Block Cipher

Elements of Applied Data Security M

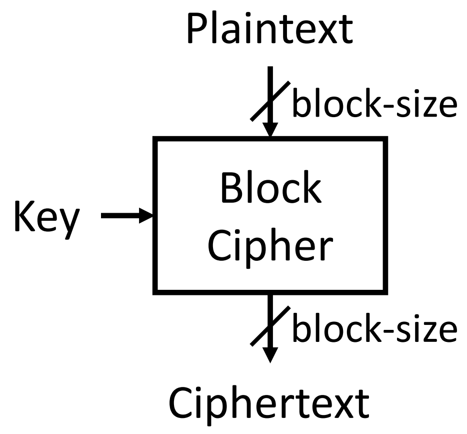
Alex Marchioni– alex.marchioni@unibo.it

Filippo Martinini– filippo.martinini@unibo.it

Andriy Enttsel– andriy.enttsel@unibo.it

Block Ciphers

Unlike Stream Ciphers, which encrypt one bit at a time, Block Ciphers encrypt a **block of text**. For example, AES encrypts 128 bit blocks.



Since a block cipher is suitable only for the encryption of a single block under a fixed key, a multitude of **modes of operation** have been designed to allow their repeated use in a secure way.

Moreover, block ciphers may also feature as **building blocks** in other cryptographic protocols, such as universal hash functions and pseudo-random number generators.

