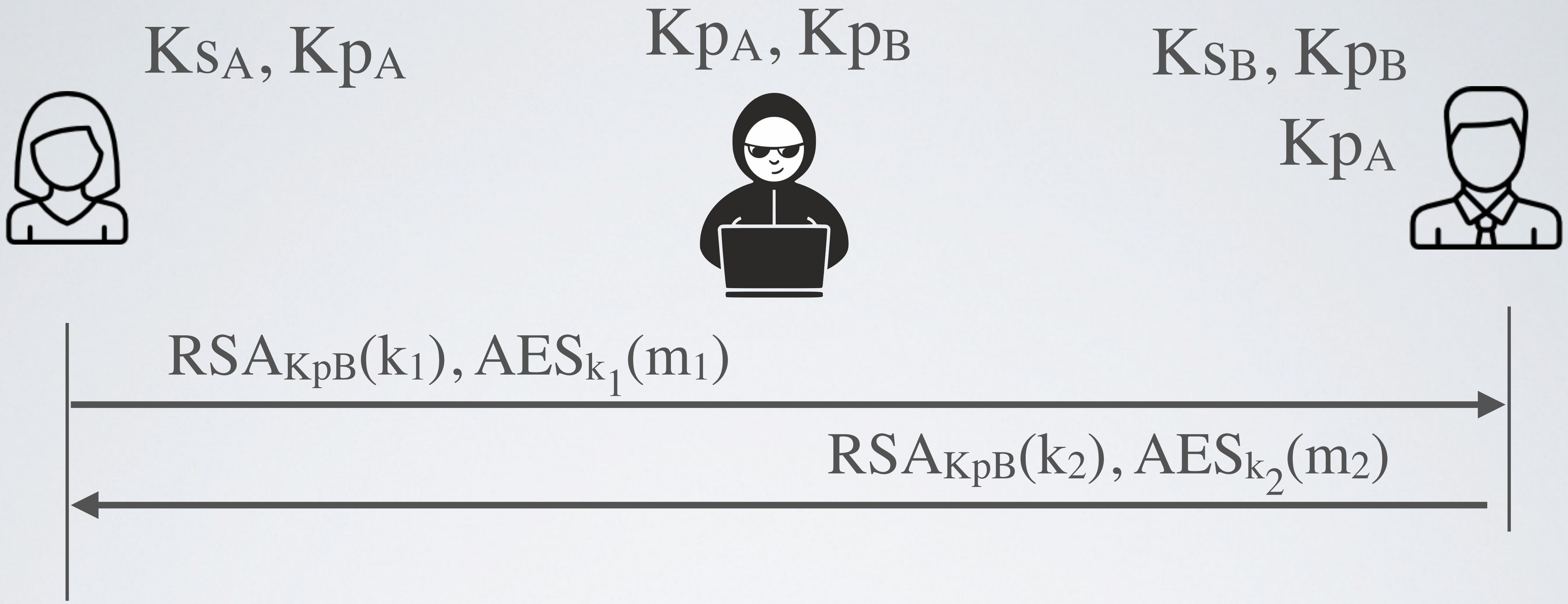
The best of both worlds

- ➡ Use RSA to encrypt a shared key
- ➡ Use AES to encrypt message

$$\boxed{E_{Kp}(m) = RSA_{Kp}(k), AES_{k}(m)}$$

Naive approach

But not perfect yet



Ks_A, Kp_A

Kp_A, Kp_B

Ks_B, Kp_B

Kp_A

$RSA_{Kp_B}(k_1), AES_{k_1}(m_1)$

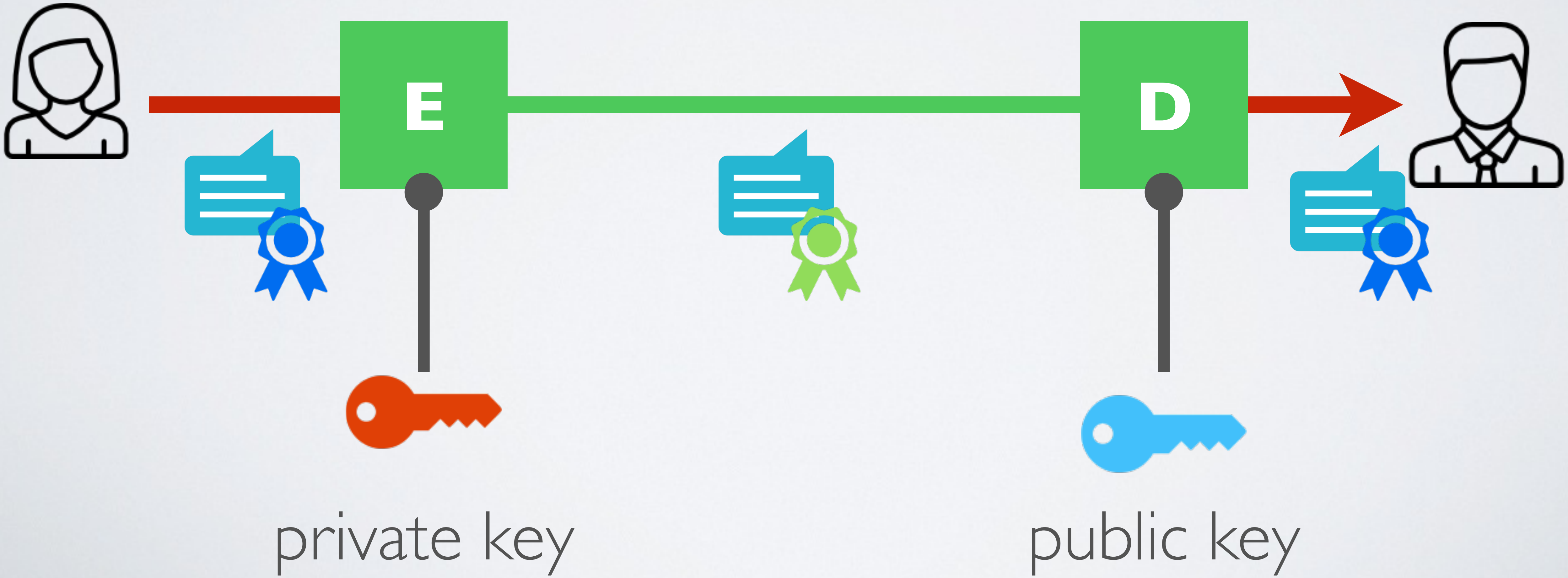
$RSA_{Kp_B}(k_2), AES_{k_2}(m_2)$

✓ Does ensure the confidentiality of the communication

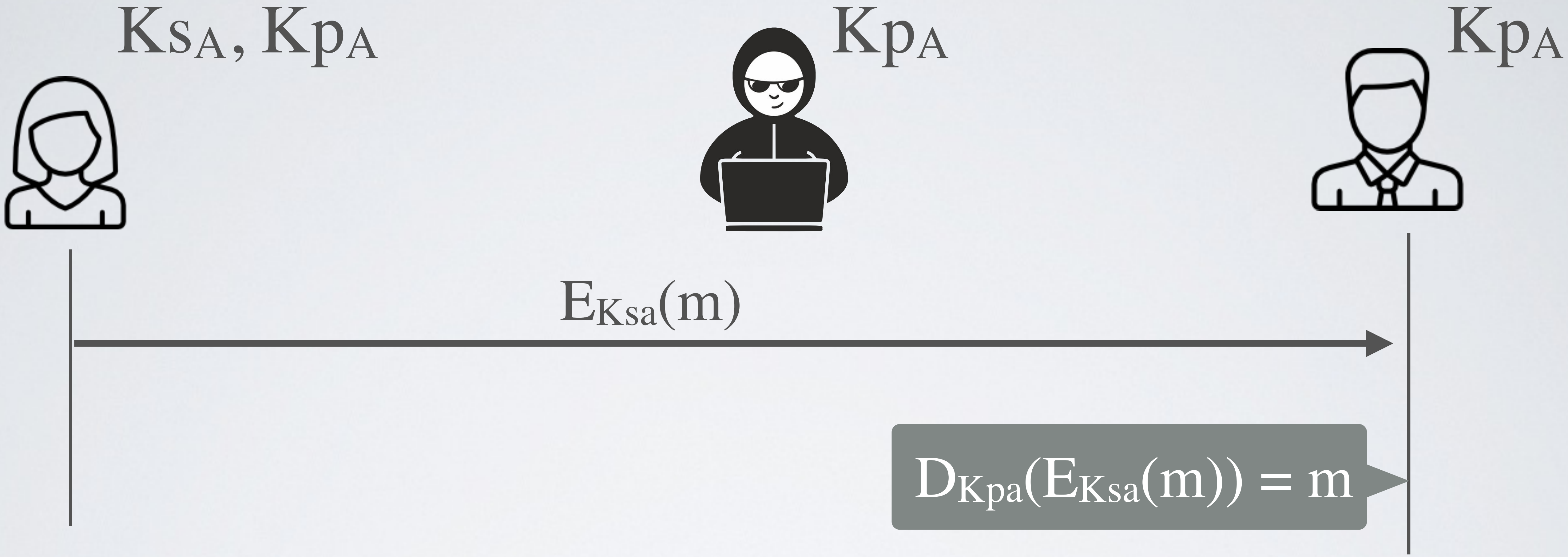
