

Practical 08 - RSA

RSA (Rivest-Shamir-Adleman) is one of the first public-key cryptosystems and is widely used for secure data transmission. It is an asymmetric cryptographic algorithm, meaning it uses two different keys: a public key for encryption and a private key for decryption.

RSA Algorithm

Key Generation

1. Choose two large prime numbers p and q .
2. Compute $n = pq$.
3. Compute $\varphi(n) = (p - 1)(q - 1)$.
4. Choose an integer e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$. Usually $2^{16} + 1 = 65537$ is used.
5. Compute d such that $de \equiv 1 \pmod{\varphi(n)}$.

The public key is (n, e) and the private key is (n, d) .

Encryption

For a plaintext message M , the ciphertext C is computed as:

$$C \equiv M^e \pmod{n}$$

Decryption

To decrypt the ciphertext C and recover the plaintext M :

$$M \equiv C^d \pmod{n}$$

Security Considerations

RSA's security relies on the difficulty of factoring large numbers. As computational power increases, larger key sizes are needed to maintain security.

Potential Attacks

1. **Factorization attacks:** Attempts to factor n to find p and q .
2. **Timing attacks:** Exploiting the time taken to perform private key operations.
3. **Padding oracle attacks:** Exploiting weaknesses in padding schemes.

To mitigate these, use:

- Large key sizes
- Secure padding schemes (e.g., OAEP)
- Constant-time implementations

Conclusion

RSA remains a cornerstone of modern cryptography. While its relative slowness compared to symmetric algorithms limits its use for bulk data encryption, its role in key exchange and digital signatures ensures its continued relevance in cybersecurity.

Listing 1: rsa.py

```

import random
import os

def miller_rabin(n, k=5):
    if n <= 1 or n == 4:
        return False
    if n <= 3:
        return True

    # Find r and s
    s = 0
    d = n - 1
    while d % 2 == 0:
        s += 1
        d //= 2

    # Witness loop
    for _ in range(k):
        a = random.randrange(2, n - 1)
        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

def generate_prime(bits=512):
    while True:
        n = random.getrandbits(bits)
        # Ensure n is odd
        n |= (1 << bits - 1) | 1
        if miller_rabin(n):
            return n

def greatest_common_divisor(a: int, b: int):
    if b == 0:
        return a
    else:
        return greatest_common_divisor(b, a % b)

def least_common_multiple(a: int, b: int):
    # LCM using the Euclidean Algorithm
    gcd = greatest_common_divisor(a, b)
    return abs(a * b) // gcd

class RSA:
    def __init__(self):
        self.p = generate_prime()
        self.q = generate_prime()

        # Compute n = pq
        self.n = self.p * self.q

        # Carmichael's totient function (lambda(n))
        self.lambda_n = least_common_multiple(self.p - 1, self.q - 1)
        # IMP: Secret

        # Choose e such that 1 < e < lambda(n) and
        # gcd(e, lambda(n)) == 1, that is e and lambda(n) are co-prime
        # for i in range(lambda_n, 1, -1):
        #     if greatest_common_divisor(i, lambda_n) == 1:
        #         self.e = i

```

```

# Most common value for e is 65537 (2**16 + 1)
self.e = 65537

# MMI(of x in base p) => pow(x, -1, p)
self.d = pow(self.e, -1, self.lambda_n)
# IMP: Secret

self.private_key = (self.d, self.lambda_n)
self.public_key = (self.e, self.n)

@staticmethod
def pad_pkcs1(unpadded: bytes, n: int) -> bytes:
    """pads a RSA string using PKCS1 v1.5 padding scheme

    Args:
        unpadded (bytes): Unpadded Plaintext (M)
        n (int): the modulus, from the public key

    Returns:
        bytes: padded plaintext (m)
    """
    mod_len = (n.bit_length() + 7) // 8 # Round up to next byte
    msg_len = len(unpadded)

    # Padding String must be atleast 8 bytes
    if msg_len > mod_len - 11:
        raise ValueError("[PKCS1 v1.5] PaddingError: Message too long.")

    padding_length = mod_len - msg_len - 3
    padding_string = b""
    while len(padding_string) < padding_length:
        padding_byte = os.urandom(1)
        if padding_byte != b"\x00":
            padding_string += padding_byte

    #          00||02 ||      PS      ||      00 ||M
    padded = b"\x00\x02" + padding_string + b"\x00" + unpadded

    return padded

@staticmethod
def unpad_pkcs1(padded: bytes) -> bytes:
    if len(padded) < 11:
        raise ValueError("[PKCS1 v1.5] PaddingError: Padded message too short.")

    if padded[:2] != b"\x00\x02":
        raise ValueError("[PKCS1 v1.5] PaddingError: Incorrect padding format.")

    sep_idx = padded.find(b'\x00', 2)
    if sep_idx == -1:
        raise ValueError("[PKCS1 v1.5] PaddingError: Separator not found.")

    return padded[sep_idx+1:]

@staticmethod
def encrypt(plaintext: bytes, public_key: tuple[int, int]) -> bytes:
    e, n = public_key

    padded = RSA.pad_pkcs1(plaintext, n)

    x = int.from_bytes(padded, "big")

    if x >= n:
        raise ValueError("[RSA] Padded message is too large to encrypt using this key")

    c = pow(x, e, n)
    return c.to_bytes((n.bit_length() + 7) // 8)