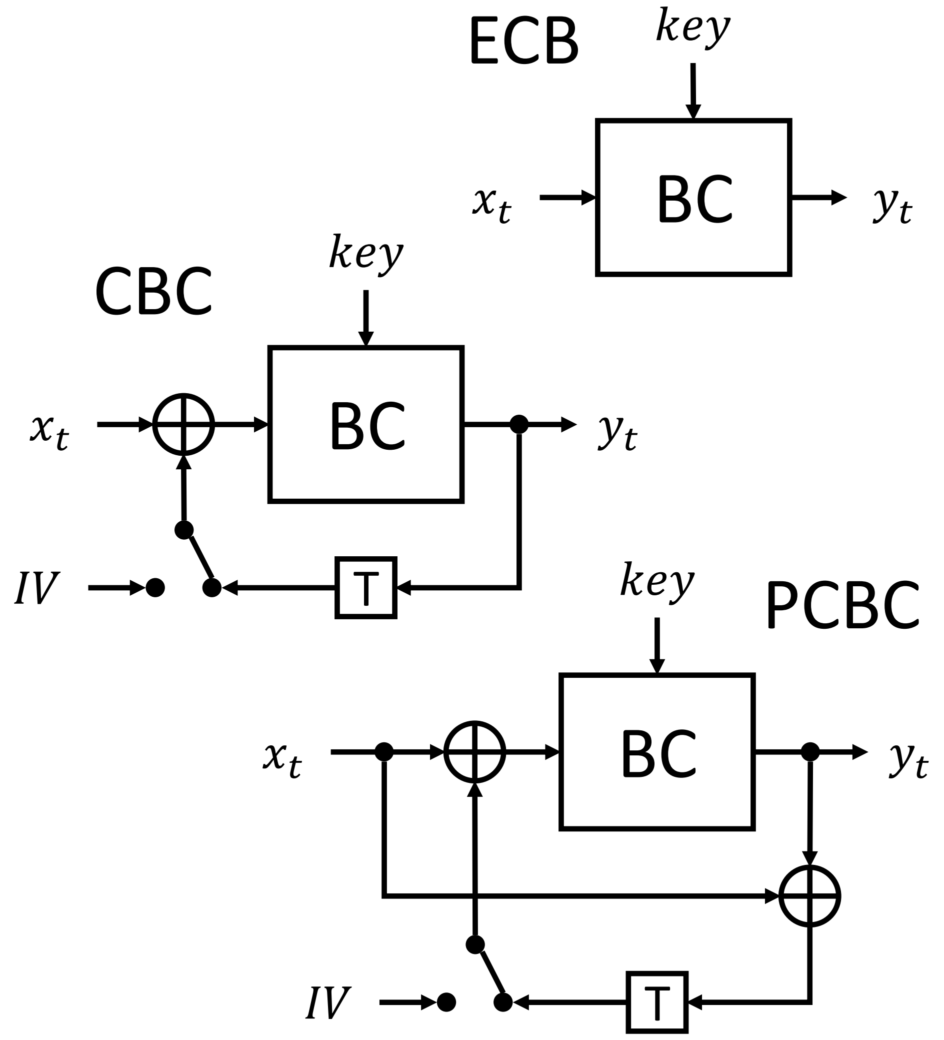
Block Ciphers

Most block cipher algorithms are obtained by iterating an invertible transformation. Each iteration is called **round** and the repeated transformation is known as **round function**.

Usually, the round function takes different **round keys**, which are derived by expanding the original key.

