Implemention of Hash Function and Asymmetric Key Encryption to Prevent Memory Tampering in Game's Anti-Cheat Systems

Zeki Amani - 13524082
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: Zekiamanioke@gmail.com, 13524082@std.stei.itb.ac.id

***Abstract*—Cheating has always been a big problem in the gaming industry. One of the method cheaters uses is by doing memory tampering. To prevent memory tampering, this paper will explore the use of cryptographic techniques such as hash functions and asymmetric key encryption to be implemented in a user-level anti-cheat system with the assumption that we can make a process that is hidden from hackers and hacker can't put breakpoint in our game program.**

Keywords—Anti-cheat, Cryptography, Digital signature, RSA, sha-256, memory tampering, number theory, game

I. Introduction

Video games are the ultimate form of entertainment media. Playing video games gives you an absolute sensation of immersion. It's not surprising to see that in this era, the act of playing video games — commonly called gaming — has been getting more and more mainstream. The gaming industry is inevitably growing so fast these past years. In 2024, the revenue of the worldwide gaming market was estimated to be 455 billion U.S. dollars[6].

While the gaming industry kept growing, there lies a challenging problem game developers had to tackle, and that is cheating. Cheating has become one of the biggest and oldest problems in the gaming industry. This problem takes away the fun and fairness from many games, especially in online competitive games. Not only does it affect the players, but it can also impact the developer's revenue, as it threatens income from in-app purchases. Moreover, the number of players leaving the game can increase significantly if cheaters are left loose. With that, the need for anti-cheat is inevitable.

One of the simplest yet effective methods to cheat is to manipulate — tamper — the game's memory. That's why an anti-cheat system that prevents memory tampering is needed. In this paper, author will discuss about memory tampering as a method of cheating, also how to prevent it by implementing cryptographic techniques such as hash functions and asymmetric key encryption in a user-level anti-cheat system.

II. Theoretical Foundations

Before stepping farther, it's important to understand the theorical foundation behind this implementation.

A. Number Theory

Number theory is a branch of pure mathematics that deals with the study of integers and functions of integers[1]. Which includes the study of prime numbers, greatest common divider, modular arithmetic, and more. These concepts are important to understand the implementation of hash functions and asymmetric key encryption.

1) Greatest Common Divider

The greatest common divisor (gcd) of a and b is the largest integer d such that d divides a and d divides b.

2) Co-prime

Two integers a and b are said to be co-prime if the greatest common divisor of a and b eal to 1

3) Modular arithmetic

Suppose a and m are integers ($0 < r < m.$). Equation (1) gives the remainder r of the operation a divided by m.

$$a \bmod m = r \qquad (1)$$

4) Modular Congruence

Suppose a and b are integers and m is a number > 0, then $a \equiv b \ (mod\ m)$ if and only if m divides $a - b$ means a and b congruence in m

5) Inverse modulo

If a and m are relatively prime and m > 1, then the inverse of $a \ (mod\ m)$ exists. The inverse of $a \ (mod\ m)$ is an integer x such that

$$xa \equiv 1 \ (mod\ m) \qquad (2)$$

6) Prime

A positive integer p (p > 1) is called a prime number if the divisor is only 1 and p[1][2].

B. *Asymmetric Key Encryption*

Asymmetric encryption, also known as public-key cryptography, is a type of encryption that uses a pair of keys to encrypt and decrypt data. The pair of keys includes a public key, which can be shared with anyone, and a private key, which is kept secret by the owner[9].

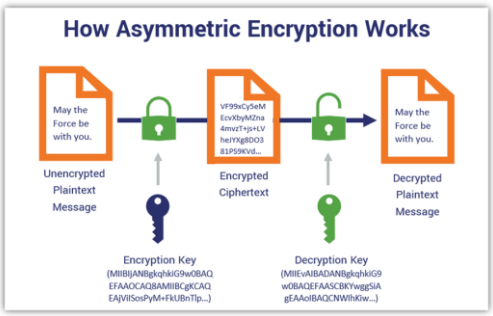

Fig 2.1 How Assymmetric Encryption Works

Source: https://www.thesslstore.com/blog/wp-content/uploads/2020/12/how-asymmetric-encryption-works.png

C. *RSA Encryption*

RSA is the most popular Asymmetric key encryption algorithm. This algorithm was founded by three researchers from MIT (Massachusetts Institute of Technology), namely Ronald Rivest, Adi Shamir, and Leonard Adleman, in 1976. The idea behind this algorithm is the fact that it is very difficult to factorize a large integer into its prime factors. Because of that, it's almost impossible to find the private key even if the public key is known.

