RSA

Elements of Applied Data Security M

Alex Marchioni– alex.marchioni@unibo.it

Livia Manovi– livia.manovi@unibo.it

RSA

- RSA (Rivest-Shamir-Adleman, 1977) is a widely used asymmetric algorithm for secure communication
- The algorithm requires a pair of keys:
  - a **public key** k_{pub} used for encryption
  - a **private key** k_{priv} used for decryption.
- Based on **integer factorization**:
  - Plaintext x and ciphertext y are modeled as integers $x, y \in \mathbb{Z}_n$.
  - Security relies on the difficulty of factoring the product of two large prime numbers.

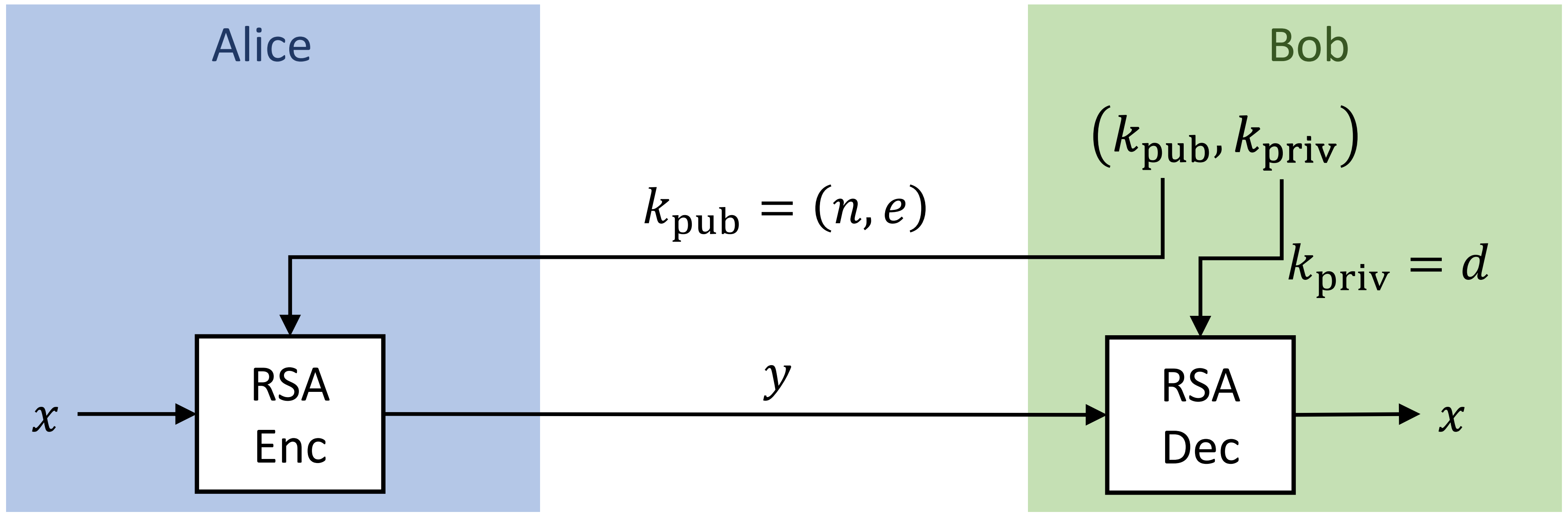
RSA – Encryption and Decryption

Encryption:

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

Decryption:

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


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

RSA – Key Generation

Key is generated through 5 steps:

1. Choose two prime numbers (p, q)
2. Compute $n = p \cdot q$
3. Compute $m = \phi(n) = (p-1)(q-1)$
4. Draw $e \in \mathbb{Z}_n$ such that $\gcd(e, m) = 1$
5. Compute d such that $d \cdot e \equiv 1 \pmod{m}$

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

RSA – Practical issues

- <u>Exponentiation involving large numbers</u>
  - When you deal with very high numbers (a, b), (e.g., $a, b \in \mathbb{Z}_{2^{2048}}$), it is not trivial to compute a^b as it may require an unworkable amount of time.
  - To solve this problem, many algorithms for fast and efficient exponentiation have been studied. **Square-and-Multiply** is the base for most of them.

- <u>Generation of large prime numbers</u>
  - Testing for primality is a much easier task than integer factorization. Therefore, one can randomly draw a big number and then test for primality.
  - A well-known algorithm for testing primality is the **Miller Rabin Test**

Python integers

- RSA implementation must rely on Python integers `int`. Python lets integers be arbitrarily large (the only limit is the amount of memory available)
  - integers of thousands of bits
  - no overflow

- Conversely, NumPy integers have fixed limits defined by the `dtype`. Therefore, NumPy integers are limited in size and suffer from overflow.

Tasks

1. RSA implementation
    - Extended Euclidean Algorithm
    - Square and Multiply
    - Miller Rabin test
    - RSA class
2. RSA + AES

Task 1: RSA

Extended Euclidean Algorithm (EEA)

It computes the greatest common divisor (gcd) of two integers a and m.

Assuming $m > a$, if the gcd is 1, EEA also computes the inverse of a number a with respect to multiplication modulo m.