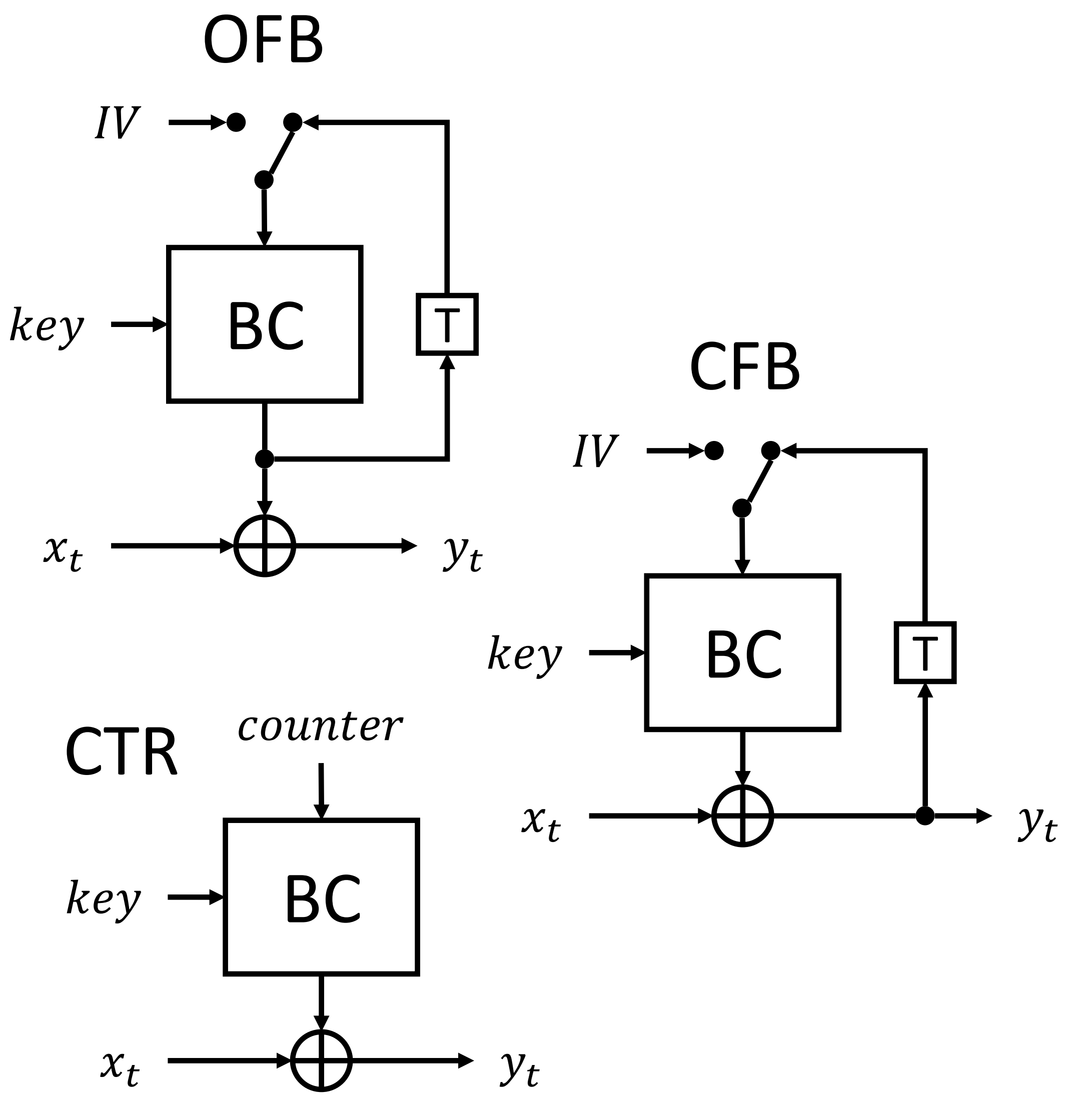
Modes of Operation

- **Electronic codebook (ECB)**: the simplest of the encryption modes where the message is divided into blocks, and each block is encrypted separately.
- **Cipher block chaining (CBC)**: each block of plaintext is XORed with the previous ciphertext block before being encrypted.
- **Propagating cipher block chaining (PCBC)**: each block of plaintext is XORed with both the previous plaintext block and the previous ciphertext block before being encrypted.



Modes of Operation

- **Output feedback (OFB)**: it generates keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext.
- **Cipher feedback (CFB)**: similar to OFB, but to generate the new keystream block, it employs the previous ciphertext instead of the previous keystream block .
- **Counter (CTR)**: the keystream block is generated from the value of a counter that is incremented at each new block.



Task 1: AES

Advanced Encryption Standard (AES)

AES is a specification for the symmetric-key encryption established by the NIST in 2001 and then adopted by the U.S. government.

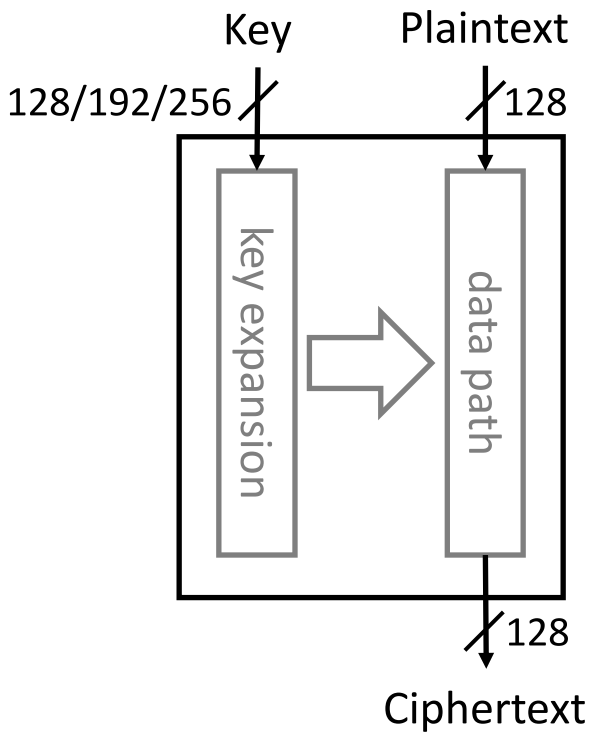
The standard comprises three block ciphers from a larger collection originally published as **Rijndael.** Each of these ciphers has a 128-bit block size, with key sizes of 128, 192 and 256 bits.

[1] FIPS PUB 197, Advanced Encryption Standard (AES), National Institute of Standards and Technology, U.S. Department of Commerce, November 2001.

[2] Joan Daemen and Vincent Rijmen, The Design of Rijndael, AES - The Advanced Encryption Standard, Springer-Verlag 2002 (238 pp.)

The Rijndael Block Cipher

Rijndael is designed to resist against all known attacks and to be fast and compact when implemented in most platforms.

