

# Asymmetric Encryption and Digital Signatures

Thierry Sans

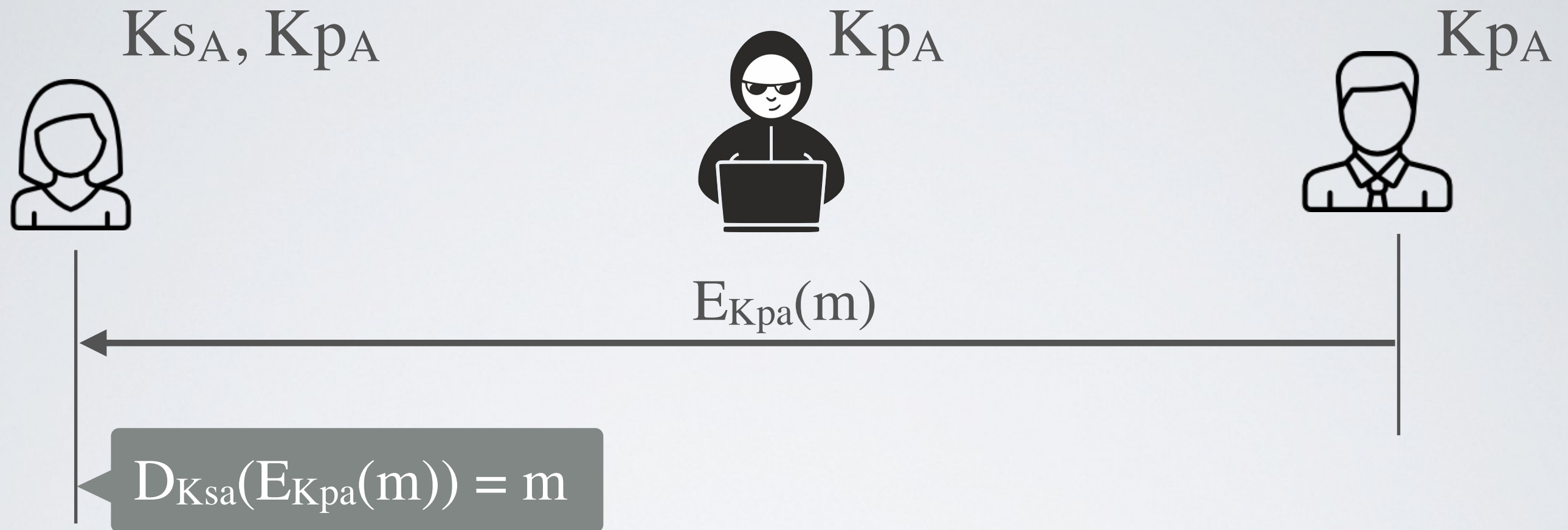
# Asymmetric keys



Alice generates a pair of asymmetric keys

- $K_{s_A}$  is the secret key that Alice keeps for herself
  - $K_{p_A}$  is the public key that Alice gives to everyone (even Mallory)
- ➔ These two keys  $K_{s_A}$  and  $K_{p_A}$  work together