- **Input**:
  - a (int)
  - m (int)
- **Outputs**:
  - $\mathrm{gcd}(a, m)$
  - $s, t \in \mathbb{Z}$ such that $\mathrm{gcd}(a, m) = \mathrm{s} \cdot a + t \cdot m$. If $\mathrm{gcd}(a, m) = 1$ then $s \equiv a^{-1} \mathrm{mod}\, m$

```
**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** $\mathrm{r}_i \neq 0$
    $i \leftarrow i + 1$
    $r_i \leftarrow r_{i-2} \setminus r_{i-1}$
    $q_i \leftarrow (r_{i-2} - r_i) / r_{i-1}$
    $s_i \leftarrow s_{i-2} - q_i s_{i-1}$
    $t_i \leftarrow t_{i-2} - q_i t_{i-1}$
**endwhile**
**Output** $r_{i-1}, s_{i-1}, t_{i-1}$
```

Square-and-Multiply

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

- **Input**:
  - base x (int)
  - exponent e (int)
  - modulo n (int)
- **Outputs**:
  - $y = x^e \backslash n$ (int)

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

Miller-Rabin Primality Test

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

- **Input**:
  - Candidate odd prime number p (int)
  - Number of trials N (int)
- **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 $x \in \{2, 3, \dots, p-2\}$
    $y \leftarrow x^q \setminus p$
    If $(y = 1$ or $y = p-1)$ then
        continue
    endif
    For $j = 0, 1, \dots, r-1$
        $y \leftarrow y^2 \setminus p$
        If $y = p - 1$ then
            continue (main loop)
        endif
    endfor
    test $\leftarrow 1$
endfor
test $\leftarrow 0$
Output test
```

RSA class

RSA class must support both encryption and decryption

- **Decryption**: user must specify the length of the key (`length`) so that k_{pub} and k_{priv} can be generated.
- **Encryption**: user must provide the $k_{\text{pub}} = (n, e)$ so that a message can be encrypted.

```
class RSA:
  ''' class docstring '''

  def __init__(self, length=None, n=None, e=None):
    ''' constructor docstring '‘’
    self.length = ...
    self.n = ...
    self.e = ...
    ...

  def encrypt(self, plaintext):
    ...
    return ciphertext

  def decrypt(self , ciphertext):
    ...
    return plaintext
```

Task 1

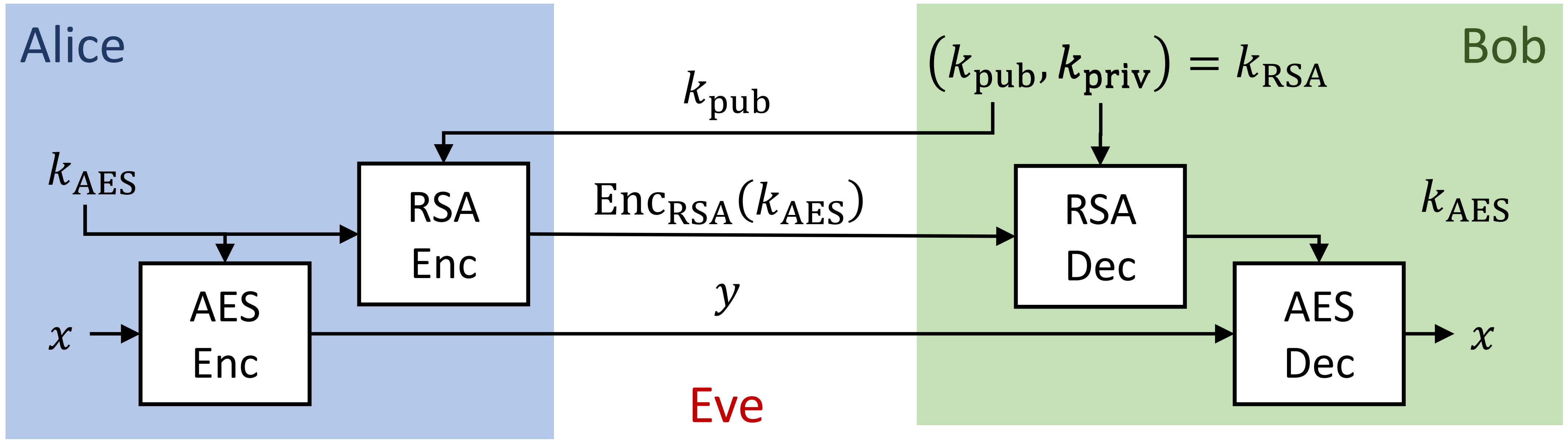
Implement and test:

- Extended Euclidean Algorithm
- Square-and-Multiply Algorithm
- Miller Rabin Test
- RSA class

Task 2: RSA + AES

RSA and AES

RSA is not suited to provide confidentiality in case of large messages. However, it can be exploited to establish a secure channel over which two entities (Alice and Bob) can exchange the key for a symmetric algorithm.



Task 2

Implementation of a secure communication channel:

- Create two instances of RSA and make them share a key $k_{\mathrm{AES}} \in \mathbb{Z}_{2^{256}}$.
- Create two instances of AES sharing the same key and use them to encrypt/decrypt the message in file `lorem_ipsum.txt`.

Bonus Task: Finding Large Primes

How common are large primes

RSA requires you to randomly draw two large prime numbers $p, q \sim 2^{L/2}$ (where L is the key length) by drawing a sufficiently large integer number and then testing for primality.

However, <u>as numbers get larger, the gaps between primes tend to increase</u>, making large primes less frequent. Moreover, predicting the exact location of large primes becomes increasingly difficult as numbers grow larger.

Knowing that prime numbers become less dense as their values increase,

is there a reasonable chance that a large random number is prime?

Bonus Task

- Determine the probability for a large random number to be prime.

$$\mathrm{Prob}\left\{p \text{ prime} | p \sim \mathcal{U}\left(\left\{2^{L/2}+1, 2^{L/2}+3, \ldots, 2^{L/2+1}-1\right\}\right)\right\}$$

Given a key length $L \in \{512, 1024, 2048\}$, estimate the likelihood for a number p to be prime when p is uniformly drawn from the set of odd numbers between $2^{L/2}$ and $2^{L/2+1}$.

Deadline

Tuesday, May 21st at 12PM (noon)