Here is the algorithm to produce a pair of keys:

1. Choose pair of two different prime number p and q,
2. Calculate $n = pq$, here n is the modulo that will be used to encrypt and decrypt,
3. Calculate $\phi(n) = (p-1)(q-1)$,
4. Choose an intenger number e that is a coprime of $\phi(n)$. This number will be used as the encryption key,
5. Calculate d as the inverse modulo of $e\ (mod\ \phi(n))$. This number will be used as the decryption key.

Equation (3) here is used to encrypt the data and (4) is used to decrypt the data back to plain text

$$c = m^e\ (mod\ n) \qquad (3)$$
$$m = c^d\ (mod\ n) \qquad (4)$$

Message can be split into smaller blocks if the original message is not just 1 block. [4]

D. *Hashing Function*

Hashing function is an irreversible function that compresses a message (M) of arbitrary size into a string (h) of fixed size. hashing function $H(x)$ have these following properties

1. Collision resistance: it is very difficult to find two inputs a and b such that $H(a) = H(b)$,
2. Preimage resistance: for any output y, it is difficult to find input a such that $H(a) = y$,
3. Second preimage resistance: for any input a and output y = H(a), it is difficult to find a second input b such as $H(b) = y$[3].

E. *SHA-256*

Secure Hashing Algorithm (SHA) is the standard algorithm used for irreversible hash function developed by NIST. SHA-256 belongs to the SHA 2 family, developed to be the successor to SHA-1 because SHA-1 family has already considered unsecure because of brute force attacks[10]. SHA-256 function takes the input of plain text and hash into 256-bit length hash value. Sha-256 is one of the most secure hashing function, that's why it's being used in many applications.

F. *Integrity Check*

One of the most popular application of hashing functions is for integrity check. Integrity check is a validation process to make sure a message or data is not modified. This is because cryptographical hash function is a one-way function[7]. This means if the hashed value is known, it's impossible to find or calculate the actual value. It also utilizes the fact that hash function is very sensitive to change, which make it easier to check if data is being modified[3].

G. *Authenticity Assurance*

Asymmetric key encryption algorithms — like RSA — can be implemented to assure the authenticity of a message. This is because, if encryption key is being used as private key and decryption key is used as public key, only the authorized entities with private key can encrypt a message. Because it is very hard to find the private key if public key is known, then we can assure that the encrypted data being sent is in fact from authorized person.

H. *Digital Signature*

A digital signature is a mathematical technique used to validate the authenticity and integrity of a message, software, or digital document[8]. Using two techniques mentioned before — asymmetric key encryption and hash function — we can achieve this. The digital signature consists of 2 main algorithms

1) Signing

When signing a message, the sender will first make a hash of the message. Then by using private key, the sender encrypts the hashed message. After all that, the sender will send the digital signature and message to the receiver. With this the authenticity of the sender can be assured.

2) Verification

After receiving both the digital signature and the message, the receiver will verify the integrity of the message. This is done by decrypting the digital signature with public key, hashing the message sent, and checking if the decrypted message is the same as the hashed message.