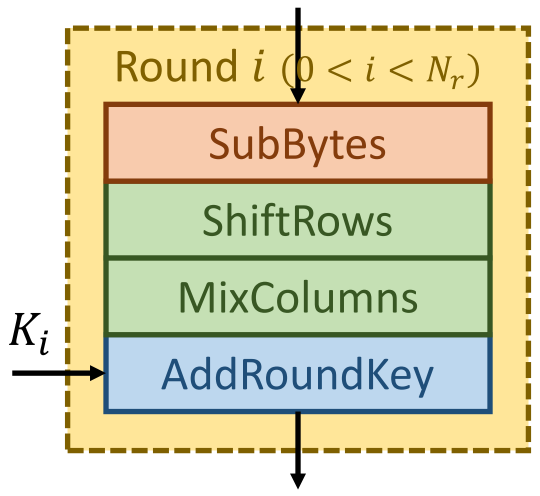


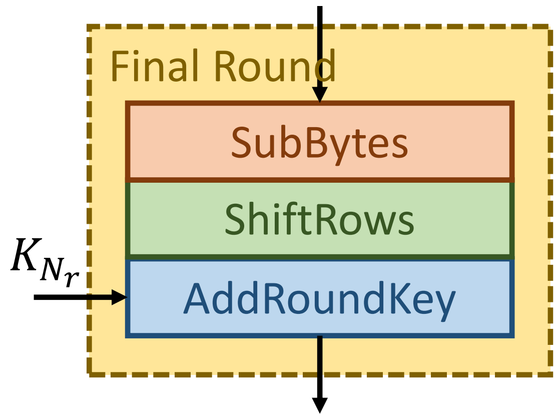
Rijndael is composed of a **key expansion** block and a **data path** that can be viewed as an iterated block cipher, where each iteration is called **round**. The number of rounds depends on the block (for AES fixed to 128bit) and key length.

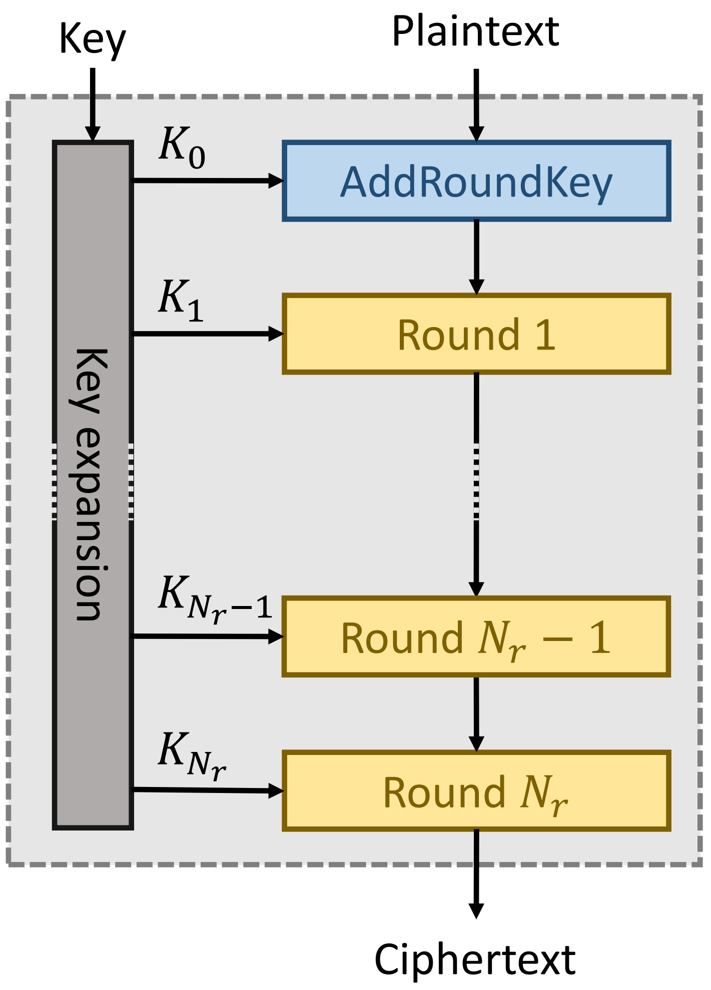
key length	128	192	256
# rounds N_r	10	12	14

AES

A brief illustration of AES.







