Network Security Technology Project

Neng Li
ln-fjpt@sjtu.edu.cn

Part I

Implement the textbook RSA algorithm.

- The textbook RSA is essentially RSA without any padding.

Choose two large primes p and q. Let $n = p \cdot q$. Choose e such that $\gcd(e, \varphi(n)) = 1$, where $\varphi(n) = (p - 1) \cdot (q - 1)$. Find d such that $e \cdot d \equiv 1 \bmod \varphi(n)$. In other words, d is the modular inverse of e, *i.e.*, $d \equiv e^{-1} \bmod \varphi(n)$

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

To encrypt a plaintext m, compute $c \equiv m^e \bmod n$
To decrypt a ciphertext c, compute $m \equiv c^d \bmod n$

Part I

Goals

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

```
lineng@ln-Surface:/mnt/d/project/demo/project$ python rsa.py
key size: 1024

public key: 65537, 113786939598950713723693400447768822542986255959758176344972883680548352984782993908812639654930
85975606865185835302530471222873033894501010329811662778787550474638266179609011209086334365939810077333568410507549
29068311812627936123501129548211156894214987733908927271004876454516560751216535069918547715415113

private key: 484736494905641847574926537238698408010380896309029788573675624376646698350879107517912032367367238892
4852665262917303938633671396814100997512551965015329304286852311117627956832533230412121586785798172898142350805168
5212744792834495276156621507523777266439029649558401993467021311200173130665775226017748269, 1137869395989507137236
93400447768822542986255959758176344972883680548352984782993908812639654930859756068651858353025304712228730338945010
1032981166277878755047463826617960901120908633436593981007733356841050754929068311812627936123501129548211156894214
987733908927271004876454516560751216535069918547715415113

message = hello world

encrypted = 6454378231592564368072102412183896710747541519499190406782533645814147374340917031864927204906224105261
6659246906730922576977456510933214115714607368145984592548034299736715733660013516559958253022530269773830892207648
341852473229661217371735247416856755570518150310603836556411786106979815604876320824613476

decrypted = hello world
```

Part II

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.

Part II

We refer to an existing work to implement our attack.

When Textbook RSA is Used to Protect the Privacy of Hundreds of Millions of Users

Jeffrey Knockel
Dept. of Computer Science
University of New Mexico
jeffk@cs.unm.edu

Thomas Ristenpart
Cornell Tech
ristenpart@cornell.edu

Jedidiah R. Crandall
Dept. of Computer Science
University of New Mexico
crandall@cs.unm.edu

4 Active Attacks on QQ Browser's Use of Textbook RSA

In this section, we explore attacks on QQ Browser's use of textbook RSA.

4.1 CCA2 attack

Part II

Server-client communication

① generate a 128-bit AES session key for the session.

② encrypt this session key using a 1024-bit RSA public key.

③ use the AES session key to encrypt the WUP request.

④ send the RSA-encrypted AES session key and the encrypted WUP request to the server.

---

① decrypt the RSA-encrypted AES key it received from the client.

② choose the least significant 128 bits of the plaintext to be the AES session key.

③ decrypt the WUP request using the AES session key.

④ send an AES-encrypted response if the WUP request is valid.

Part II

CCA2 attack

Let C be the RSA encryption of 128-bit AES key k with RSA public key (n, e). Thus, we have

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

Now let C_b be the RSA encryption of the AES key

$$k_b = 2^b k$$

i.e., k bitshifted to the left by b bits. Thus, we have

$$C_b \equiv {k_b}^e \pmod{n}$$

We can compute C_b from only C and the public key, as

$$\begin{aligned} C_b &\equiv C(2^{be} \bmod n) \pmod{n} \\ &\equiv (k^e \bmod n)(2^{be} \bmod n) \pmod{n} \\ &\equiv k^e 2^{be} \pmod{n} \\ &\equiv (2^b k)^e \pmod{n} \\ &\equiv {k_b}^e \pmod{n} \end{aligned}$$

We begin the attack by considering C_{127}. It is the RSA encryption of k_{127}, the AES key where every bit but the highest bit are necessarily zero and where k_{127}'s highest bit is k's lowest bit (recall that the QQ Browser server ignores all but the lowest 128 bits of the decrypted key). We first guess that k_{127}'s high bit is zero and send a WUP request with C_{127} and encrypt the request with the key where that bit is zero. If the server responds, that means that the bit was zero, since it was able to decrypt our request. If not, the bit must have been a one. After we know this bit, we consider C_{126} and guess the next bit (note that we know one of C_{126}'s bits from C_{127}). We repeat this process for each bit of the AES key. In total, this requires 128 guesses, since the AES key is 128 bits and each request reveals one bit of the key. By using this approach, we can iteratively learn every bit of the AES key.

