ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Rivest Shamir Wagner time-lock puzzle implementation

Laboratory for cryptologic algorithms
Bachelor semester project

Autumn 2019

Novak Kaluderovic, Dusan Kostic, Arjen K. Lenstra

Lucie HOFFMANN

January 9, 2020

Contents

1 Introduction **3**
1.1 Outline . . . 3

2 Background **4**
2.1 RSA . . . 4
2.2 OTP . . . 4
2.3 AES and CBC encryption mode . . . 4

3 RSW Time-lock puzzle **5**

4 Implementation of the RSW Time-lock puzzle **6**
4.1 Tools . . . 6
4.2 Functions . . . 6
4.2.1 main.c . . . 6
4.2.2 Squarings.c . . . 7
4.3 Workflow . . . 7
4.3.1 Message encryption . . . 8
4.3.2 Image encryption . . . 8
4.3.3 Note for image encryption . . . 8

5 Results **9**
5.1 Time as a function of the number of modular squaring operations . . . 9
5.2 Time as a function of the modulus size . . . 9
5.3 Concrete example of encryption time and number of squaring operations . . . 10

6 Conclusion **11**

7 References **11**

1 Introduction

The goal of time-release cryptography is to encrypt a message such that it can be decrypted only after a certain amount of time has passed. Several applications are suggested in the Rivest, Shamir and Wagner paper on time-lock puzzles [1], such as sealed-bid auctions, payement in several times, encrypting a diary for some years, and key escrow - where the third party gains access to the key only after some delay.

The RSW paper [1] discusses two ways for implementing time-release crypto: the time lock puzzle and trusted agents. The time lock puzzle is a computational problem that cannot be solved without having a computer running continuously for a certain amount of time: this problem must be intrinsically sequential.

In the secret sharing approach, some trusted agents promise not to reveal information until a specified time: they must be trustworthy.

Here we study and implement the first approach: the RSW time-lock puzzle.

1.1 Outline

We first set the mathematical and technical background. Then we describe the implementation for the time-release encryption of a message and a picture. Finally we show the observations made on the different parameters of the implementation.

2 Background

The time-lock puzzle is used to encrypt a symmetric key and decrypt it in the desired amount of time. The symmetric key is used both for encryption and decryption. As in the RSA cryptosystem, the RSW puzzle [1] relies on the hardness of RSA modulus factorization.

2.1 RSA

RSA is a public key cryptosystem. It takes its name from the name of its inventors - Rivest, Shamir and Adleman. RSA works as follows:

```
1: function ENCRYPT($M$)                          ▷ $M$ is the plaintext to encrypt
2:     $n \leftarrow pq$                          ▷ $p$, $q$ are distinct primes
3:     $k \leftarrow (p-1)(q-1)$                  ▷ $k$ is the private key
4:     $e \leftarrow$ integer coprime with $k$    ▷ $e$ is the public key
5:     $C \leftarrow M^e \mod n$                  ▷ $C$ is the ciphertext
1: function DECRYPT($C$)
2:     $d \leftarrow e^{-1} \mod k$
3:     $M \leftarrow C^d \mod n$
```

In practice, we choose p and q to be big enough so that n is very large (as of today, 2048 bits for n is considered secure). Here we assume M is smaller than n.

Let's assume the private and the public key, k and e, are Alice's. If Bob wants to send a secret message to Alice, he encrypts the message using the public key and sends the obtained ciphertext to Alice. Alice can then decrypt the message with her private key (that only she knows).

2.2 OTP

OTP, One-time pad, is a simple symmetric encryption scheme in which the security of the symmetric key is not guaranteed when used more than once. It works as follows:

$$C \leftarrow K \oplus M$$

$$M \leftarrow K \oplus C$$

where $\oplus$ is the bitwise XOR operation and K is the symmetric key.
The key K, the ciphertext C and the plaintext M have the same length.

2.3 AES and CBC encryption mode

AES, Advanced Encryption Standard, is one of the most widely used symmetric encryption algorithms. It is a block cipher algorithm : encrypts data by fixed-length blocks.

Block cipher has high diffusion (a single bit changed in the plaintext makes a significant difference in the resulting ciphertext) and is immune to tampering (it is difficult to insert symbols without detecting it).

CBC, Cipher Block Chaining, is a mode of operation of the block cipher algorithm where block encryption depends on the previous blocks.