Python Packages for Cryptography

Pycryptodome: self-contained Python package of low-level cryptographic primitives. It is a fork of PyCrypto that has been enhanced to add more implementations and fixes to the original library.

PyNaCl: Python binding to libsodium, which is a fork of the Networking and Cryptography library. These libraries have a stated goal of improving usability, security and speed.

Cryptography: cryptography is a package which provides cryptographic recipes and primitives to Python developers. It includes both high level recipes and low level interfaces to common cryptographic algorithms such as symmetric ciphers, message digests, and key derivation functions.

Pycryptodome

PyCryptodome is a self-contained Python package of low-level cryptographic primitives. It is organized in sub-packages dedicated to solving a specific class of problems:

- `Cryptodome.Cipher`: Modules for protecting **confidentiality** that is, for encrypting and decrypting data (example: AES).
- `Cryptodome.Signature`: Modules for assuring **authenticity**, that is, for creating and verifying digital signatures of messages (example: PKCS#1)
- `Cryptodome.Hash`: Modules for creating cryptographic **digests** (example: SHA-256).
- `Cryptodome.PublicKey`: Modules for generating, exporting or importing public keys (example: RSA or ECC).

`Cryptodome.Cipher` subpackage

The base API of a cipher is fairly simple:

- You instantiate a cipher object by calling the `new()` function from the relevant cipher module. The first parameter is always the cryptographic **key**. You can (and sometimes must) pass additional cipher- or mode-specific parameters such as a nonce or a mode of operation.

```python
from Cryptodome.Cipher import AES

key = b'0123456701234567'
cipher = AES.new(key, AES.MODE_ECB)
```

`Cryptodome.Cipher` subpackage

- For encrypting data, you call the `encrypt()` method of the cipher object with the plaintext. The method returns the piece of ciphertext. For most algorithms, you may call encrypt() multiple times (i.e. once for each piece of plaintext).
- For decrypting data, you call the `decrypt()` method of the cipher object with the ciphertext. The method returns the piece of plaintext.

```
plaintextA = b'this is a secret'
ciphertext = cipher.encrypt(plaintextA) # b'\x8dk\x84\xcey*h\xach\x9b\xd0[\xb6pR\x95'
plaintextB = cipher.decrypt(ciphertext) # b'this is a secret'
```

Task 1 – AES

- Import the AES class from Cryptodome package.
- Create an instance of AES for each of the following operation modes:
  - Electronic Code Book (ECB)
  - Cipher Block Chaining (CBC)
  - Cipher FeedBack (CFB)
  - Counter (CTR)
- For each instance of AES, encrypt and decrypt the given image and comment your results.

