南京邮电大学

# 《密码学》实验报告（四）

（ 2024 / 2025 学年 第 二 学期）

## 题　　目：公钥加密算法实现

| | |
|---|---|
| 专　　业 | 信息安全 |
| 学 号 姓 名 | B230410 |
| | B23041011 |
| | 于明宏 |
| 指 导 教 师 | 李琦 |
| 指 导 单 位 | 计算机学院、软件学院、网络空间安全学院 |
| 日　　期 | 2025.5.16 |

# 公钥加密算法实现

## 一、课题内容和要求

本实验的目标是实现 RSA 算法。

## 二、实现分析

RSA 算法先计算出密钥对，之后发送方对明文用公钥加密，接收方对密文用私钥解密。

## 三、概要设计

采用 Python 语言编写，完整实现 RSA 算法，对过长的数据使用分组加密，采用 PKCS#1 v1.5 填充。

## 四、源程序代码

```
import random
from typing import Tuple

class RSA:
    def __init__(self, key_size: int = 2048):
        self.key_size = key_size
        self.public_key, self.private_key = self._generate_keys()
        self.block_size = (key_size // 8) - 11

    @staticmethod
    def _is_prime(n: int, k: int = 5) -> bool:
        if n <= 1:
            return False
        elif n <= 3:
            return True
        elif n % 2 == 0:
            return False
        d = n - 1
        s = 0
        while d % 2 == 0:
            d //= 2
            s += 1
        for _ in range(k):
            a = random.randint(2, n - 2)
            x = pow(a, d, n)
            if x == 1 or x == n - 1:
                continue
            for __ in range(s - 1):
                x = pow(x, 2, n)
```

```
                if x == n - 1:
                    break
            else:
                return False
        return True

    @staticmethod
    def _generate_prime(bits: int) -> int:
        while True:
            p = random.getrandbits(bits)
            p |= (1 << bits - 1) | 1
            if RSA._is_prime(p):
                return p

    @staticmethod
    def _extended_gcd(a: int, b: int) -> Tuple[int, int, int]:
        if a == 0:
            return b, 0, 1
        else:
            g, y, x = RSA._extended_gcd(b % a, a)
            return g, x - (b // a) * y, y

    @staticmethod
    def _modinv(a: int, m: int) -> int:
        g, x, y = RSA._extended_gcd(a, m)
        if g != 1:
            raise ValueError('模逆元不存在。')
        else:
            return x % m

    def _generate_keys(self) -> Tuple[Tuple[int, int], Tuple[int, int]]:
        p = self._generate_prime(self.key_size // 2)
        q = self._generate_prime(self.key_size // 2)
        while p == q:
            q = self._generate_prime(self.key_size // 2)
        n = p * q
        phi = (p - 1) * (q - 1)
        e = 65537
        while True:
            try:
                d = self._modinv(e, phi)
                break
            except ValueError:
                e += 2
                if e >= phi:
```

```
                p = self._generate_prime(self.key_size // 2)
                q = self._generate_prime(self.key_size // 2)
                n = p * q
                phi = (p - 1) * (q - 1)
                e = 65537
        return (n, e), (n, d)

    @staticmethod
    def _pkcs1_pad(data: bytes, block_size: int) -> bytes:
        if len(data) > block_size - 11:
            raise ValueError("数据过长，需要先分组。")
        padding_length = block_size - len(data) - 3
        padding = bytes([random.randint(1, 255) for _ in range(padding_length)])
        return b'\x00\x02' + padding + b'\x00' + data

    @staticmethod
    def _pkcs1_unpad(padded_data: bytes) -> bytes:
        if not padded_data.startswith(b'\x00\x02'):
            raise ValueError("无效的 PKCS#1 填充。")
        try:
            sep_index = padded_data.index(b'\x00', 2)
        except ValueError:
            raise ValueError("无效的 PKCS#1 填充。")
        return padded_data[sep_index+1:]

    def encrypt(self, plaintext: bytes, public_key: Tuple[int, int] = None) -> bytes:
        if public_key is None:
            public_key = self.public_key
        n, e = public_key
        max_block_size = (self.key_size // 8) - 11
        ciphertext = b''
        for i in range(0, len(plaintext), max_block_size):
            block = plaintext[i:i+max_block_size]
            padded_block = self._pkcs1_pad(block, (self.key_size // 8))
            m = int.from_bytes(padded_block, 'big')
            if m >= n:
                raise ValueError("填充后数据仍然过大。")
            c = pow(m, e, n)
            ciphertext += c.to_bytes(self.key_size // 8, 'big')
        return ciphertext

    def decrypt(self, ciphertext: bytes, private_key: Tuple[int, int] = None) -> bytes:
        if private_key is None:
            private_key = self.private_key
        n, d = private_key
```

```
        block_size = self.key_size // 8
        if len(ciphertext) % block_size != 0:
            raise ValueError("密文长度不正确。")
        plaintext = b''
        for i in range(0, len(ciphertext), block_size):
            block = ciphertext[i:i+block_size]
            c = int.from_bytes(block, 'big')
            m = pow(c, d, n)
            padded_block = m.to_bytes(block_size, 'big')
            try:
                plaintext += self._pkcs1_unpad(padded_block)
            except ValueError as e:
                raise ValueError("解密失败: 无效的填充。") from e
        return plaintext

if __name__ == "__main__":
    rsa = RSA(2048)
    print("公钥 (n, e):", rsa.public_key)
    print("\n 私钥 (n, d):", rsa.private_key)
    message = b"Hello, RSA! This is a long message that will be split into blocks and padded."
    print("\n 明文:", message.decode())
    encrypted = rsa.encrypt(message)
    print("\n 密文 (HEX):", encrypted.hex())
    decrypted = rsa.decrypt(encrypted)
    print("\n 解密所得:", decrypted.decode())
```

