Goorb encode
(A bistoyek R.I.C. Research Project)

Kasra Fouladi

12/14/2024

Contents

0 Introduction

0.1 Acknowledgments

This article was written solely by myself without any external assistance. I am grateful for the attention and support of all my friends and colleagues.

0.2 Importance

In today's world, We face with the increase in data exchange on digital platforms and the rapid advancement of attack methods, supercomputers, and quantum attacks, there is a significant risk that a particular cipher-text could be broken, causing massive damages in the future. The need for strong and efficient encryption and hash algorithms is more critical than ever. [1]

This urgency drives cryptography specialists to make considerable efforts to create more secure encryption and hash algorithms. These advancements aim to ensure that even if hackers gain unauthorized access to information, they cannot easily retrieve the original data. In this article, I examine a new encryption method that uses mathematical and algorithmic techniques to resist various attacks, especially quantum attacks, thereby keeping information secure for future decades.

0.3 Abstract

As I mentioned earlier, one of the primary goals of creating this encryption method is to ensure security against numerous attacks. Therefore, I have invented a new encryption style called **Goorbian encryption**, which utilizes one-way functions, an algorithm as private key and cipher-text bank as public key. This approach offers significant advantages in terms of security, and by employing this style, any encryption algorithm will be immune to many attacks. In the remainder of the article, I will address each of these topics in detail. [2]

This encryption method offers significant flexibility, meaning you can use any one-way function depending on your specific application. In this research I Introduced a messaging protocol using this method named **Raz** and **Goorb post** a messenger using this protocol, two versions of this style, **Goorb encode 2** using the SHA-256 algorithm and **Goorb encode 1** using an algorithm similar to Bubble Shooter in it's structure of one-way function of my encryption. [3] [4]

[1] cipher-text: An encoded text

[2] For more information about public and private keys:
https://ssldragon.com/blog/public-key-cryptography

[3] For more information about SHA-256:
https://networkencyclopedia.com/sha-256-unmasked-deciphering-cryptographic-hash-functions

[4] For more information about Bubble Shooter game:
https://games.skillz.com/guides/bubble-shooter

1 Goorbian encryption

To understand better **Goorbian encryption** mechanism first it's needed to know about this method's two main parts F function and cipher-text bank.

1.1 Function

This method is consists of two main parts: function (Let call it F) and cipher–text bank. First I explain the function part. Imagine a function F from A to B, where A is an infinite set and B is finite, and for any element of B like b there are same order infinite number of elements like a belongs to A and $F(a) = b$. For instance A could be the real numbers set and B could be the set of alphabet letters. This function is clearly one-way. If A is the set of cipher-texts and B is the set of plain-texts, you can encode a same plain-text to infinite number of cipher-texts and this make this method immune against many attacks such as crypt-analysis attacks, statistical-analysis attacks especially machine learning attacks and even quantum attacks. [5]

Here is a very simple C/C++ pseudo-code to better understand the F function:

```
//An example of cipher-blocks (elements of A):
struct cipher_block{
    vector<int> v;
};
//Private key is a part (probably all) of F structure
int key(int num){
    return num % 256;
}
//A simple example of F function (from A to B):
int F(cipher_block a){
    int res = 0;
    for(int i = 0; i < a.v.size(); ++i)
        res += a.v[i];
    int b = key(res) //b belongs to B = {0, 1, ..., 255}
    return b;
}
```

Decoding the encoded objects is straightforward by calculating $F(a)$, where a is a block in the encoded file. However, encoding something is a different journey. Due to the complexity of the F mechanism, you probably cannot directly find an element a in A such that $F(a) = b$ for a particular element b in B. Instead, you need to create a bank of cipher-texts and split them into equivalence categories. In each category, every two cipher-block have the same F result, and no two cipher-block in different categories have the same F result.

[5] In section 4 I examined all of these attacks and show why they are ineffective.