Task 2: RSA

RSA

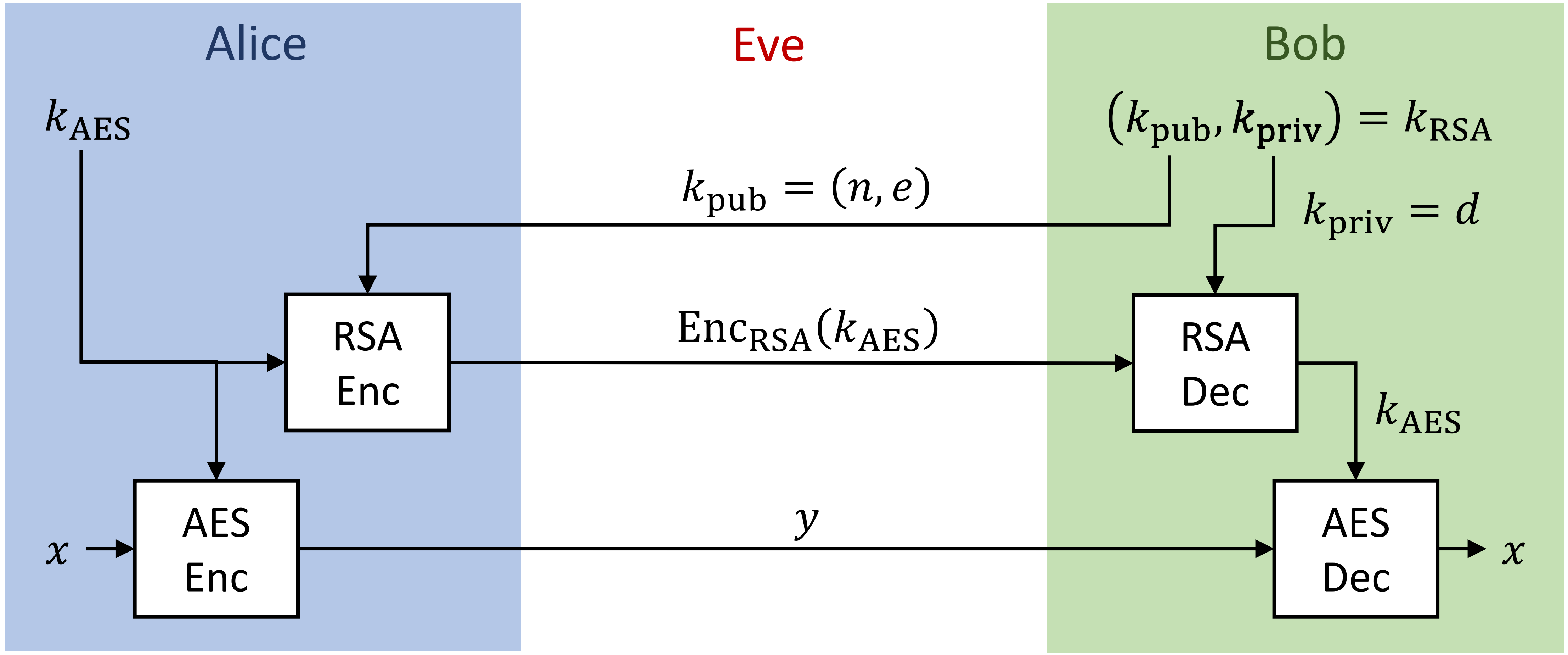
To establish a secure symmetric communication channel, it is necessary for Alice and Bob to share a common key.

The key exchange is usually carried out using an asymmetric algorithm.

You are asked to implement:

1. The RSA algorithm (a class RSA) to exchange a key.
2. The symmetric block cipher AES (using Cryptodome) that takes the shared key to encrypt and decrypt a message.

RSA and AES



Alice
Eve
Bob
k_{AES}
$(k_{\text{pub}}, k_{\text{priv}}) = k_{\text{RSA}}$
$k_{\text{pub}} = (n, e)$
$k_{\text{priv}} = d$
RSA Enc
$\text{Enc}_{\text{RSA}}(k_{\text{AES}})$
RSA Dec
k_{AES}
x
AES Enc
y
AES Dec
x

RSA – Key Generation

Key is generated through 5 steps:

1. Choose two prime numbers (p, q)
2. Compute $n = pq$
3. Compute $m = \phi(n) = (p-1)(q-1)$
4. Select $e \in \mathbb{Z}_n \text{ s.t. } \gcd(e, m) = 1$
5. Compute $d \text{ s.t. } de \equiv 1 \bmod m$

Steps (4) and (5) require the Extended Euclidean Algorithm (EEA).

Extended Euclidean Algorithm

It computes the inverse of a number a with respect to multiplication modulo m, when the inverse exists. Otherwise, it raises an error.