3 RSW Time-lock puzzle

As described in the RSW paper [1], several issues come with the time-lock puzzle approach for time-release crypto:

- "The CPU time and the real time must agree as closely as possible." This is a problem since, as technology improves, CPUs get faster.
- The puzzle must be "intrinsically sequential", i.e. not parallelizable, such that using more computational resources in parallel does not speed up the resolution of the puzzle.
- Different hardware in computers may lead to different speed in solving the puzzle.
- It is costly in terms of computational resources since the puzzle is solved continuously during the specified amount of time.

The RSW time-lock puzzle [1] is based on repeated modular squaring operations. It allows us to encrypt a message/file using any symmetric key cryptosystem and decrypt it in a given time. The only important parameter here is the symmetric key, which we encrypt and decrypt using the puzzle. The other parameters related to the message encryption/decryption are independent from the puzzle.

The symmetric key K is encrypted as $C_K = K + a^{2^t} \mod n$, where n is a composite RSA modulus, a a positive integer modulo n and t the number of squaring operations depending on the encryption time desired.

We decrypt C_K as $K = C_K - a^{2^t} \mod n$.

The encryption can be done faster than the decryption by reducing the exponent modulo $\phi(n)$, the totient of n, thanks to Euler's theorem :

1. $e = 2^t \mod \phi(n)$
2. $C_K = K + a^e \mod n$

Since we presume that squaring is a sequential operation, under the assumption that the factorisation of n is unknown, this puzzle is intrinsically sequential.

4 Implementation of the RSW Time-lock puzzle

OTP was used in the LCS35 MIT puzzle [2], which was supposed to be decrypted in 35 years but has been solved in 2.5 years on a desktop computer. We also used OTP to implement the creation and solution of the corresponding puzzle for a message encryption/decryption.

We used AES CBC algorithm as a secure cryptographic primitive for the image encryption/decryption.

4.1 Tools

- C language for efficiency.
- GMP library for big integers.
- OpenSSL library for AES CBC symmetric encryption.
- File encryption/decryption using symmetric AES CBC cryptosystem with OpenSSL strongly inspired from this implementation :
  https://medium.com/@amit.kulkarni/encrypting-decrypting-a-file-using-openssl-evp-b26e0e4d28d4

4.2 Functions

In this section, we explain the functions implemented for message and image encryption, and for puzzle creation and solution.

4.2.1 main.c

int get_decryption_time(void):
asks the user for the desired time of encryption of the message/image.

long long int get_nb_squaring(int decryption_time):
returns the number of squaring operations needed for the key to be decrypted in the given *decryption_time*.

void generate_n_phi_n(mpz_t n, mpz_t phi_n):
generates a random composite modulus of 2048 bits and puts it in the initialized mpz_t *n*, computes the totient of *n* and puts it in the initialized mpz_t *phi_n* (integers are represented with mpz_t in GMP library).

void message_encryption(void):
method called to do encryption/decryption on a given message.

int image_encryption(void):
method called to do encryption/decryption on an image.

<u>Message encryption</u>

size_t get_plaintext_to_encrypt(char* plaintext):
asks the user to write a message to encrypt in *plaintext* of maximum size 256 characters; returns the number of characters in the given message.

void get_mpz_from_ASCIIstr(char* str, mpz_t mpz, size_t nb_chars):
string to mpz_t integer conversion : gets the integer ASCII representation of the string *str* of *nb_chars* characters and puts the resulting integer in the given *mpz*.

void get_ASCIIstr_from_mpz(char* str, mpz_t mpz, size_t str_nb_chars):
mpz_t integer to string conversion : gets the ASCII representation in the *mpz* integer and puts the corresponding string of length *str_nb_chars* in *str*.

void generate_key(mpz_t key, mpz_t n):
generates a random integer modulo n and puts it in the initialized mpz_t *key*.

void OTP_encrypt(mpz_t key, mpz_t ptext, mpz_t ctext):
encrypts *ptext*, the integer representation of the given plaintext to encrypt, with *key* and puts the ciphertext integer representation in *ctext*. The cryptosystem used is OTP: this is a simple xor operation.

void OTP_decrypt(mpz_t dec_key, mpz_t ctext, mpz_t dtext)
decrypts *ctext* with *dec_key*, the solution of the puzzle, and puts the decrypted text integer representation in *dtext*. This also consists of a simple xor operation.

