ELTE EÖTVÖS LORÁND UNIVERSITY

CRYPTOGRAPHY AND SECURITY

Practice

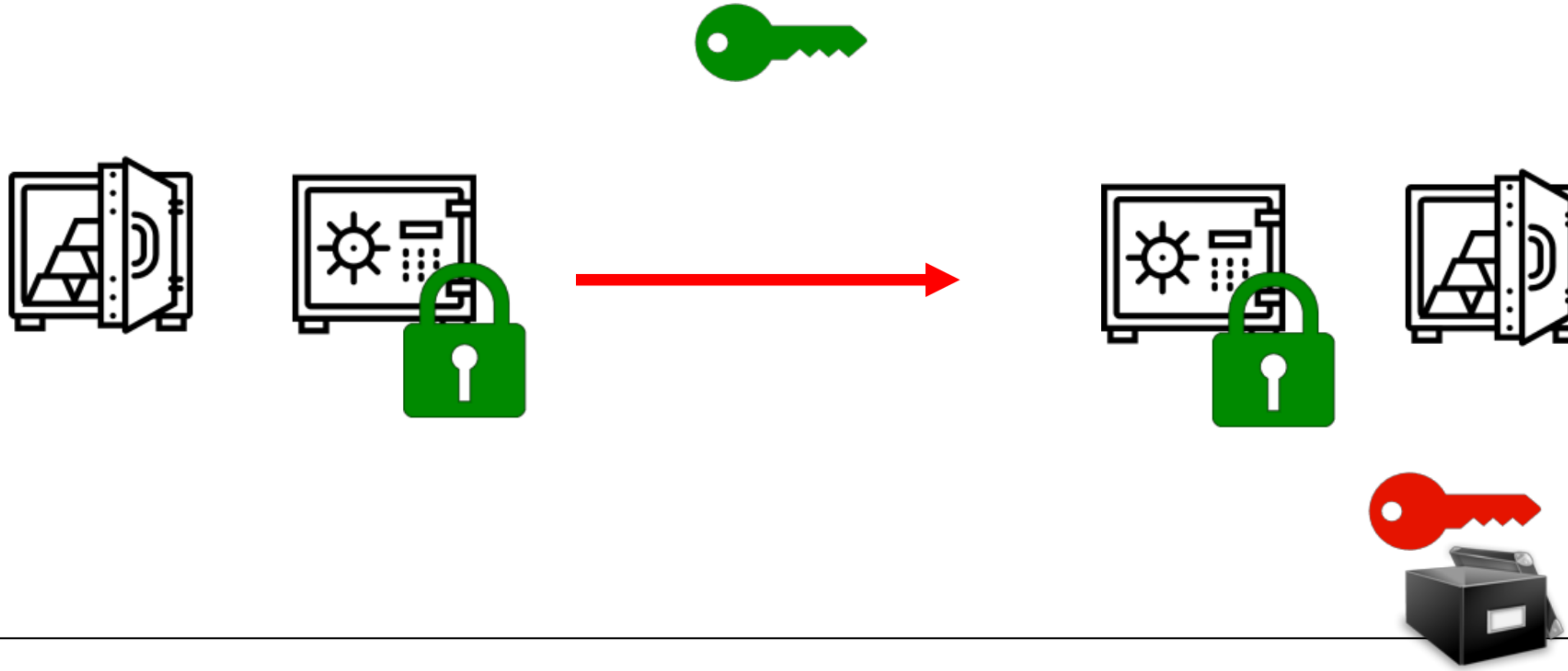
IP-18FKVKRBG

Lecture 9

2022-2023-2

Asymmetric Cryptography

Asymmetric Ciphers



Asymmetric (Public-Key) Cryptography

In key generation, a key pair is generated for each of the members of the system.