◉ Does not authenticate Alice or Bob

Asymmetric encryption: Digital Signature

- ➡ The private key for encryption
- ➡ The public key for decryption



Asymmetric encryption for **integrity**



Ks_A, Kp_A

Kp_A

Kp_A

$E_{Ksa}(m)$

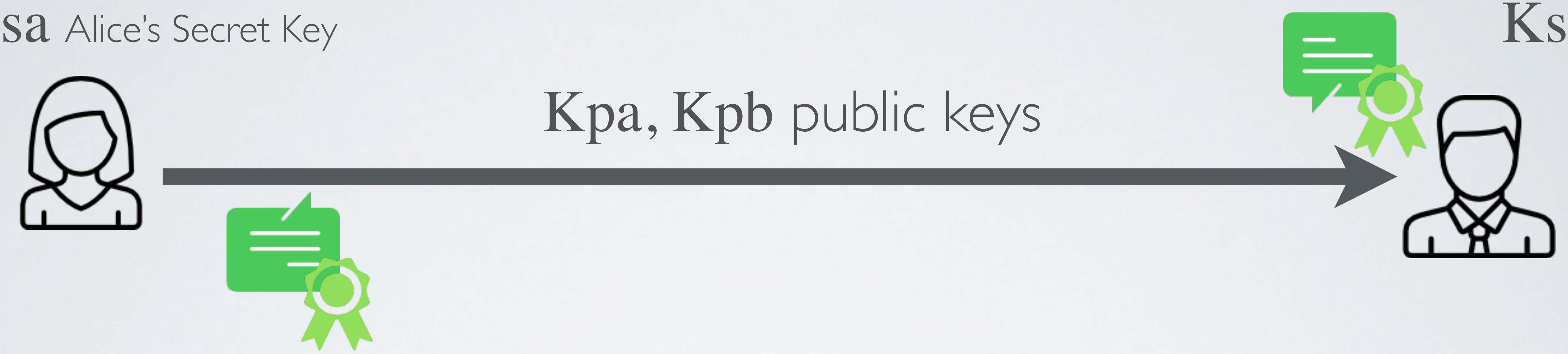
$D_{Kpa}(E_{Ksa}(m)) = m$

Alice encrypts a message **m** with her private key $\mathbf{Ks_A}$

➡ Everybody can decrypt **m** using Alice's public key $\mathbf{Kp_A}$

✓ Authentication with non-repudiation (a.k.a Digital Signature)

