Network Security Technology Project

Yiheng Wang 522031910710

Task 1: Implement the textbook RSA algorithm (without any padding)

▶ 点击展开/折叠任务要求

Requirements

Goal: Implement the textbook RSA algorithm (without any padding)

Your code should be able to:

- Generate a random RSA key pair with a given key size (e.g., 1024-bit)
- Encrypt a plaintext with the public key.
- Decrypt a ciphertext with the private key.

Files to be Submitted and Standard of Grading:

- Code : 6 points
- RSA parameters (Decimal, 1024bits):
  - RSA_Moduler.txt 1 point
  - RSA_p.txt 1 point
  - RSA_q.txt 1 point
- RSA key (Decimal, 1024bits):
  - RSA_Secret_Key.txt 1 point
  - RSA_Public_Key.txt 1 point
- Encryption:
  - Raw_Message.txt 1 point
  - Encrypted_Message.txt (hexadecimal) 1 point
  - Pass Decryption (TA) 2 points

▶ 点击展开/折叠算法实现

Report

RSA 是最具代表性的公钥加密算法，而 **Textbook RSA** 指的是 *不加任何随机填充*、直接对明文执行模幂运算的最基本方案。本报告基于课程需求，对随附脚本 `textbook_rsa_assignment.py` 进行简要说明，帮助理解其实现细节。

算法概览

- **密钥生成**
  1. 随机生成两 512 bit 素数 `p`、`q`
  2. 计算模数 `n = p · q` 与 欧拉函数 `ϕ(n) = (p − 1)(q − 1)`
  3. 取公钥指数 `e = 65537`
  4. 通过扩展欧几里得算法求私钥 `d ≡ e⁻¹ (mod ϕ(n))`

- **加密** 将明文字节串转为整数 `m`，计算 `c ≡ mᵉ (mod n)`，以十六进制保存。
- **解密** 读取密文整数 `c`，计算 `m ≡ cᵈ (mod n)`，还原为原始字节流输出。

代码结构

函数 / 模块	作用	说明
`is_probable_prime()`	Miller–Rabin 素性测试	默认 40 轮，确保高置信度
`keygen()`	生成 `p, q, n, e, d`	自动规避 `gcd(e, ϕ) ≠ 1` 冲突
`encrypt() / decrypt()`	裸模幂运算	直接调用 Python 内置 `pow(base, exp, mod)`
CLI 接口	`generate / encrypt / decrypt`	对接测评所需 8 个文件

文件接口一览

随附文件为本地测试结果，可更换原始明文和密钥以重新测试。

文件名	内容 / 格式
`RSA_p.txt`, `RSA_q.txt`	素数 `p, q`（十进制）
`RSA_Moduler.txt`	模数 `n`（十进制）
`RSA_Public_Key.txt`	公钥指数 `e`（十进制）
`RSA_Secret_Key.txt`	私钥指数 `d`（十进制）
`Raw_Message.txt`	原始明文（字节流）
`Encrypted_Message.txt`	密文（十六进制字符串）
`Decrypted_Message.txt`	解密结果（字节流）

▶ 点击展开/折叠使用说明

How to use

```
# 生成 RSA 参数
python textbook_rsa_assignment.py generate

# 对指定明文加密
python textbook_rsa_assignment.py encrypt <input_plaintext_file>

# 对指定密文解密
python textbook_rsa_assignment.py decrypt <input_cipher_hex_file>
```

Task 2: Perform a CCA2 attack on textbook RSA

▶ 点击展开/折叠任务要求

Requirements

Goal : Perform a CCA2 attack on textbook RSA

Textbook RSA is elegant, but has no semantic security.

An adaptive chosen-ciphertext attack (abbreviated as CCA2) is an interactive form of chosen-ciphertext attack in which an attacker sends a number of ciphertexts to be decrypted, then uses the results of these decryptions to select subsequent ciphertexts.

The goal of this attack is to gradually reveal information about an encrypted message, or about the decryption key itself.

Refer an existing work for the implementation: (Details of this attack can be found in Chap 4.)