Key

Public key

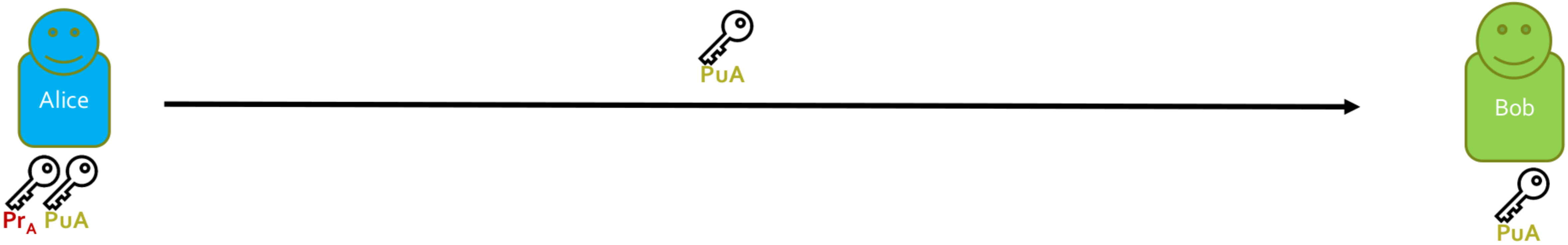
Private key

(Encrypt) ---------------------> (Decrypt)

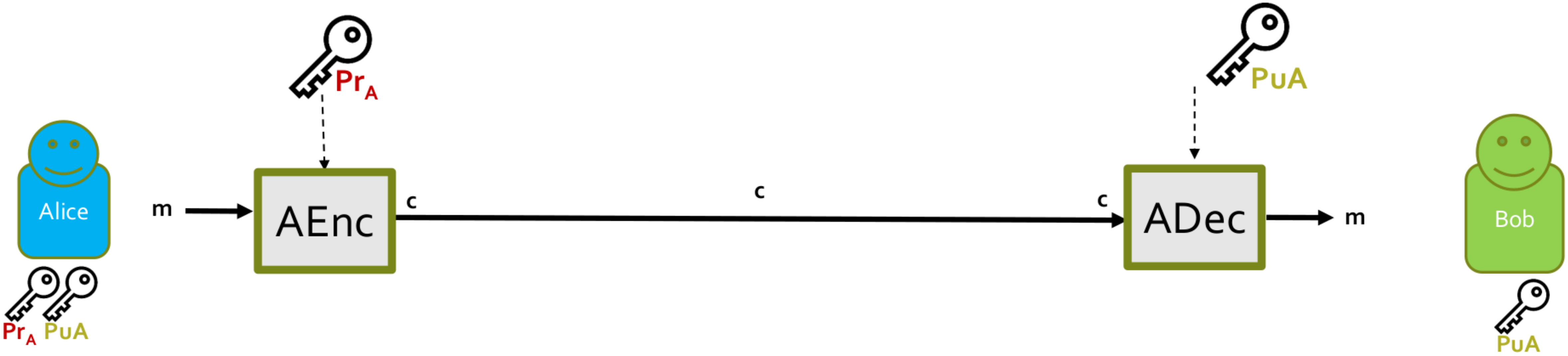
(Decrypt) <--------------------- (Encrypt)

A text Can NOT be encrypted and then decrypted by the SAME key

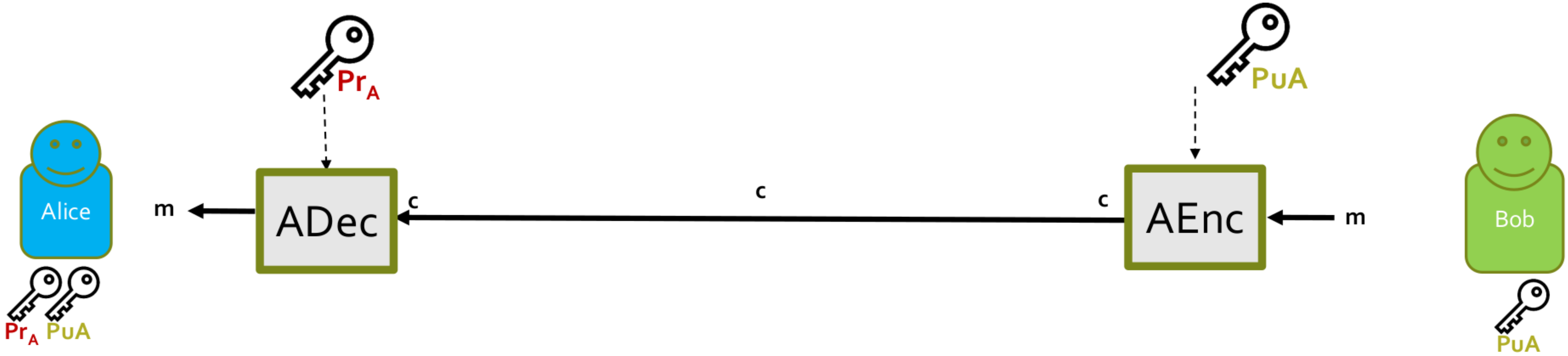
Public-Key Encryption: Sharing public key