Digital Signature



Ksa Alice's Secret Key

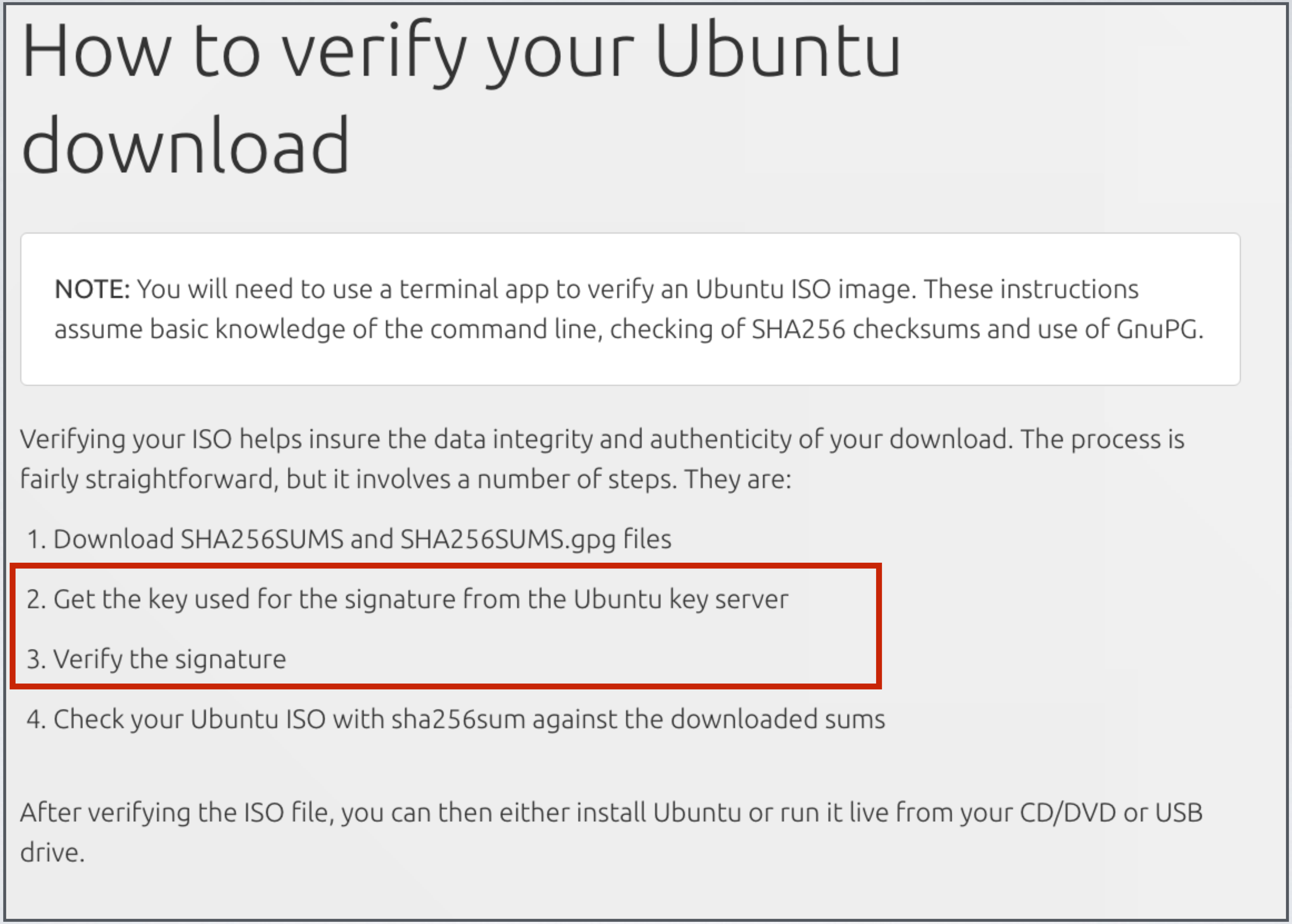
Kpa, Kpb public keys

Ksb

- ➡ Use public cryptography to **sign and verify**

$$m \parallel SIG_{Ksa}(m)$$

$$SIG_{Ksa}(m) = E_{Ksa}(H(m))$$



How to verify your Ubuntu download

NOTE: You will need to use a terminal app to verify an Ubuntu ISO image. These instructions assume basic knowledge of the command line, checking of SHA256 checksums and use of GnuPG.

Verifying your ISO helps insure the data integrity and authenticity of your download. The process is fairly straightforward, but it involves a number of steps. They are:

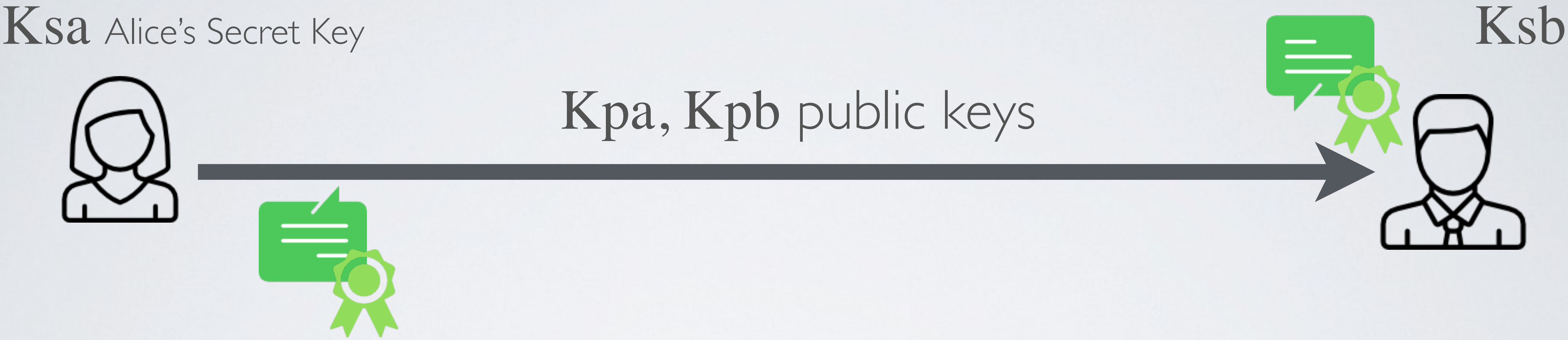
1. Download SHA256SUMS and SHA256SUMS.gpg files
2. Get the key used for the signature from the Ubuntu key server
3. Verify the signature
4. Check your Ubuntu ISO with sha256sum against the downloaded sums

After verifying the ISO file, you can then either install Ubuntu or run it live from your CD/DVD or USB drive.

Non-repudiation as a special case of integrity

	MAC	Digital Signature
Integrity	✓	✓
Non-repudiation	✗	✓

Digital Signatures and Confidentiality



Ksa Alice's Secret Key

Ksb

Kpa, Kpb public keys

1. Alice generates an asymmetric session key k
2. Use both symmetric and asymmetric cryptography to **encrypt, sign and verify** the message and the key

$$E_{Kpb}(k) \,\|\, E_{k}(m \,\|\, E_{Ksa}(H(m)))$$