- **Input**:
  - Number $a \in \mathbb{N}_m$
  - Modulo m
- **Outputs**:
  - The inverse of a wrt multiplication modular m, $s = a^{-1} \bmod m$

```
**Input** $a, m$
$r_0, r_1 \leftarrow m, a$
$s_0, s_1, t_0, t_1 \leftarrow 0, 1, 1, 0$
$i \leftarrow 1$
**while** $\mathrm{r}_i \neq 0$
    $i \leftarrow i + 1$
    $r_i \leftarrow r_{i-2} \bmod r_{i-1}$
    $q_i \leftarrow (r_{i-2} - r_i) / r_{i-1}$
    $s_i \leftarrow s_{i-2} - q_i s_{i-1}$
    $t_i \leftarrow t_{i-2} - q_i t_{i-1}$
**endwhile**
**If** $r_{i-1} \neq 1$ **then**
    **raise Error**
**endif**
**Output** $s_{i-1}$
```

RSA encryption and decryption

Encryption:

$$y = x^e \bmod n$$

Decryption:

$$x = y^d \bmod n$$

RSA implementation issues:

- Exponentiation of large numbers
- Generation of large prime numbers

RSA – issue with the exponentiation

When you deal with very high numbers (a, b), (e.g., $a, b \in \mathbb{Z}_{2048}$), it is not trivial to compute a^b as it may requires an unworkable amount of time.

To solve this problems many algorithms for fast and efficient exponentiation have been studied.

Square-and-Multiply is the base for the most used algorithms.

Square-and-Multiply

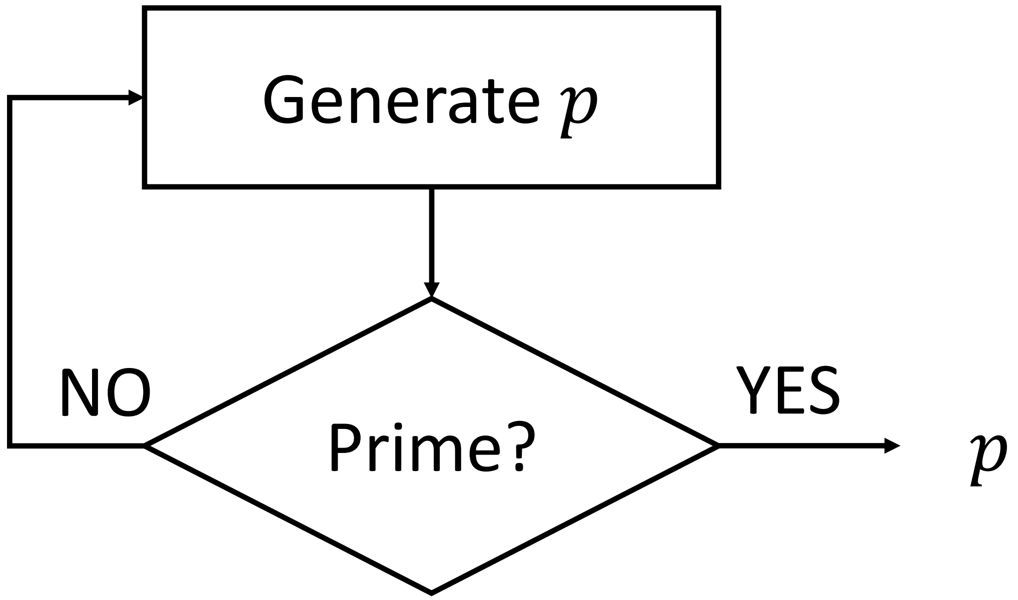
Computes the exponentiation $x^e \bmod n$ by means of squaring and multiplication.