1.2 Bank

As mentioned in the previous paragraph, due to the complexity of F, it is highly improbable to find an element from A whose F result equals a particular element from B on the first attempt. Given that for any element b in B, there are infinitely many elements a in A such that $F(a) = b$, the probability of $F(a)$ equaling any specific element in B is uniformly distributed. Therefore, by testing a large number of elements from B, we can construct a cipher-text bank where for each element in B, there is at least one corresponding element in A such that $F(a) = b$. According to the **Goorbian Probability Distribution**, If n elements are randomly tested from A, it is likely that the bank will be complete. (n is the answer of following equation) [6]

$$\frac{(|B|-1)^n}{|B|^{n-1}} = 0.1$$

Since it's impossible for computers to have infinite memory and it's better to minimize $\frac{|\text{cipher-text}|}{|\text{plain-text}|}$, it's possible to choose cipher-blocks from an extremely huge finite subset of A provided that they are mapped uniformly into elements of B by F.

Here is a C/C++ simple pseudo-code that following previous one to better understand the cipher-block bank concept and decode and encode mechanism in Goorbian encryption:

```
//This code follows the previous one
const int B = 256; //It's the size of B

vector<cipher_block> bank[B];
//Equivalence categories are bank[i] (0 <= i < B)
//For any 2 cipher-blocks in bank[b] like a, F(a) = b

cipher_block random_block(){
    cipher_block res;
    int sz = 10 + rand() % 10; //sz is size of the vector res.v
    //Now there are 2^(32*10) + ---- + 2^(32*19) possible outputs
    for(int i = 0; i < sz; ++i)
        res.v.push_back(rand()); //A random 32-bit integer
    return res;
}
//At first you have to create a bank and it's your public key
void new_bank(){
    for(int i = 0; i < B; ++i)
        bank[i].clear();
    bool mark[B] = {};
    //If mark[i] is false then bank[i] is empty
    int zero = B;
    //Number B's elements like b that bank[b] is empty
    srand(time(nullptr));
    while(zero){
        cipher_block a = random_block();
        int b = F(a);
```

[6] Goorbian Probability Distribution is explained in section 3

```
        if(!mark[b])
            mark[b] = true, --zero;
        bank[b].push_back(a);
    }
}
//You can add some (a, b) to the bank and make it stronger
void make_stronger(int times){
    srand(time(nullptr));
    while(times--){
        cipher_block a = random_block();
        bank[F(a)].push_back(a);
    }
}

void decode(string dir){
    ifstream cipher_text(dir);
    ofstream plain_text(dir + ".decoded");
    cipher_block a;
    int sz; //sz is the size of the vector a.v
    while(cipher_text >> sz){
        a.v.assign(sz);
        for(int i = 0; i < sz; ++i)
             cipher_text >> a.v[i];
        plain_text << (char)F(a);
    }
    cipher_text.close();
    plain_text.close();
}

void encode(string dir){
    ifstream plain_text(dir);
    ofstream cipher_text(dir + ".encoded");
    //The main reason the bank is needed:
    srand(time(nullptr));
    unsigned char c;
    while(plain_text.get(c)){
        int ind = rand() % bank[c].size();
        cipher_text << bank[c][ind].v.size() << ' ';
        for(int i = 0; i < bank[c][ind].v.size(); ++i)
            cipher_text << bank[c][ind].v[i] << ' ';
    }
    cipher_text.close();
    plain_text.close();
}
```

Also it's possible to read the source codes of **Goorb encode 1** and **Goorb encode 2** from links in the **section 5.2** for better understanding how this encryption style works. [7]

[7] You can also view the codes related to this paper from the project's repository.
Link of the project's repository:
https://github.com/bistoyek-official/Goorb-encode

2 Keys

By examining the bank and the F function (or some dynamic parts of F), it can be determined that each of them can serve as a key in this encryption process. The bank could use encryption, while F could handle decryption.

2.1 Pseudo-Asymmetric Property

If attention is paid to their role in encryption and decryption, it can be understood that they are somewhat similar to asymmetric encryption. This is because cipher-text is created by the cipher-text bank and decoded by the F. On the other hand cipher-text banks are obtained from the F, and if the function is expressed as a set of ordered pairs (a, b) where $a \in A$ and $b \in B$ such that $F(a) = b$, then the banks are subsets of this infinite set that meet a certain condition: each member of B appears as the second member in at least one pair. The simultaneous presence of these features made me attribute the **pseudo-asymmetric** attribute to this style. Here are some possible questions that I will try to answer them.