Knockel J, Ristenpart T, Crandall J. When textbook RSA is used to protect the privacy of hundreds of millions of users[J]. arXiv preprint arXiv:1802.03367, 2018. (https://arxiv.org/abs/1802.03367)

In this attack, the server knows

- RSA key pair
- AES key

The adversary knows

- RSA public key
- a RSA-encrypted AES key
- an AES-encrypted WUP request

The adversary wants to know

- AES key

In this part, you are supposed to:

- Properly design your own WUP request format, server-client communication model, etc.
- Generate a history message by yourself, it should includes a RSA-encrypted AES key and an AES-encrypted request.
- Present the attack process to obtain the AES key (and further decrypt the encrypted request) from the history message.

You can use third-party library to implement AES encryption and decryption.

Files to be Submitted and Standard of Grading:

- Code : 10 points
- CCA2 (Use RSA parameters in task 1):

  - History_Message.txt 1 point
  - AES_Key.txt (hexadecimal, 128bits) 1 point
  - WUP_Request.txt (hexadecimal) 1 point
  - AES_Encrypted_WUP.txt (hexadecimal) 2 points
  - Attack Process to Obtain the AES key: 10 points
    - Both Screenshot and Log Files are OK

▶ 点击展开/折叠算法实现

Report

> **环境**：Python 3 + PyCryptodome

> **脚本**：`cca2_attack_demo.py`（依赖 `textbook_rsa_assignment.py` 的 1024-bit Textbook RSA 实现）

> **参考**：Knockel, Ristenpart & Crandall, *Analyzing QQ Browser* §4.1

协议回顾

步骤	QQ Browser 6.5 行为
①	客户端用 **“当前时间（ms）”作种子**生成 128-bit AES 会话密钥 *k*
②	用 *textbook RSA*（e = 65537, 1024-bit *n*）加密 *k* → `C = k^e (mod n)`
③	用 *k* 以 **AES-ECB** 加密 WUP (JSON) 请求 → `CT_AES`
④	将 `C ‖ CT_AES` 发送服务器
⑤	服务器解密 `C` 并 **取最低 128 bit** 作为会话密钥
⑥	若 `CT_AES` 解密后是合法 JSON，则回应，否则静默

无填充 + LSB 截断使服务器自然成为 **按位泄露**的选择密文（CCA2） oracle。

攻击原理

设原密钥 *k* 的 RSA 密文为 `C = k^e (mod n)`，公开 *(n, e)*。

- 把 *k* 左移 *b* 位得 `k_b = 2^b·k`；其密文 `C_b = (2^b)^e·k^e = C·(2^{be} mod n)` 可由攻击者直接计算。
- 服务器仅保留 LSB-128，因此 `k_b` 的高 *(128 − b)* 位全为已知 0。 猜测该位为 0 并提交 AES 密文；若服务器响应 → 猜对，否则该位必为 1。
- 从最高位到最低位迭代 128 次即恢复完整 *k*。

该方法正是论文 §4.1 “CCA2 attack”，最多 128 oracle 查询即可。

脚本结构

模块 / 函数	**作用**	**关键点**
`setup_phase()`	扮演受害者客户端，生成工件	*PRNG(seed=now_ms)* 生成 AES key；产生最小 JSON WUP；写 `History_Message.txt` 等四个文件
`Oracle` 类	模拟服务器	RSA 私钥解密 → 取 LSB-128 → AES-ECB 解密 → `json.loads()` 验证
`attack_phase()`	128 轮查询恢复 k	预计算 `2^e mod n`；循环 $(b = 127 \to 0)$ 构造 `C_b` 与假密钥；判断服务器响应更新位
辅助	`aes_encrypt_ecb`, `aes_decrypt_ecb`, `int_to_bytes` 等	纯 Python，无外部网络交互

输出文件一览：

文件	**生成阶段**	**内容**
`History_Message.txt`	setup	第1行 RSA 密文 (hex); 第2行 AES 密文 (hex)
`AES_Key.txt`	setup	真实 AES-128 key (供评分)
`WUP_Request.txt`	setup	明文 JSON (hex)
`AES_Encrypted_WUP.txt`	setup	加密的 WUP (hex)
`Recovered_AES_Key.txt`	attack	攻击后复原的 key (hex)

▶ 点击展开/折叠使用说明

How to use

```
# 生成 RSA 参数
python textbook_rsa_assignment.py generate

# 生成历史消息
python cca2_attack_demo.py setup

# 对捕获的 History_Message.txt 发起 CCA2 攻击
python cca2_attack_demo.py attack
```