Part II

Goals

- In a basic version, you should present the attack process to obtain the AES key (and further decrypt the encrypted request) from a history message.
- The history message can be generated by yourself in advance, it should includes a RSA-encrypted AES key and an AES-encrypted request.
- Feel free to design your own WUP request format, server-client communication model, etc. A nice design will bring you a bonus.
- AES encryption and decryption can be achieved with the help of third-party library.

Part II
Demo

- What server knows: RSA key pair, AES key.
- What client (attacker) knows: RSA public key, a RSA-encrypted AES key, an AES-encrypted WUP request.
- The attacker wants to learn the AES key.

```
public key: 65537, 113737834153044474165203214471243357136920612769752487582665551705260017363108699011808643231612
703266904358607556631744095288857619708463307885826576742338244359660512956064905337081105876241854920243050216005 8
855477371257680704604358011480149174213984750399992658875566305731119098343575351125038221014590963
private key: 386715925115932987140580622863835043880861500274409063705164485504113549125250032818122891974003367089
8011539820535613704739798303767282127484657328851715897579077997016549439239360911695907269654925388149145911157156
816832312491305524949692081387496654549607977073293406630557947535347602725556719913361 4233, 1137378341530444741652
0321447124335713692061276975248758266555170526001736310869901180864323161270326690435860755663174409528885761970846
3307885826576742338244359660512956064905337081105876241854920243050216005885547737125768070460435801148014917421398
475039999265887566305731119098343575351125038221014590963
aes_key = 0000111122223333
aes_key = 30303030313131313232323233333333
```

Part II

Demo

- CCA2 attack
- 128-round guesses (k127 - k0).

```
  Cb: 2f225818dbdea25f0e10503b287c66bef620ef8d5b3fafc71654031c23b682aad25b438128f036e96e7b38118bc0d35d42c98cc66f0df
cf1408314910d703d9100bf8a15ee07e7161783d464e8b07fbec3fd33c7a7a4c1c9741b3917427071ebd0d080cfc83f92625961c31f7132fba7
8da2a2e7c22c1c4ba28bcf6188617810
  kb: 7e9c551e07a7b38cf48aed115a544e4a72a7062b7f38c0d703a4959c5910fa7b4587150fd4ac9702ce8f94e14e12b25b8b99d2805b736
77ba2fff0f5e7df0995a795684b8ed9db378d04cc198360289769429c366cf4d73057690a1e8509d144f967ee8c942119195f574f4869ed4d22
bd3e087dcca37d42935a0a5d946314c3
complete aes key: 1818181818989898991919191999999998000000000000000000000000000000

k127: 00000000000000000000000000000000
encrypted_msg: 2a0aa54dba2db7205aa83f71531af651
current key: 80000000000000000000000000000000
wup request: I[`□3□(□

k127: 80000000000000000000000000000000
encrypted_msg: 9d8f0724790727d6f494f124006a2306
current key: 80000000000000000000000000000000
wup request: test wup request

80000000000000000000000000000000
------------------------------
  Cb: 80b1136399c379110e9bdffb60bfcb70a2765a56fc169a534faddaefde84e5a5cd063490e2ad22be90a2081554ce5dd7632b75ac2a4a0
61c898d7f115c75ac8c028e205ae65b27e30410d49b580c0464221e1b63c3e65e1b8a8a76f9898598e4554685e3aba48d1695df029957058a04
8f83d9948c200ab8ffe8376185bf5269
  kb: f0b569a5e88030d0a385db96b0182bbf6b7b4882c7b75098b77a12ab85b94bf104cb11d2d102f56960ded373067c497cd1872a3deb718
66427b7d0b74bc55fe2c292511ea104190effb5a7a30de6b7043e1bf5416cfb54810e7e9fda14120e9807d872d721a0c62df015cef8f0061556
ae8d46b424b2706b324810b4ce3af5b
complete aes key: c0c0c0c0c4c4c4c4c8c8c8c8cccccccc00000000000000000000000000000000

k126: 40000000000000000000000000000000
encrypted_msg: 0bc86d5f447ab83ab9fac620e175aa56
current key: c0000000000000000000000000000000
wup request: C? □$s^□□□□

k126: c0000000000000000000000000000000
encrypted_msg: 0cbd2cd5552a7fe84e2a71e28f2c3d4f
current key: c0000000000000000000000000000000
wup request: test wup request

c0000000000000000000000000000000
```