# Asymmetric encryption for **confidentiality**

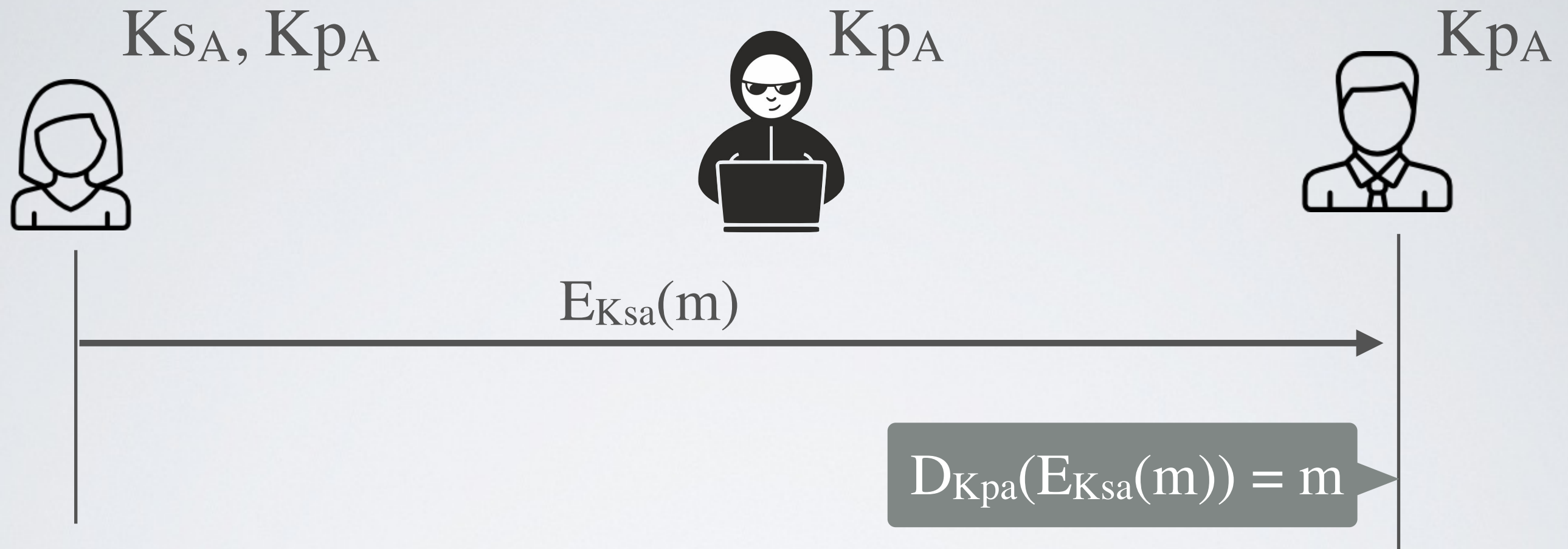


Bob encrypts a message  $m$  with Alice's public key  $K_{PA}$

➔ Nobody can decrypt  $m$ , except Alice with her private key  $K_{SA}$

✓ Confidentiality without the need to exchange a secret key

# Asymmetric encryption for **integrity**



Alice encrypts a message  $m$  with her private key  $K_{SA}$

➔ Everybody can decrypt  $m$  using Alice's public key  $K_{PA}$

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



# Functional Requirements

$D_{K_s}(E_{K_p}(m)) = m$  and  $D_{K_p}(E_{K_s}(m)) = m$  for every pair  $(K_p, K_s)$

- ✓ Generating a pair  $(K_p, K_s)$  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  $K_s$  for a given  $K_p$  is hard (exponential)
- Decryption without knowing the corresponding key is hard (exponential)

RSA

# RSA - Rivest, Shamir and Alderman

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

# Number Theory - Prime numbers

## Prime Numbers

- $p$  is prime if 1 and  $p$  are its only divisors e.g 3, 5, 7, 11 ...
  - $p$  and  $q$  are relatively prime (a.k.a. coprime) if  $\gcd(p,q) = 1$   
e.g  $\gcd(4,5) = 1$
- ➡ There are infinitely many primes

## Euler-Fermat Theorem

If  $n = p \cdot q$  and  $z = (p-1) \cdot (q-1)$

and  $a$  such that  $a$  and  $n$  are relative primes

Then  $a^z \equiv 1 \pmod{n}$



# 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  $n$  find  $p$  and  $q$  such that  $n = p \cdot q$

# RSA - generating the key pair

1. Pick  $p$  and  $q$  two large prime numbers and calculate  $n = p \cdot q$   
(see primality tests)
2. Compute  $z = (p-1) \cdot (q-1)$
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  $e * d = 1 \pmod{z}$  to find  $d$   
➔  $(d, n)$  is the **private key**  
however  $p$  and  $q$  must be kept secret too

# RSA - encryption and decryption

Given  $K_p = (e, n)$  and  $K_s = (d, n)$

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

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

➡  **$(m^e)^d \bmod n = (m^d)^e \bmod n = m$**

# The security of RSA

**RSA Labs Challenge** : factoring primes set

Key length	Year	Time
140	1999	1 month
155	1999	4 months
160	2003	20 days
200	2005	18 months
768	2009	3 years

Challenges are no longer active



# Key length and Key n-bit security

- RSA has very long keys, 1024, 2048 and 4096 are common
- Is it more secure than asymmetric crypto with key lengths of 56, 128, 192, 256 ?

➔ Key lengths **do not compare !**

RSA Key length	Effective key length
1,024	80
2,048	112
3,072	128
7,680	192
15,360	256