- **Input**:
  - base x
  - Exponent e with binary representation $0be_{L-1}e_{L-2}\cdots e_1 e_0$
  - Modulo n
- **Outputs**:
  - Exponentiation result $y = x^e \bmod n$

```
**Input** $x, e = 0be_{L-1}e_{L-2}\cdots e_1 e_0, n$
$L_{\max} = \max_i\{e_i = 1\}$
$y \leftarrow 1$
**For** $i = L_{\max}, L_{\max} - 1, \dots, 1, 0$
    $y \leftarrow y^2 \bmod n$
    **If** $e_i = 1$ **then**
        $y \leftarrow y \cdot x \bmod n$
    **endif**
**endfor**
**Output** $y$
```

RSA – issue with the primes

How do we find two prime numbers (that are usually very large)?
We can run a TEST (e.g., Miller-Rabin test):



Miller-Rabin Primality Test

Determines whether a given number is likely to be prime or surely composite

- **Input**:
  - Candidate odd prime number p
  - Number of trials N
- **Outputs**:
  - Whether p is probably prime (True) or p is surely composite (False)

```
**Input** $p = q \cdot 2^r + 1, N$
**For** $i = 0, 1, \dots, N-1$
    **draw** $x \in \{2, 3, \dots, p-2\}$
    $y \leftarrow x^q \bmod p$
    **If** $(y = 1$ **or** $y = p - 1)$ **then**
        **continue**
    **endif**
    **For** $j = 0, 1, \dots, r-1$
        $y \leftarrow y^2 \bmod p$
        **If** $y = p - 1$ **then**
            **continue** (main loop)
        **endif**
    **endfor**
    test $\leftarrow 1$
**endfor**
test $\leftarrow 0$
**Output** test
```

Task 1

Implementation of the RSA public-key cryptosystem.

- Key generation
  - Generation of two big prime numbers p and q for the factorization of n. (Usage of the Miller-Rabin for primality test).
  - Choice of e and computation of d. (Usage of the Extended Euclidean algorithm)
- Encryption and decryption
  - Exponentiation of plain/ciphertext (Usage of Square and Multiply to perform exponentiation)

RSA – Debug (1)

- $(p, q) = (59, 97)$
- $n = p \cdot q = 5723$
- $m = \phi(n) = (p - 1)(q - 1) = 5568$
- $e = 5$, note that $\gcd(e, m) = 1$
- $d = \mathrm{EEA}(e) = 3341$
- $x = 8$
- $y = Enc_{5723,5}(8) = 8^5 \bmod 5723 = 4153$
- $x = \mathrm{Dec}_{3341}(4153) = 4153^{3341} \bmod 5723 = 8$

RSA – Debug (2)

p	`1 083e 9356 4892 2e73`
q	`1 cc1a 881e 3682 1695`
n	`1 daeb d39a 6f08 9f23 b7b1 64f6 a578 eaef`
m	`1 daeb d39a 6f08 9f20 e358 4982 2664 a5e8`
e	`0 0000 0000 0000 0000 0000 0000 0000 0003`
d	`1 3c9d 37bc 4a05 bf6b 423a dbac 1998 6e9b`
x	`3031 3233 3435 3637 3031 3233 3435 3637`
y	`1 2366 ac62 e7b4 6d1a 1f67 6a2b 97d9 65d2`

Task 2

Implementation of a secure communication channel:

- Create two instances of RSA and make them share a key $k_{\mathrm{AES}} \in \mathbb{Z}_{128}$.
- Create two instances of AES sharing the same key and use them to encrypt/decrypt a message.

Key length

- AES works with keys that have a fixed length, e.g., 128. For this reason, Bob must exchange an $k_{\text{AES}} \in \{0,1\}^{128}$ with Alice.
- To preserve the AES key, RSA must work with $n > x$, so it is convenient to pick only $p, q \in [2^{63}, 2^{64})$, such that: $n = p \cdot q > x$.
- RSA implementation must rely on Python integers.
  Python, unlike NumPy, supports arbitrary large integers, which means that integers can be of any size, limited only by the amount of memory available.

Deadline

23/05/2023 at 12:00 pm (noon)

Calendar

5th May	Presentation and work on Assignment 3
12th May	• Presentation of the projects • Presentation of Assignment 3 bonus Task • Working on Assignment 3
19th May	Working on Assignment 3
23rd May	• Deadline of Assignment 3 • Discussion of the solutions of all Assignments (in preparation of the lab exam)
30th May	Lab exam (assignments discussion)
6th June	Question time (about theory)