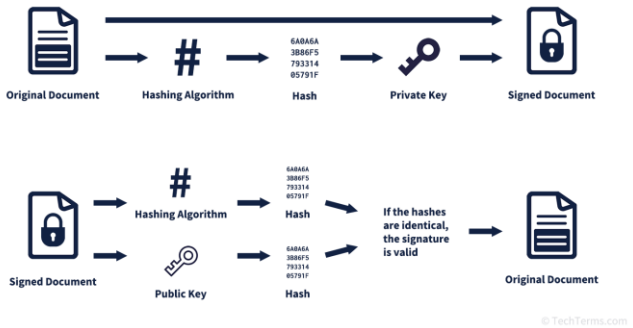

Fig 2.2 How Digital Signature Works

I. *Memory Tampering*

This is where our biggest vulnerability lies. Just like any other computer programs, games have their own memory allocation. in these memories, the program will allocate the value of important things like score, money, and many other things that can affect the gameplay. If hackers manage to modify these important values, unexpected behavior will occur, like changing the current money into the user's desire and many other cases.

J. *Debugging and Memory Scan*

Debugging is the activity to examine a running process. By attaching a debugger to a process, user can examine the program's past, current, and upcoming execution. While debugging, user also can read and write at the memory of the attached process. With that, if the user manages to understand the flow of the program, he can easily change any value the user wanted. But this method is harder to do, because it requires user to understand how the program works.

Another technique that is easier and more commonly used to do memory tampering is by using memory scanners. Just like debugging, user can read and write the currently running programs, but user doesn't even need to understand how the program works. This is how it usually works:

1. enters the target value to memory scanner program,
2. Program sends memory address that contain the entered value,
3. If the program sends multiple memory addresses, then change the current in-game value to a new state, either by decreasing or increasing it. Then do another scan with the new value,
4. Repeat until you are sure that this is the correct memory address.

Besides being easier to use, this method is also harder to detect by anti-debugger. That's why this method is more commonly used, especially by beginner cheaters.

III. Design Considerations

A. *Kernel-level vs user-level anti-cheat*

There's already a lot of anti-tampering methods being used in many anti-cheats with their strengths and weaknesses. One of them is by using kernel-level anti-cheat, which gives the most protection against memory tampering. This is because kernel-level anti-cheat run on kernel which gives it high privilege on process's memory, this also makes it harder for hackers to detect and see. Even though this sounds like the perfect choice, it doesn't come without any risk. The fatal problem with this method is the fact that it literally accesses the computer's kernel directly. This can open a door for hackers to inject malicious malware and viruses. The slightest vulnerability is all it takes for hackers to completely have control over every user of the game. Which is why kernel level anti-cheat has become controversial. As opposed to kernel level anti-cheat, user-level anti-cheat relative easier to hack, because it runs on user-level process, which can be easily detected and seen by hackers. With that being considered, as already mentioned in the introduction, we will implement user-level anti-cheat.

B. *Design Generation and Threat Analysis*

Our main threat for now is debugging and memory scanning. Let's assume that hacker can:

1. Bypass any anti-debugging technique except breakpoint prevention,
2. Understand the program workflow,
3. Find the memory address of any value they wanted,
4. Read and write every memory address.

With that in mind, these are the design ideas the author proposed with short explanations of why it can be used but also its vulnerabilities

1) Simple validation check

This is by comparing our value with a copy of it that is stored in another address. But hacker can just find both memory addresses and change both the values.

2) Integrity check with sha-256

By making a sha-256 hash of the value and scatter the hash values, we can make it harder for hacker to find the sha-256. But, with enough time hacker can understand that it used sha-256 so they can just use the same algorithm to make a new hash value for the new hash value.

3) validation check with symmetric key encryption

by encrypting the value and storing the ciphered value, then check if the value and ciphered value is the same after decoded. With this, the hacker will have to search for addresses of the value, ciphered value, and key. This design would make it difficult for hacker to tamper the memory, but if the hacker can understand the program workflow, then it wouldn't be impossible to find these three addresses and use the same encoding algorithm to change the encrypted value and the value itself, therefore bypassing the validation check.

4) Validation Checks with Digital Signature

By using digital signature, we can assure that only authorized programs with private key — like server, or another separate process — can change the targeted value. If the hacker can't find the private key, then it would be impossible to change the validation value aka the hash value that is used for comparison for integrity check. But of course, if hackers can find the private key, they can do anything they want.