```

```

def decrypt(self, ciphertext: bytes) -> bytes:
    c = int.from_bytes(ciphertext, "big")
    if c >= self.n:
        raise ValueError("[RSA] Ciphertext is too large to decrypt for this key")

    m = pow(c, self.d, self.n)
    padded = m.to_bytes((self.n.bit_length() + 7) // 8)
    return RSA.unpad_pkcs1(padded)

class ANSI:
    RED = "\033[0;31m"
    GREEN = "\033[0;32m"
    BLUE = "\033[0;34m"

    BOLD = "\033[1m"

    END = "\033[0m"

if __name__ == "__main__":
    rsa = RSA()

    print(f"{ANSI.BOLD}RSA Key Components:{ANSI.END}")
    print(f"p (first prime): {hex(rsa.p)}")
    print(f"q (second prime): {hex(rsa.q)}")

    print(f"\n{ANSI.GREEN}Public Key (e, n):")
    print(f"({ANSI.BOLD}{hex(rsa.e)}{ANSI.END}{ANSI.GREEN}, {hex(rsa.n)}){ANSI.END}")
    print(f"\n{ANSI.RED}Private Key (d, λ(n)):")
    print(f"({ANSI.BOLD}{hex(rsa.d)}{ANSI.END}{ANSI.RED}, {hex(rsa.lambda_n)}){ANSI.END}")

    message = b"Practical 08 - Rivest-Shamir-Adleman PKCS - Devansh Parapalli"
    print(f"\nOriginal message: {message}")

    encrypted = rsa.encrypt(message, rsa.public_key)
    print(f"\nEncrypted message: {encrypted.hex()}")

    decrypted = rsa.decrypt(encrypted)
    print(f"\nDecrypted message: {decrypted}")

```

```
> python -u "c:\DevParapalli\Projects\RTMNU-SEM-7\CNS\practical\practical-08\rsa.py"
RSA Key Components:
p (first prime): 0xa2a468b18a52e2a1de86245cba4f53bea47c45c5d4d1b8befd6291022c3baa5a8d953f99279dff51815094c5d821deaa6ce5602202eec7ceb95123881726c791

q (second prime): 0x8ec35539475956358da890f84dc9afb27daeb9a3befa8763a96be1cf4c408ceb45da7b1f9470b398a61dff8bb71e2f67f412cea8bac98112e799133caad4395d

Public Key (e, n):
(0x10001, 0x5ab34b7334a8f444a0ba27d41b7af796bf1e1c185df83a9d5caa1dd21634140b8125ee2a2d7b2235d2bb57dc7b1f95f2802c8ce5611968130b0aedc5e547e2d231d63fc74775b484d5cf0e8b64a3ebb09cb6c6d5754470ec0ded48e446b48fb35d6e5de9bb4a29d955809583559ccc83ecb507ad8151369a05ab379999c8ad)

Private Key (d, λ(n)):
(0x2e8b944caad989812e7a59b61fe41aeae169334a77b6c18c52f51ad7f2d832219745cf914953cdbce0a2244212839cd1f680905c3dda768f523678854ce93cda2d09893310577a047c8244514f6fa00b44ce56ab24e6f7aeb31ac05e4892c17907d1943f5011220d2ddeb3a0abc1e45ce1535b102a7da028fb2d7bb4f3b2281, 0x16acd2dccc2a3d11282e89f506debd5afc7878706177e0ea7572a8774858d0502e0497b8a8b5ec88d74aed5f71ec7e57ca00b233958465a04c2c2bb717951f8683286153d66cf374629c66e60d2e81ea39c6dd7506a4113914fde9ddaf29f5277fb6ce94b6b25cfec79bb10fd0563ae88bd218f3bf242153e46dd1cb5e7b1f0)

Original message: b'Practical 08 - Rivest-Shamir-Adleman PKCS - Devansh Parapalli'

Encrypted message: 59ae9d0655d3800ead71064374b4013ad835a04847de9b6af75112beffacecfb66b9f5389c19de87b87c7f2eb8693f613a57d5145ff419e319cd0b49161c762532f013a5eb2204254bd26a4bddd4f4555658196338177747d65164c9d10cfb51d057112c8305e03ecd266219b638cf4be32e7af1ada7b111daba414e0c3041f9a

Decrypted message: b'Practical 08 - Rivest-Shamir-Adleman PKCS - Devansh Parapalli'

[?] pwsh [?] main [?] ~1
00:15:32 | 23 Aug, Friday | in C: → DevParapalli → Projects → RTMNU-SEM-7
>
```

Figure 1: Output