Part II

Demo

- In the final round (k0), the attacker can revert the AES key successfully.

```
  Cb: 3c77557f1c0d744e46aea30145613ad8fa4e532e299d47e0f1912a91906a72dc4e6426e030fa29d19c4641fe232744a11452e5871a970
b1f9c5fa1a2bde62ee994cddbf1964e5f2db096b3f0fb9d4165ed7254760ae4f15c8800a3fac14a3ba675d3e0c941e2fb497fb482937b92cdcd
49aa7a818bc8be2b9db25e178b80920d
  kb: 992bd0d3125a5c5ce33608ba1904b5154646120bf7ea5b68dfee8f814e38e404709399de399a5f522d18d09910fedd2531656fc1a6252
146ee9c1b637ddb605f30fe7790bf75dd01f2b180200580be6c74c87851a53cd4368eb33721ed50aefa820867e29091819b6fb88e97a0114cc9
b0a65b3f56f69afd296c67ae26e78651
complete aes key: 30303030313131313232323233333333

k0: 30303030313131313232323233333333
encrypted_msg: 4bc41b964b18b094150dca157caeff98
current key: 30303030313131313232323233333333
wup request: test wup request

30303030313131313232323233333333
------------------------------
```

Part III

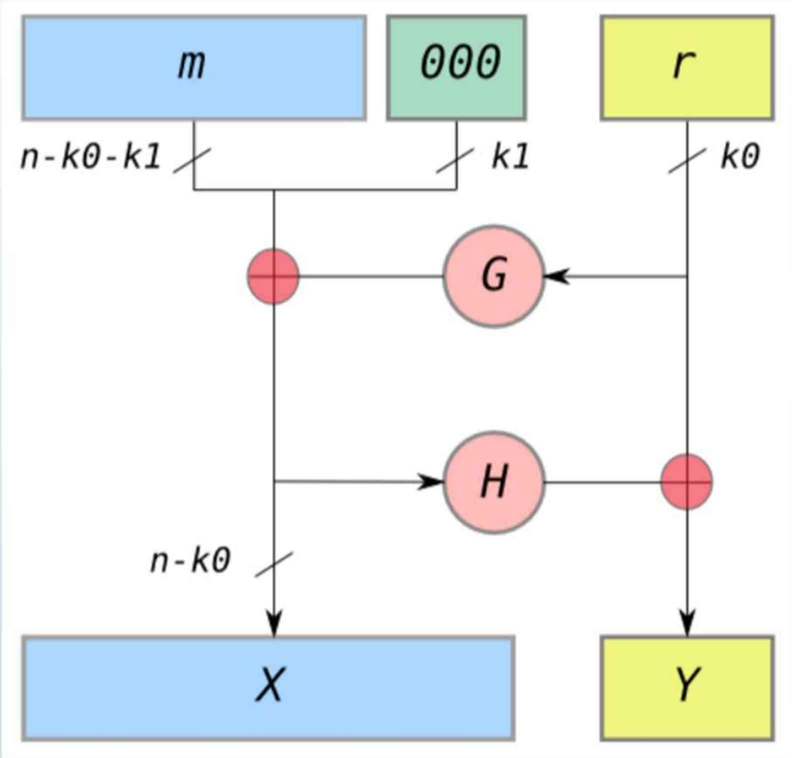
Implement an RSA-OAEP algorithm and discuss why it can thwart such kind of attacks.

- 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.

Part III

OAEP

- n is the number of bits in the RSA modulus.
- k0 and k1 are integers fixed by the protocol.
- m is the plaintext message, an (n−k0−k1) bit string
- G and H are typically some cryptographic hash functions fixed by the protocol.
- ⊕ is an xor operation.



Part III

OAEP encode

1. messages are padded with k1 zeros to be n−k0 bits in length.