C. *Final Design and Algorithm*

Considering all the strengths and weaknesses and assuming that user can't put breakpoint, the last idea seems to be the most optimal option. That's why we will discuss further on the algorithm and farther the implementation of this design.

As mentioned before, implementing validation checks with digital signature requires a way to hide the private key. This can be done by separating the game into 2 processes: the game itself with the verification algorithm, and another process to make the digital signature. By assuming that hacker won't be able to see the signing process, we can implement this design. Now, we can move on to how the algorithm works.

1. The signer program generates key pairs. The public key will be sent to the main program, and the private key will be kept secret,
2. The main program sends the current targeted value to signer program,
3. The signer program will receive the current targeted value, generate digital signature of it, and send it to the main program,
4. If the target value needs to be checked, the main program will decrypt the digital signature and verify if the current value is the same as the one in digital signature. If the value is invalid, then the program will be topped

IV. Implementation

In this implementation, author used c language to make a simple rock paper scissor game with score system. Let's say that hacker try to change the value of the score and we try to prevent it from being modified. With the proposed design, this is the result implementation. The source code is separated into 4 different files which are "RSA.c", "Hash.c", "Main.c", and "Signer.c".

To simulate server-client communication, in this implementation we will be using 2 program, first program is the main game in the "Main.c" and the second is the digital signature generator in "Signer.c". these two processes will communicate using txt file named "message.txt" which used for sending message from main game process to signer process, and another one named "transfer.txt" which used for sending message from signer process to main game process.

A. *RSA.c*

The "RSA.c" source code contains the RSA key generation, decryption and encryption algorithm.

1) Prime(num)

This function is used for checking if num is a prime or not.

```
// Check if the input number is a prime
number or not
bool prime(ull num){

    if(num == 0 || num == 1){
        return false;
    }
    for(int i = 2; i < num; i++){
        if (num % i == 0 && i != num){
            return false;
        }
    }

    return true;
}
```

2) generate_random_prime(min, max)

Generate random prime number between max and min.

```
// Generate a random prime number between
min and max
ull generate_random_prime(ull min, ull
max){
    ull num;
    do{
        num = (rand() % (max - min + 1)) +
min;
    }while(!prime(num));
    return num;
}
```

3) gcd(num1,num2)

Calculate the greatest common divisor of num1 and num2 using euclidean algorithm.

```
// Find the Greatest Common Divisor between
two numbers
int gcd(ull num1, ull num2){
    ull temp;
    while(num2 > 0){
      temp = num1 % num2;
      num1 = num2;
      num2 = temp;
    }
    return (int)num1;
}
```

4) modInverse(A,M)

used to calculate the inverse modulus of A (mod M)

```
// Calculate the modular inverse of u mod v
// d = (1/e) mod n
ull modInverse(ull A, ull M){
    for (ull X = 1; X < M; X++){
        if (((A % M) * (X % M)) % M == 1){
            return X;
        }
    }
    return -1;  // Inverse doesn't exist
}
```

5) generate_private_key

Generate the private key for the RSA encryption from the input phi. Phi is $\phi(n)$.

```
// generate a private key
// e is the public key, which is coprime to
phi
// e will be used to encrypt the message
ull generate_private_key(ull phi){
    ull key;
    do{
        key = rand() % (phi - 2) + 2;
    }while(gcd(key,phi) != 1 && key > 1 &&
key < phi);
    return key;
}
```

6) power(base,expo,m)

Calculate $base^{expo} (mod\ m)$

```
// The Modular Exponentiation Algorithm
ull power(ull base, ull expo, ull m) {
    long long res = 1;
    long long b = base % m;

    while (expo > 0) {
        if (expo % 2 == 1)
            res = (res * b) % m;
        b = (b * b) % m;
        expo = expo / 2;
    }

    return (res % m);
}
```