Image encryption

void generate_key_ofsize(mpz_t key, size_t nb_bits):
generates a random integer of *nb_bits* bits and puts it in the initialized mpz_t *key*.

void encrypt_decrypt_file(unsigned char* key, unsigned char* iv, FILE* input_file, FILE* output_file, unsigned int encrypt):
if *encrypt* is set to 1: encrypts $input\_file$ with given *key* and *iv* using AES CBC encryption.
if *encrypt* is set to 0: decrypts $input\_file$ with given *key* and *iv* using AES CBC decryption.

4.2.2 Squarings.c

void encrypt_key(mpz_t key, long long int nb_squarings, mpz_t a, mpz_t enc_key, mpz_t n, mpz_t phi_n):
encrypts *key* as $key + a^{2^{nb\_squarings}} \mod n$ and puts it in *enc_key*. *phi_n* is used for reducing the exponent $2^{nb\_squarings}$, a is the base used for squaring operations (set to 2, as suggested in the RSW paper [1]).

void e_mod_phi_n(mpz_t e, long long int nb_squarings, mpz_t phi_n):
computes $2^{nb\_squarings} \mod phi\_n$ and puts the result in initialized e.

void b_mod_n(mpz_t b, mpz_t a, mpz_t e, mpz_t n):
computes $a^e \mod n$ and puts the result in b.

void decrypt_key(mpz_t a, long long int nb_squarings, mpz_t n, mpz_t enc_key, mpz_t dec_key):
computes $a^{2^{nb\_squarings}} \mod n$, decrypts *enc_key* as $enc\_key - a^{2^{nb\_squarings}} \mod n$ and puts the result in *dec_key*.

4.3 Workflow

First the user must choose whether to encrypt a message or an image. For the message encryption, the user enters a message of 256 characters maximum. Then they must enter the desired decryption time: the amount of time the message/image will stay encrypted.

4.3.1 Message encryption

1. Convert plaintext into ASCII integer representation. Store integer in an *mpz_t*, integer in the GMP library.
2. Generate an *mpz_t* composite modulus n of 2048 bits and get its *mpz_t* totient.
3. Generate a random *mpz_t* key in the range of 0 to $n-1$ included.
4. We encrypt the plaintext with OTP using the generated key. The corresponding ciphertext is stored as an *mpz_t* integer.
5. Convert the ciphertext into its string representation.
6. Encrypt the key.
7. Print the values of n, a, t (number of squaring operations), ciphertext and encrypted key.
8. Decrypt key with the help of values of step 7.
9. Decrypt the ciphertext with the obtained key.
10. Convert the obtained plaintext into its string representation.

4.3.2 Image encryption

1. Generate a random *mpz_t* key of 256 bits.
2. Convert key into its string representation.
3. Encrypt the image with AES CBC algorithm from Openssl library.
4. Reproduce steps 2, 6, 8 of message encryption.
5. Convert decrypted *mpz_t* key into its string representation.
6. Decrypt image with decrypted key.

4.3.3 Note for image encryption

We want to encrypt a file of size FILE_SIZE. In AES, the BLOCK_SIZE is 16 bytes. AES encrypts inputs with size multiple of BLOCK_SIZE. The padding used in EVP encryption functions of Openssl works as follows:

- If FILE_SIZE = K $*$ BLOCK_SIZE for some integer K, the encrypted file consists of K $*$ BLOCK_SIZE with the encrypted data + 1 BLOCK_SIZE for the padding bytes.

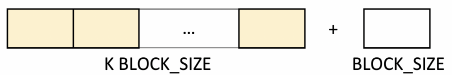

Figure 1: Encrypted file structure for image size multiple of BLOCK_SIZE

- If FILE_SIZE = K $*$ BLOCK_SIZE + R bytes, the encrypted file consists of K $*$ BLOCK_SIZE with the encrypted data + 1 BLOCK_SIZE containing the R remaining bytes and the corresponding padding bytes.

Therefore, the encrypted data will have a size between FILE_SIZE + 1 byte and FILE_SIZE + BLOCK_SIZE. To have sufficient room, the output encrypted file should have a size of FILE_SIZE + BLOCK_SIZE.