2. r is a randomly generated k0 bit string

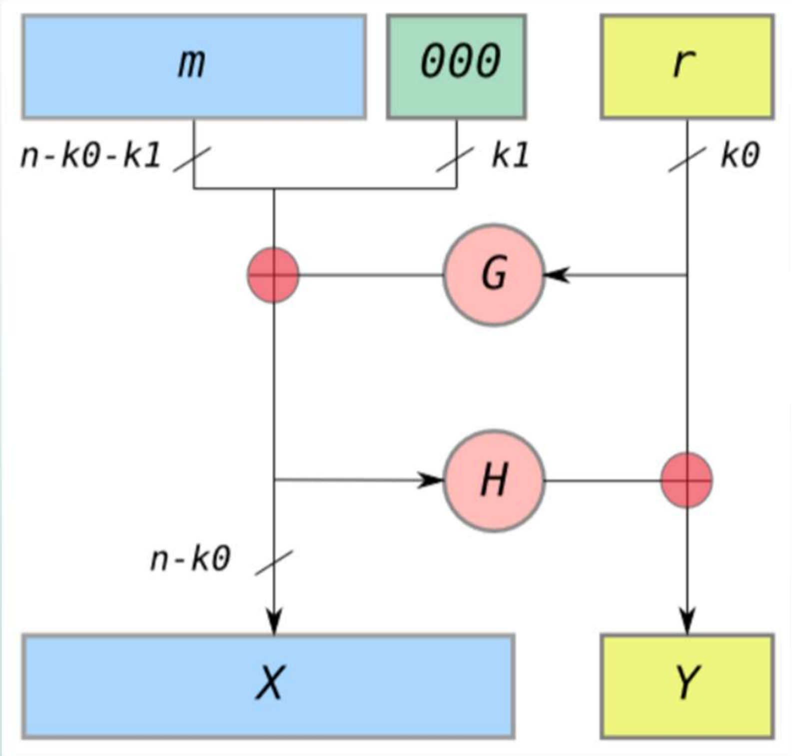
3. G expands the k0 bits of r to n−k0 bits.

4. X = m00..0 ⊕ G(r)

5. H reduces the n−k0 bits of X to k0 bits.

6. Y = r ⊕ H(X)

7. The output is X || Y where X is shown in the diagram as the leftmost block and Y as the rightmost block.



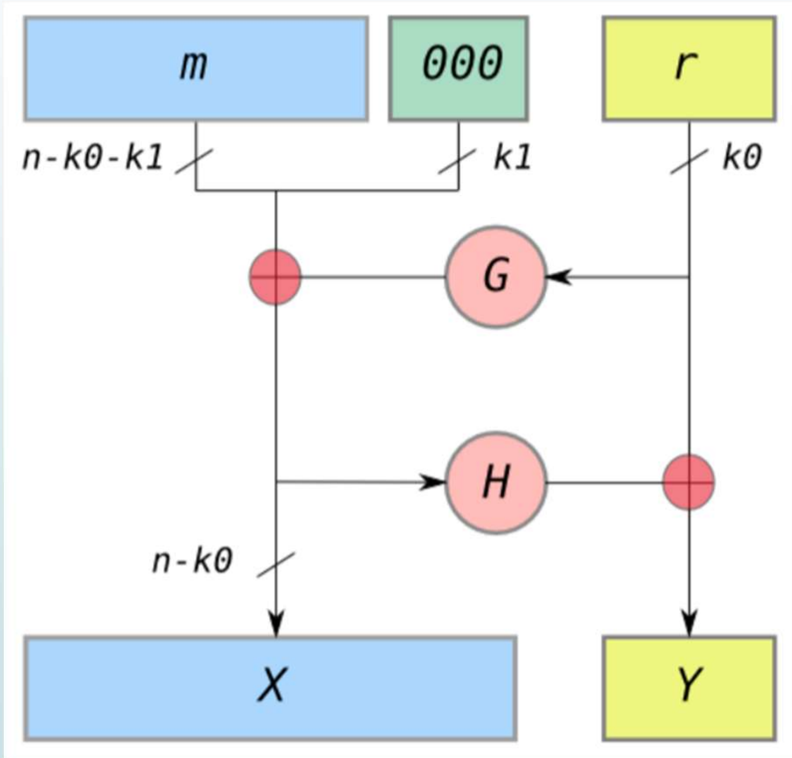
Part III

OAEP decode

1. recover the random string as $r = Y \oplus H(X)$

2. recover the message as $m00..0 = X \oplus G(r)$

The "all-or-nothing" security is from the fact that to recover m, you must recover the entire X and the entire Y; X is required to recover r from Y, and r is required to recover m from X. Since any changed bit of a cryptographic hash completely changes the result, the entire X, and the entire Y must both be completely recovered.



Part III

Goals

- You can achieve it by adding the OAEP padding module to the textbook RSA implementation.
- You should give a discussion on the advantages of RSAOAEP compared to the textbook RSA.
- As a bonus, you can further try to present that RSA-OAEP can thwart the CCA2 attack you have implemented in part 2.

Note

- Feel free to choose your preferred language to do this project (python recommended).
- You must not implement RSA & CCA2 & RSA-OAEP by directly using existing libraries.

Thank You