There are infinite banks corresponding to a particular F function, so it's not hard for hackers to guess a bank right?

Not really, If you assume that $|B| = 256$ so the chance of guessing a valid bank like S even if they have A and B is $|B|^{-|S|}$ since we know $|B| \leq |S|$ so the greatest possible value of $|B|^{-|S|}$ is 256^{-256} which is almost zero, It means there is almost no chance for hackers to create a valid bank.

What is the point of having so many choices for banks?

By utilizing this feature, you can assign different keys to n individuals, ensuring that no two people share a common pair. This separation prevents them from discovering F and decoding each other's cipher-texts. Consequently, only you, possessing the F, can decoded all the cipher-texts and identify the sender by reading the cipher-text. In this style F is more like private key and banks are more like public keys they are similar but not completely.

Can hackers regenerate the F from valid banks?

Since F could be any function with required conditions it's nearly impossible for hackers to find any relation between any two pair like (a_1, b_1) and (a_2, b_2) which $a_1, a_2 \in A$ and $b_1, b_2 \in B$ so even if hackers have infinite valid cipher-text bank they can't be a serious threat for the encryption security. To keep the connection more safe I recommend to change the banks after a while.

Can this method use in digital signature?

It can't use in digital signatures, but it is possible to achieve a very secure encrypted communication protocol using these two keys, and if two people have the same F it can be used like symmetric encryptions too.

2.2 Raz

It's not possible to use this encryption in digital signatures because F is required to decode the cipher-texts, but if we make F public the encryption loses it's security, but as i said it is possible to achieve a secure communication protocol which I named it **Raz**. [8]

In this protocol, two users, X and Y, can communicate securely and send any data to each other, ensuring that only they can read the content of the messages. Initially, user X has a cipher-text bank named b_Y and a function F_X, which serves as the key for an Goorbian encryption method. Similarly, user Y has a cipher-text bank named b_X and a function F_Y. The cipher-text bank b_X is generated using the function F_Y, and b_Y is generated using F_X Users can exchange new cipher-text banks or add pairs of cipher-text and plain-text to their existing banks to facilitate secure communication or even change the F function and cipher-text bank at the same time.

Here is a simple C/C++ pseudo code to understand better how Raz works:

```
//In this example we are user X communication with user Y:

const int BX = 256, BY = 256, lim = 20;
//BX and BY are number of possible values of Fx and Fy respectively
//lim is the size limit for by[i]
vector<cipher_block_y> by[BY];

//get_y(f, c) is a function that reads a cipher_block_y c from f
void add_pairs_to_by(ifstream &f){
    cipher_block_y c;
    int ind;
    while(get_y(f, c)){
        f >> ind;
        by[ind].push_back(c);
        if(lim < by[ind].size())
            by[ind].erase(by[ind].begin());
    }//If by[ind] exceeded the limit oldest element should remove
    return;
}
//write_x(f, c) is a function that writes a cipher_block_x c into f
//random_block() is a random generator that returns cipher_blocks_x
void add_pairs_to_bx(ofstream &f, int n){
    cipher_block_x c;
    srand(time(nullptr));
    while(n--){
        c = random_block();
        write_x(f, c);
        f << Fx(c) << '\n';
    }
}

void change_by(ifstream &f){
    for(int i = 0; i < BY; ++i)
        by[i].clear();
```

[8] Raz/raaz/: A Persian word means secret.

```
    cipher_block_y c;
    int ind;
    while(get_y(f, c)){
        f >> ind;
        if(by[ind].size() < lim)
            by[ind].push_back(c);
    }
}

void change_bx(ofstream &f){
    int cnt[BX] = {};
    cipher_block_x c;
    srand(time(nullptr));
    while(*min_element(cnt, cnt + BX) == 0){
        c = random_block();
        int res = Fx(c);
        write_x(f, c);
        f << res << '\n';
        ++cnt[res];
    }
}
//write_y(f, c) is a function that writes a cipher_block_y c into f
void encrypt_message(vector<int> pt, ofstream &f){
    srand(time(nullptr));
    //pt vector is the plain-text
    for(int i = 0; i < pt.size(); ++i)
        write_y(f, by[pt[i]][rand() % by[pt[i]].size()]);
}
//get_x(f, c) is a function that reads a cipher_block_x c from f
vector<int> decrypt_message(ifstream &f){
    vector<int> res;
    cipher_block_x c;
    while(get_x(f, c))
        res.push_back(Fx(c));
    return res;
}
```