# Other asymmetric cryptography schemes

## **Diffie-Hellman** (precursor)

- ➡ No Authentication but good for key-exchange

## **El-Gamal**

- ➡ Good properties for homomorphic encryption

## **Elliptic Curve Cryptography** (trending 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

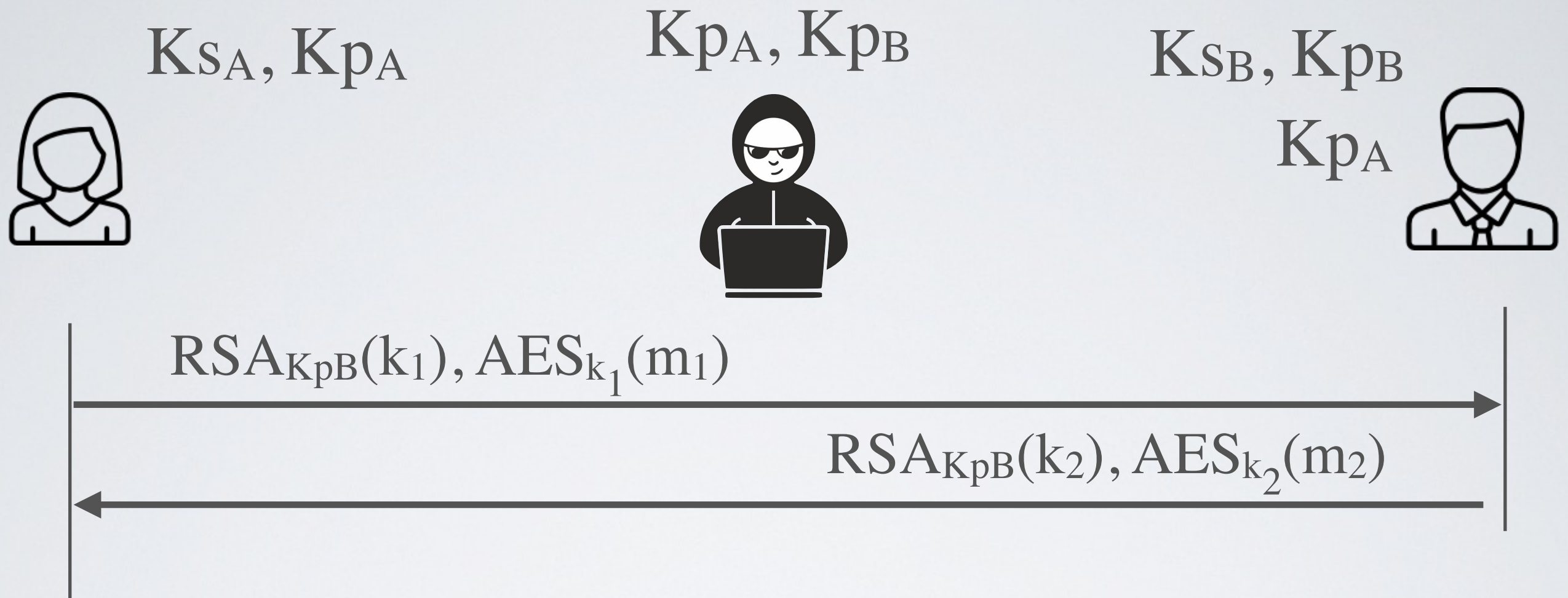
The best of both worlds

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

$$E_{K_p}(m) = \text{RSA}_{K_p}(k), \text{AES}_k(m)$$

Naive  
approach

But not perfect yet



- ✓ Does ensure the confidentiality of the communication
- Does not authenticate Alice or Bob



# Digital Signatures

# 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

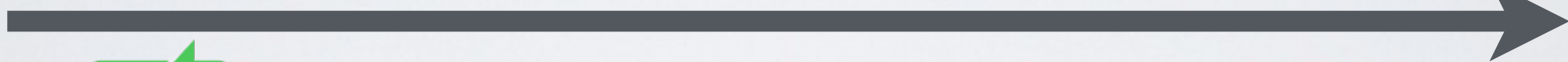
After verifying the ISO file, you can then either install Ubuntu or run it live from your CD/DVD or USB drive.

# Digital Signature

$K_{sa}$  Alice's Secret Key



$K_{pa}, K_{pb}$  public keys



$K_{sb}$







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

$$m \parallel \text{SIG}_{K_{sa}}(m)$$

$$\text{SIG}_{K_{sa}}(m) = E_{K_{sa}}(H(m))$$

# Non-repudiation as a special case of integrity

	MAC	Digital Signature
Integrity		
Non-repudiation		

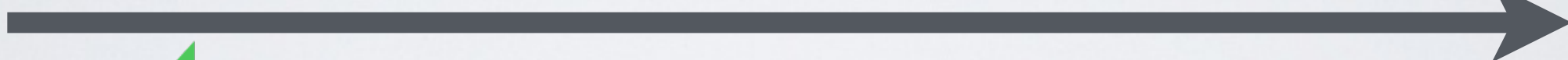


# Digital Signatures and Confidentiality

$K_{sa}$  Alice's Secret Key



$K_{pa}, K_{pb}$  public keys



$K_{sb}$

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

$$E_{K_{pb}}(k) \parallel E_k(m \parallel E_{K_{sa}}(H(m)))$$

# This how GPG works



As of versions 2.0.26 and 1.4.18, GnuPG supports the following algorithms:

- Pubkey: RSA, ElGamal, DSA
- Cipher: IDEA (since versions 1.4.13 and 2.0.20), 3DES, CAST5, Blowfish, AES-128, AES-192, AES-256, Twofish, Camellia-128, -192 and -256 (since versions 1.4.10 and 2.0.12)
- Hash: MD5, SHA-1, RIPEMD-160, SHA-256, SHA-384, SHA-512, SHA-224
- Compression: Uncompressed, ZIP, ZLIB, BZIP2

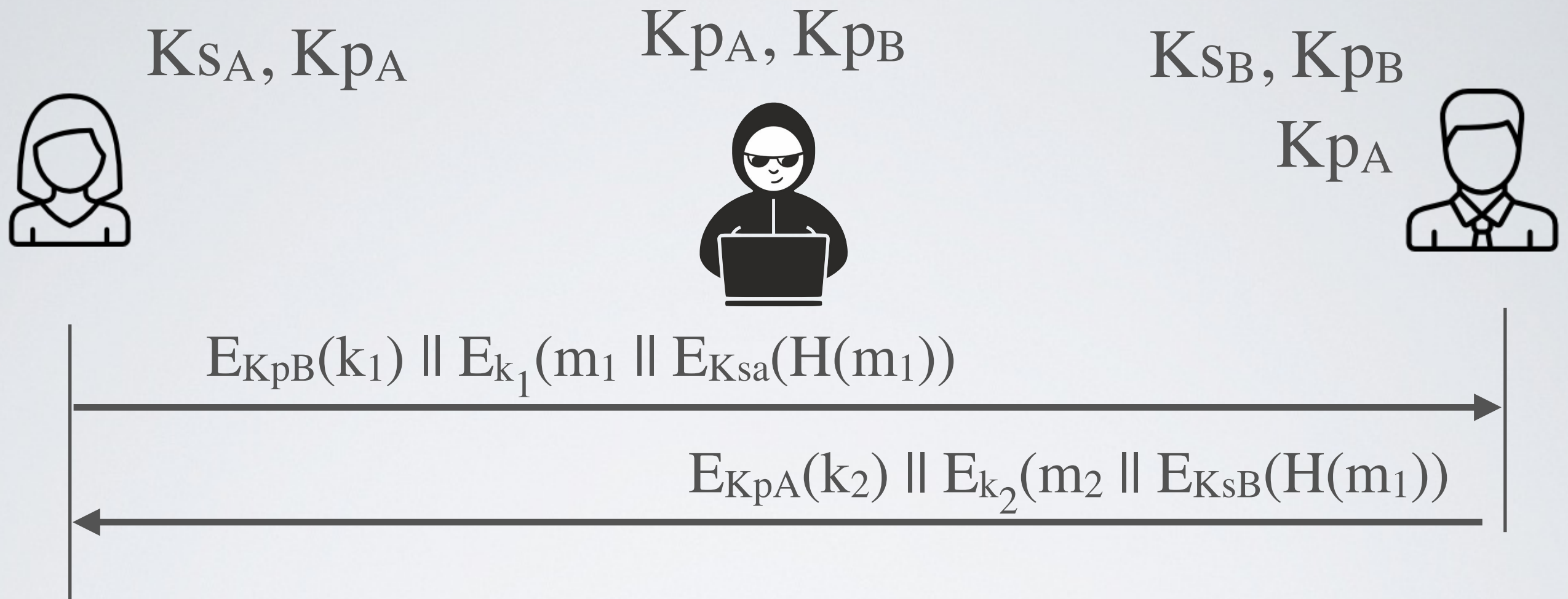
More recent releases of GnuPG 2.x ("stable" and "modern" series) expose most cryptographic functions and algorithms [Libgcrypt](#) (its cryptographic library) provides, including support for [elliptic curve cryptography](#) (ECDSA, ECDH and EdDSA)<sup>[10]</sup> in the "modern" series (i.e. since GnuPG 2.1).

# Asymmetric encryption for key exchange

Should we use asymmetric encryption for key exchange?

- ✓ Simple solution for non-interactive protocol (e.g GPG)
- But not a good solution for interactive protocols

# Not a good solution for key exchange



- ✓ Does ensure the confidentiality of the communication
- ✓ Does authenticate Alice and Bob
- Does not prevent replay attacks
- Does not ensure Perfect Forward Secrecy
- Does not ensure the authenticity of the public keys