7) generate_key_pair

generate encryption key e and decryption key d and the modulo n using RSA algorithm

```
// Generate private(e), public keys(d), and
modulus(n)
void generate_key_pair(ull *n, ull *e, ull
*d) {
    srand(time(NULL));
    ull p, q, phi;
    do {
    // Generate two distinct random prime
numbers p and q
        p = generate_random_prime(100,
999);
        q = generate_random_prime(100,
999);
    } while (p == q);

    // Calculate n
    *n = p * q;
    // Calculate phi(n)
    phi = (p - 1) * (q - 1);
    // Generate private key
    *e = generate_private_key(phi);
    // Calculate public key d
    *d = modInverse(*e, phi);
}
```

8) encode(input data, input e, input n, output enc, output len)

encode the plain data into cipher text with the length of len using RSA encoding algorithm with the encryption key e and moduluo n

```
//encode the data using the private key (e,
n)
void encode(unsigned char *data, ull e, ull
n, ull enc[], int *len){
  int i = 0;
  while(data[i] != '\0' && data[i] !=
'\n'){
    enc[i] = power((ull) data[i], e, n);
    (*len)++;
    i++;
  }
}
```

9) decode(input enc, input d, input n, int len, output dec)

deocde the cypher text enc with the length of len back to it's plain text dec using RSA decryption algorithm with the decryption key d and moduluo n

```
// decode the data using the public key (d,
n)
void decode(ull enc[], ull d, ull n, int
len, ull dec[]){
```

```
  for(int i = 0; i < len; i++){
    dec[i] = power(enc[i], d, n);
  }
}
```

B. Hash.c

The "Hash.c" source code contains the implementation sha256 hashing algorithm. This implementation uses the openssl library to compute the hash value

1. Hash_to_str(data)

This function takes input arbitrary string of data and returns its hash value in the form of string

```
char* Hash_To_Str(char* data) {
    int strLen = strlen(data);
    unsigned char hash[32];
    char* hashStr = malloc(65);
    strcpy(hashStr, "");

    unsigned char str2[32];
    SHA256("hello world",strlen("hello
world"),hash);

    char s[3];
    for (int i = 0; i < 32; i++) {
        sprintf(s, "%02x", hash[i]);
        strcat(hashStr, s);
    }

    return hashStr;
}
```

C. Main.c

The "MAIN.c" source code contains the main game and the validation check. The targeted value "score" is stored as global variable.

```
int score = -1;
```

1) update(win)

This function will be used to update the value of score also to send the new score value to signer process.

```
void update(int win){
    if (win==1){
        score++;
    } else{
        score--;
    }
    FILE *message = fopen("message.txt",
"w");
    fprintf(message, "%d\n", 1);
    fprintf(message, "%d",score);
    fclose(message);
    int check;
    do{
        message = fopen("message.txt",
"r");
        fscanf(message, "%d", &check);
        if (check == 0) {
            fclose(message);
            break;
        }
        fclose(message);
    } while(check==1);
    printf("%d\n",score);
}
```

2) read()

This function will be used to read messages from signer process, and decrypt the message, returning the original hash value.

```
unsigned char *read(){
    unsigned char
*hash=malloc(sizeof(char)*64);
    unsigned long long n, e, d;
    unsigned long long enc[64], dec[64];
    FILE *tranfer = fopen("transfer.txt",
"r");
    if (tranfer == NULL) {
        perror("Error opening file");
        return NULL;
    }
    fscanf(tranfer, "%llu %llu\n", &n, &d);
    for (int i = 0; i < 64; i++)
    {
        fscanf(tranfer, "%llu", &enc[i]);
    }
    fclose(tranfer);
    decode(enc, d, n, 64, dec);
    for (int i = 0; i < 64; i++) {
        sprintf(&hash[i],"%c", (unsigned
char)dec[i]);
    }
    return hash;
    fclose(tranfer);
}
```

3) check()

This function will be used for validity checking and will stop the program if memory tampering happens.

```
void check(){
    char *Check=read();
    char msg[99];
    sprintf(msg,"%d",score);
    unsigned char *hash=Hash_To_Str(msg);
    if(strcmp(Check,hash)){
        printf("original hash =
%s\n",Check);
        printf("changed hash= %s\n",hash);
        printf("data tampering
detected\n");
        exit(0);
    }
    free(hash);
    free(Check);
}
```