Public-Key Encryption: Sender Authentication



Public-Key Encryption: Confidentiality



RSA

RSA: Key generation

- Choose two unequal large prime numbers p and q
- Compute modulo $n = p.q$
- Compute totient function $\varphi(n) = \varphi(p.q) = (p-1)(q-1)$

- Generate numbers e, d such that
  - 1< e < $\varphi(n)$ and e relatively prime to $\varphi(n)$
  - Find d such that ed $\equiv 1 \pmod{\varphi(n)}$ → it only exist if gcd(e, $\varphi(n)$) = 1
  - $x^{ed} \equiv x \bmod n$

RSA: "schoolbook" Encryption and Decryption

- Public encryption key $\langle n, e \rangle$
  - $\mathrm{Encrypt}(\langle n, e \rangle, x) = x^e \bmod n$
- Private decryption key $\langle n, d \rangle$
  - $\mathrm{Decrypt}(\langle n, d \rangle, y) = y^d \bmod n = x^{ed} \bmod n = x$

RSA: Example

Preparation (Bob Side)

- Bob chooses p = 41, q = 61, → and gets n = 2501, $\varphi(n) = 2400$
- He chooses e = 23, → and gets d = 2087
  - By choosing other values of e we get other values of d.
- Alice receives n and e, from Bob

Alice (Sender) Side

- Plaintext = **100**
- Encryption: $100^{23} \bmod 2501 =$ **2306**

Bob (Receiver) Side

- Decryption: $2306^{2087} \bmod 2501 =$ **100**

RSA Coding: Necessary functions

```
from sympy import randprime
from math import gcd

def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, x, y = egcd(b % a, a)
        return (g, y - (b // a) * x, x)

def modinv(b, n):
    g, x, _ = egcd(b, n)
    if g == 1:
        return x % n
```

RSA Coding: Generate two prime numbers

It makes sure that:

- both generated prime numbers are not the same, and
- their product does not exceed the final number of bits (key size)

```
while prime_1 == prime_2 or (prime_1 * prime_2) > 2**key_size:
    prime_1 = randprime(3, 2**key_size/2)
    prime_2 = randprime(3, 2**key_size/2)
```

Note: For in lecture testing choose a small key size.

RSA Coding: Generating the public parameter

```python
def generate_public_exponents(totient):
    public_exponent = 0
    for e in range(3, totient-1):
        if gcd(e, totient) == 1:
            public_exponent = e
            break
    return public_exponent
```

RSA Coding: Generating the e and d parameters

```
public_exponent = generate_public_exponents(totient)

private_exponent = modinv(public_exponent, totient)
```

RSA Coding: Encryption and decryption

```
def rsa_encrypt(plain_text, rsa_modulus, public_exponent):
    cipher = ''.join(chr((ord(ch)**public_exponent) % rsa_modulus)
            for ch in plain_text)
    return cipher.encode().hex()

def rsa_decrypt(cipher_text, rsa_modulus, private_exponent):
    return ''.join(chr((ord(ch)**private_exponent) % rsa_modulus)
                   for ch in bytearray.fromhex(cipher_text).decode())
```