示例结果：

```
[+] Key pair generated and files written.
[SETUP] Loading RSA parameters …
[SETUP] AES key (from PRNG seeded 1750242148669): 61967946b2cde0fc5f98c57510df8219
[SETUP] Files generated: History_Message.txt etc.
[ATTACK] Loading data …
[ATTACK] Starting 128-query bit-oracle attack …
    recovered k[  0] = 1
    recovered k[  1] = 0
    recovered k[  2] = 0
    recovered k[  3] = 1
    ...
    recovered k[127] = 0
[ATTACK] Recovered AES key: 61967946b2cde0fc5f98c57510df8219
[ATTACK] Successfully decrypted victim WUP →
{"imei":"990000862471854","url":"https://example.com","ts":1750242148}
[ATTACK] Total oracle queries: 128
```

Task3: Implement RSA-OAEP algorithm and discuss why it can defend such kind of attacks

▶ 点击展开/折叠任务要求

Requirements

Goal: defend the attack

- Implement RSA-OAEP algorithm and discuss why it can defend such kind of attacks.

Since textbook RSA is vulnerable to attacks, in this paper, the authors give a solution: using OAEP key padding algorithm.

In cryptography, Optimal Asymmetric Encryption Padding (OAEP) is a padding scheme often used together with RSA encryption. OAEP satisfies the following two goals:

- Add an element of randomness which can be used to convert a deterministic encryption scheme (e.g., traditional RSA) into a probabilistic scheme.
- Prevent partial decryption of ciphertexts (or other information leakage) by ensuring that an adversary cannot recover any portion of the plaintext without being able to invert the trapdoor one-way permutation.

In this part, you are supposed to

- Add the OAEP padding module to the textbook RSA implementation.
- Give a discussion on the advantages of RSA-OAEP compared to the textbook RSA.
- Further try to present CCA2 attack to RSA-OAEP to see whether it can thwart the CCA2 attack you have implemented in part 2.

Files to be Submitted and Standard of Grading:

- Code : 10 points
- Encryption (Use RSA parameters and Message in task 1):
  - Random_Number.txt 1 point
  - Message_After_Padding.txt (hexadecimal) 1 point
  - Encrypted_Message.txt (hexadecimal) 1 point
  - Pass Decryption (TA) 2 points
    - (Recommended using n=1024, k0=512, hash: sha512)

- Any extra file added is OK but need to be explained in report!

▶ 点击展开/折叠算法实现

Report

> **脚本**：`rsa_oaep_assignment.py`（配合先前的 `textbook_rsa_assignment.py` 随附工具函数）

> **参数**：n = 1024 bit，k = 128 B，k_0 = 64 B，k_1 = 1 B（固定 0x00），G = H = SHA-512

设计要点

项目	取值	说明
模数长度 *k*	128 B	1024-bit RSA，符合课程要求
随机串 *r* (k_0)	64 B / 512 bit	`os.urandom` 生成，写入 `Random_Number.txt`
`k₁` 尾字节	1 B (0x00)	用于帧分隔，确保解码可验证
G, H	SHA-512	自定义 **MGF1**：`mgf_sha512()` 可裁剪扩展
最长明文	$k - k_0 - k_1$ = 63 B	超出抛出 `message too long`

OAEP 编码流程 (`oaep_encode`)

1. **填充明文**：`m‖0x00` 并右补零至 $(k - k_0)$ 字节长度。
2. **生成随机串** *r* (64 B)。
3. **G(r)**：使用 `mgf_sha512` 将 *r* 扩展至 64 B。
4. **X = m′ ⊕ G(r)**。
5. **Y = r ⊕ H(X)**，其中 H = SHA-512。
6. 输出 **EM = X‖Y** (128 B) 并返回 *(EM, r)*。

生成文件：

文件	内容
`Random_Number.txt`	r (hex, 64 B)
`Message_After_Padding.txt`	EM (hex, 128 B)
`Encrypted_Message.txt`	RSA 密文 (hex)

OAEP 解码流程 (`oaep_decode`)

1. 拆分 EM 为 **X** (64 B) 与 **Y** (64 B)。
2. 复原 *r* = Y ⊕ H(X)。
3. 计算 **G(r)** 并得到 *m′* = X ⊕ G(r)。
4. 验证末尾 `k₁` = 0x00 字节；去除右侧零填充，还原明文 *m*。
5. 如验证失败抛出 `decoding error`。

RSA-OAEP 为何能抵御逐比特 CCA2 攻击?