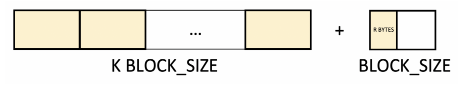

Figure 2: Encrypted file structure for image size not multiple of BLOCK_SIZE

5 Results

The following results have been obtained on a MacBook Pro laptop with a 2.9 GHz Intel Core i7 processor.

5.1 Time as a function of the number of modular squaring operations

The time needed for t squaring operations modulo n is linear in t: $T_t = T_1 * t$, where T_1 is the time needed for one squaring. This is consistent since we reduce all intermediate results modulo n. Without reduction, it would not be linear anymore, but rather exponential.

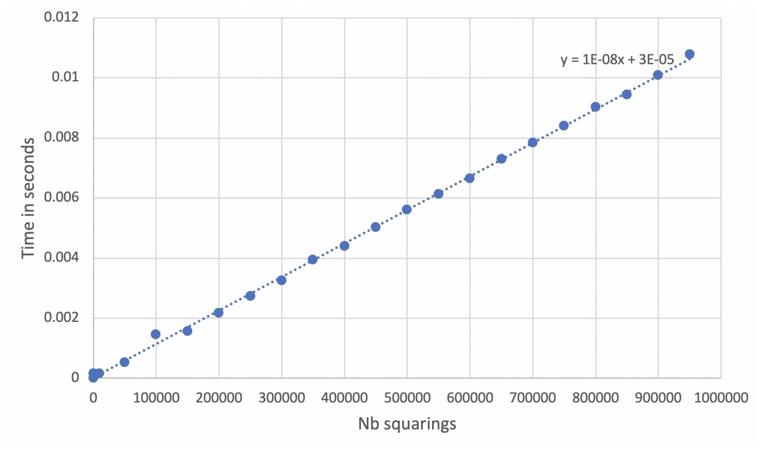

Figure 3: Time, in s, as a function of the number of modular squaring operations, with a fixed modulus size of 2046 bits

.

We chose a size of 2046 bits like size of the modulus in the LCS35 time capsule [2]. With $t = 79685186856218$, we estimated approximately 195 weeks of decryption for this time-lock puzzle, that is almost 4 years.

5.2 Time as a function of the modulus size

For the same amount of squaring operations modulo n, the bigger the modulus size, the more time needed for decryption. This is because the bigger the integer n is, the longer it takes to do a squaring mod n.

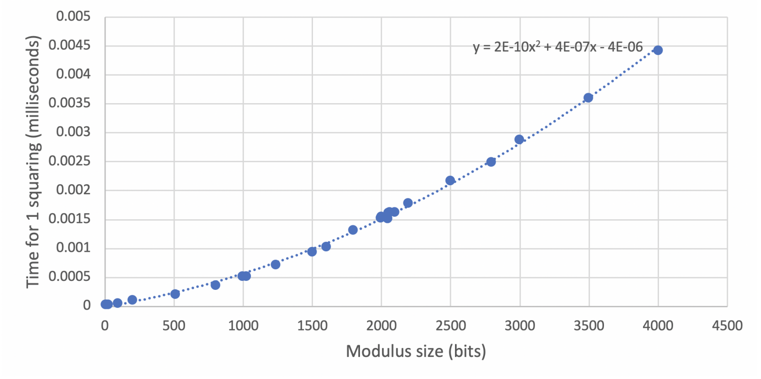

Figure 4: Time, in 10^{-3} s, needed for 1 modular squaring operation $\mod n$

5.3 Concrete example of encryption time and number of squaring operations

Desired decryption time entered: 10 seconds.
Number of squaring operations computed for this amount of time (with 2048 bits modulus): 8814043.
Actual decryption time : 9.845877 seconds.

6 Conclusion

We have implemented the time-lock puzzle suggested by Rivest, Shamir and Wagner [1]. It enables time-release cryptography on messages and larger files such as pictures. We observed that the larger the modulus size, fewer squaring operations needed for the same amount of time of decryption.

Time didn't allow us to fully implement row-by-row decryption of the image. This would make an interesting use-case, and we hope that it would be pursued on a different project.

7 References

[1] R. L. Rivest, A. Shamir and D. A. Wagner. *Time-lock puzzles and timed-release Crypto*. 1996.

[2] R. L. Rivest. *Description of the LCS35 Time Capsule Crypto-Puzzle*. 1999