## 五、测试数据及其结果分析

公钥 (n, e):
(152873665382367571101950677869117454297828684196387911824945758365940712308592137985190023385866091859415449408264005014718037411375175910620156354218704108935703009853066724984058843868866424125913053480407708803165612775406547079871591819332163403754348646455949901264924022399761628997541283283763537462783718466695681956776999211630179609788585070995533782477514809133321929363122973287949612677330910288292434082380842096386187907776552732586299419992276704482729285471586552116693107271088985690460513603960283053784200620003711143808952423376337209805262444422329759248825043626111548005336711222232070794906796553537)

私钥 (n, d):
(152873665382367571101950677869117454297828684196387911824945758365940712308592137985190023385866091859415449408264005014718037411375175910620156354218704108935703009853066724984058843868866424125913053480407708803165612775406547079871591819332163403754348646455949901264924022399761628997541283283763537462783

718466695681956776999211630179609788585070995533782477514809133321929363122973
287949612677330910288292434082380842096386187907776552732586299419992276704448
272928547158655211669310727108898569046051360396028305378420062000371114380895
242337633720980526244442232975924882504362611154800533671122223207079490679,
122323191672663616408842234881921957113968234726316158751547302572744113301838
224452498051884505208616624901459776377053753053723126084879578268752235055670
424161402516056869526588732662211503138118628019478771528113962096866187479873
449730774656284127714772115751603252096002786876113106047956046211451615854316
329734297178365688675798055948554638378714347241122310581401262115710090281502
812952963000834024148158281714999770557639142933645361876806783442087427902822
745626821127815007058062784588698729149398404303024146768313794126177888633777
507524671633610053791465397009494249241960181773342555877909673244142 73)

明文: Hello, RSA! This is a long message that will be split into blocks and padded.

密文 (HEX):
45949a51ea79d294c979a3da9129226f18a7d584c9b11257b676b13e8304ff410d091bbbd7eebbee
e23912891e7d521b12a22f8849854a0e80e01589bcb6751be916408e1078a94584942589ce388c8
eb18c0ec8eb146eaae71421c4cea640330359561695274 43a4aa3277d1a9570afb96a80a1b30c8c8
6fef90d03416cbb477f882eb479baba32cd8c6313010162ea51fcf1a2e854c016d77f67fda5442c98
30d0dddd38ada991b0706401347bb6d4bf6cc090036d4dad7758408c0615e981f0b5b173244ff340
aa5a203d711588fc562b2e998dce5f469d5f25c58efbe290f76749a1559cdd0361aec57370965a129
4906db0b2e177f39516536786cf12ad

解密所得: Hello, RSA! This is a long message that will be split into blocks and padded.

## 六、调试过程中的问题

调试过程中未出现问题。

## 七、课程总结

通过本次实验，深入理解了 RSA 的原理，掌握了基于 Python 的数据结构和算法的实现方法。