1. 随机化打破“同态偏移”

- **Textbook RSA** 的明文到密文是简单的幂模运算

  $$ C = m^e \pmod n $$

  乘上 2^{be} 会对应明文左移 b 位，攻击者就能通过构造 $C_b=C\cdot 2^{be}\bmod n$ 并观察解密结果来逐位恢复密钥。

- **OAEP** 先用随机串 r 和两趟掩码函数 G,H 对消息做双向掩盖：

  $$ X = (m\parallel0)\oplus G(r),\quad Y = r \oplus H(X),\quad M' = X\parallel Y $$

  最终加密的是 M'，它与原消息 m 再无线性关系。即便乘以 2^{be}，“左移”效果会被掩码破坏，解密端无法通过简单裁剪得到结构合法的数据，服务器因填充校验失败而拒绝响应，从而彻底熄火位-oracle。

2. “全或无”安全属性

- 恢复明文必须**同时**拿到完整的 X 与 Y；任何一比特错误都会导致哈希结果完全不同，解码链条立即断裂。这使得分步泄露或逐位猜测的攻击不再可行。

相比 Textbook RSA 的优势

维度	Textbook RSA	RSA-OAEP
安全级别	仅保证密码学硬问题（RSA 因子分解）→ **IND-CPA 不安全**，易遭 CCA2、重放、广播、共模等攻击	经过理论证明可达 **IND-CCA2**；填充随机化抵御已知选择密文/明文攻击
随机化	无；同一明文→同一密文，可被流量特征分析	引入 k_0 位随机种子，每次加密输出不同密文
错误传播	明文与密文一一映射，局部篡改易被“位翻转”利用	解码需验填充与哈希，全局一致性，局部篡改→整体失败
标准化/兼容	非标准、不被现代协议采纳	被 PKCS #1 v2.3、FIPS 186-5、TLS 1.3 等广泛采用
实现风险	易被开发者无意调用 *NoPadding* 造成漏洞	主流库默认提供 `OAEP` 接口并做参数校验，误用概率低

OAEP 通过**随机种子+双哈希掩码**打破了 RSA 的可乘性同态，从根本上封堵逐比特 CCA2 利用的“左移泄露”通道；同时提供随机化、填充校验和经过形式化证明的 CCA2 安全，因而在现代协议中取代了 textbook RSA，成为默认且推荐的公钥加密填充方案。

▶ 点击展开/折叠使用说明

How to use

OAEP 加解密可行性验证：

```
# 生成 RSA 参数
python rsa_oaep_assignment.py generate

# 加密明文
python rsa_oaep_assignment.py encrypt Raw_Message.txt

# 解密并验证
python rsa_oaep_assignment.py decrypt Encrypted_Message.txt
```

OAEP 抵抗 CCA2 攻击能力验证：

```
# 生成 RSA 参数
python rsa_oaep_assignment.py generate

# 生成历史消息
python cca2_attack_demo.py setup

# 对捕获的 History_Message.txt 发起 CCA2 攻击
python cca2_attack_demo.py attack
```

示例结果：

```
[+] 1024-bit RSA key pair generated.
[SETUP] Loading RSA parameters …
[SETUP] AES key (from PRNG seeded 1750255303212): 193f22ed77a0fcfd6d3ce7bdb5e9a07c
[SETUP] Files generated: History_Message.txt etc.
[ATTACK] Loading data …
[ATTACK] Starting 128-query bit-oracle attack …
    recovered k[  0] = 1
    recovered k[  1] = 1
    recovered k[  2] = 1
    recovered k[  3] = 1
    ...
    recovered k[127] = 1
[ATTACK] Recovered AES key (expected to be WRONG with OAEP):
ffffffffffffffffffffffffffffffff
[ATTACK] Verification failed (as intended): Padding is incorrect.
[ATTACK] Total oracle queries: 128
```