4) play()

This function is used to decide if the player wins or not and update the score accordingly.

```
void play(int player) {
    int computer = rand() % 3;
    if (player == computer) {
        printf("Draw\n\n");
    } else if ((player == 0 && computer ==
1) || (player == 1 && computer == 2) ||
(player == 2 && computer == 0)) {
        printf("You lose\n\n");
        update(-1);
    } else {
        printf("You win\n\n");
        update(1);
    }
}
```

5) main()

This is the main gameloop.

```
int main() {
    int choice;
    update(1);
    srand(time(NULL));
    while(1){
        check();
        printf("Your Choice\n(1: Rock, 2:
Paper, 3: Scissors)\n");
        scanf("%d", &choice);
        if (choice==0) {
            break;
        }
        play(choice-1);
        printf("Current Score: %d\n",
score);
    }
    system("pause");
    return 0;
}
```

D. Signer.c

The “Signer.c” source code contains the digital signature generator Program that will be used to make digital signature of the data.

```
unsigned long long n, e, d;
int main(){
    generate_key_pair(&n, &e, &d);
    while(1){
        FILE *message =
fopen("message.txt", "r+");
        char msg[100];
        int check;
        fscanf(message, "%d", &check);
        if (check == 0) {
            fclose(message);
            continue;
        }else if(check==2){
            break;
        }
        fscanf(message, "%s", msg);
        //make hash
        unsigned char
*hash=Hash_To_Str(msg);
        printf("%s\n", hash);

        //encode with rsa
        unsigned long long enc[1000];
        int len = 0;
        encode(hash, e, n, enc, &len);

        //send the encoded message to
transfer.txt
        FILE *tranfer =
fopen("transfer.txt", "w");
        if (tranfer == NULL) {
```

```
            perror("Error opening file");
            return 1;
        }
        fprintf(tranfer, "%llu %llu\n",
n,d);
        for (int i = 0; i < len; i++)
        {
            fprintf(tranfer, "%llu ",
enc[i]);
        }
        fseek(message, 0, SEEK_SET);
        fprintf(message, "%d\n", 0); //to
tell program that the message has been
processed
        fflush(message);
        fclose(tranfer);
        fclose(message);
    }

}
```

V. Result

A. Without Anti-Cheat

Using memory scanner such as cheat engine let us to do memory tampering, once we find the memory address, as demonstrated in fig 5.1, fig 5.2, and fig 5.3

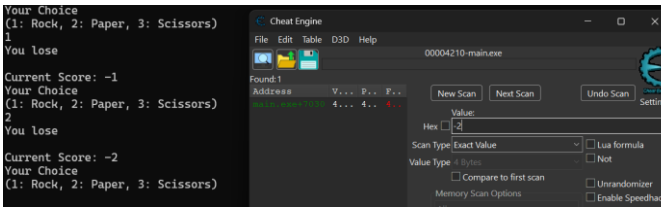

Fig 5.1 finding the memory address

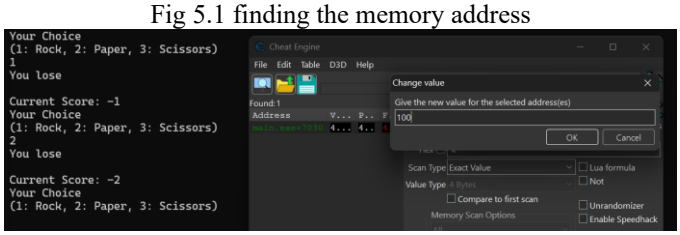

Fig 5.2 changing the target value

```
Your Choice
(1: Rock, 2: Paper, 3: Scissors)
0
Press any key to continue . . .
PS C:\Users\ASUS\Downloads\tes> gcc -o main main.c
PS C:\Users\ASUS\Downloads\tes> ./main
Your Choice
(1: Rock, 2: Paper, 3: Scissors)
1
You lose

Current Score: -1
Your Choice
(1: Rock, 2: Paper, 3: Scissors)
2
You lose

Current Score: -2
Your Choice
(1: Rock, 2: Paper, 3: Scissors)
1
You lose

Current Score: 99
Your Choice
(1: Rock, 2: Paper, 3: Scissors)
```

Cheat Engine
File Edit Table D3D Help
Found: 1
Address V... P.. F..
main.exe+7030 99 4.. 4..

Fig 5.3 value successfully tampered

B. Using Proposed Anti-cheat system

By running signer process, before running the program and did the same momory tampering as before, we obtain the following result as shown in fig 5.4

```
Current Score: 100
original hash= 6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b
changed hash= ad57366865126e55649ecb23ae1d48887544976efea46a48eb5d85a6eeb4d306
data tampering detected
```

Fig 5.6 data tampering detected

VI. Conclusion

cryptographic techniques such as hash functions and asymmetric key encryption indeed can be implemented in a user-level anti-cheat system to prevent memory tampering. but just like any user level anti-cheat, this implementation is not free from vulnerability, if hacker manages to obtain the private key, or if hacker manages to put breakpoint in our program, then it's a game over for our anti cheat system. Therefore, further research will be needed to farther enhance the security of this anti-cheat implementation. This code's implementation can also be implemented in any other games, it just needs several adjustments to the game's source code.

Acknowledgment

Praise be to God Almighty for His grace, the paper with the title "Implementation of Hash Function and Asymmetric Key Encryption to Prevent Memory Tampering in Game's Anti-Cheat Systems" can be completed properly. The author would also like to thank the lecturer of K02 IF1220 - Discrete Mathematics, Arrival Dwi Sentosa, S.Kom., M.T. for the knowledge that has been taught during lectures so that the author can complete this paper smoothly. In addition, the author also wants to thank everybody who has given their support to the author in completing this research paper.

ATTACHMENT

Github link: implementation code's Github link

REFERENCES

[1] R. Munir, "Teori Bilangan (Bagian 1)," *Mata Kuliah Matematika Diskrit*, Institut Teknologi Bandung, 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/15-Teori-Bilangan-Bagian1-2024.pdf

[2] R. Munir, "Deretan, Rekursi, dan Relasi Rekurens (Bagian 2)," *Mata Kuliah Matematika Diskrit*, Institut Teknologi Bandung, 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/11-Deretan,%20rekursi-dan-relasi-rekurens-(Bagian2)-2024.pdf

[3] R. Munir, "Fungsi Hash," *Mata Kuliah Kriptografi*, Institut Teknologi Bandung, 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2023-2024/24-Fungsi-hash-2024.pdf

[4] R. Munir, "Algoritma RSA," *Mata Kuliah Kriptografi*, Institut Teknologi Bandung, 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2023-2024/18-Algoritma-RSA-2024.pdf

[5] R. Munir, "Kriptografi Kunci Publik," *Mata Kuliah Kriptografi*, Institut Teknologi Bandung, 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2023-2024/17-Kriptografi-Kunci-Publik-2024.pdf

[6] Statista, "Video games - statistics & facts," *Statista*, 2024. [Online]. Available: https://www.statista.com/topics/868/video-games/#topicOverview

[7] D. Sharma, "Securing your data: An in-depth look at hashing and integrity verification," *Medium*, Oct. 2023. [Online]. Available: https://deepaksharma2007.medium.com/securing-your-data-an-in-depth-look-at-hashing-and-integrity-verification-c969a4e9d2db

[8] GeeksforGeeks, "RSA Algorithm in Cryptography," *GeeksforGeeks*, [Online]. Available: https://www.geeksforgeeks.org/computer-networks/rsa-algorithm-cryptography/

[9] GeeksforGeeks, "What is Asymmetric Encryption?," *GeeksforGeeks*, [Online]. Available: https://www.geeksforgeeks.org/computer-networks/what-is-asymmetric-encryption/

[10] GeeksforGeeks, "SHA-256 and SHA-3," *GeeksforGeeks*, [Online]. Available: https://www.geeksforgeeks.org/computer-networks/sha-256-and-sha-3/

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Juni 2025

Zeki Amani
13524082