This messaging protocol is used in a messenger named **Goorb post** and you can download and read it using links in source codes section or checkout Goorb-encode repository in bistoyek-official github account. I wish I could explain more but due to sensitive legal aspects, I do not provide an explanation about how to work with this messenger. [9]

Attention:
This messaging application employs some of the most advanced encryption technologies, rendering it virtually unbreakable by any conventional or quantum computing systems. Despite the robust security measures, the developer assumes no legal responsibility for any misuse or legal infringements by users or third parties. Users are solely responsible for the lawful use of this messaging application and any consequences arising therefrom.

[9] Link of the repository:
https://github.com/bistoyek-official/Goorb-encode

3 Goorbian Probability Distribution

3.1 Explanation

To check how long it will takes for the bank to be created or become as strong as the user wants, we need to look at the issue from a probabilistic perspective, so I made a personalized version of the **Markov process** and **Random walk problem** named **Goorbian Probability Distribution**, In this distribution there are n $(1 < n)$ players doing a game and name of one of them is **Goorba**! At first all of them are in coordinate $x = 0$ in every turn a random player will chosen and the player will move one unit into right the game continues for k turns, this distribution's random variable is about Goorba's position in x border at the end of the k^{th} turn. [10] [11]

$$I \sim GOORB(n, k)$$

To examine how Goorba's place in the x border will updates he can write 1 if he moved one unit to right, and write 0 the otherwise. Now suppose s is a random string of his moves, in the string s, i^{th} number is 1 with chance of $\frac{1}{n}$ and it's 0 with chance of $\frac{n-1}{n}$. So if a particular string has O ones and Z zeros the chance of it's happening is:

$$O + Z = k,\ (\frac{1}{n})^O(\frac{n-1}{n})^Z$$

And clearly the $P(I = i)$ is equal to the sum of all chance of strings which have i ones so it's needed to know how many strings exist which satisfies this condition and it's similar to the problem of coefficient of monomials in expressions, for a better understanding, consider the following example:

$$0, 1, 1, ..., 0 \ \rightarrow\ (\frac{1}{n} + \frac{\mathbf{n-1}}{\mathbf{n}})(\frac{\mathbf{1}}{\mathbf{n}} + \frac{n-1}{n})(\frac{\mathbf{1}}{\mathbf{n}} + \frac{n-1}{n})...(\frac{1}{n} + \frac{\mathbf{n-1}}{\mathbf{n}})$$

Now we know what is $f_I(i)$ but there are two other useful concepts named $e(i)$ and evacuation function $evac(i, \epsilon)$, $e(i)$ is expected value of players in $x = i$ after the last turn and since all players have symmetry because they will chosen completely random so $e(i)$ equals $nf_I(i)$. $evac(i, \epsilon)$ is the value of k that satisfies $e(i) \leq \epsilon$ and $k - 1$ doesn't or $k = 0$ and any $k < k'$ satisfies it too. It could be shown the answer always exists. (In small values of ϵ it could be use as the expected value of k that $x \leq i$ is empty e.g. $n = 256$, $evac(0, 0.1) = 2006$) [12]

$$P(I = i) = f_I(i) = \binom{k}{i}\frac{(n-1)^{k-i}}{n^k},\ e(i) = nf_I(i) = \binom{k}{i}\frac{(n-1)^{k-i}}{n^{k-1}}$$

$$evac(i, \epsilon) = k \mid ((\forall k' \geq k \rightarrow e(i) \leq \epsilon)\ \wedge\ (k = 0 \vee (k' = k - 1 \rightarrow \epsilon < e(i))))$$

[10] For more information about Markov Process:
https://web.stanford.edu/class/cme241/lecture_slides/david_silver_slides/MDP.pdf

[11] For more information about Random Walk problem:
https://ocw.mit.edu/courses/6-042j-mathematics-for-computer-science-fall-2010/eb5072ea046760bb81e5ce6bc1c97b37_MIT6_042JF10_chap20.pdf

[12] It's same equation to what wrote in page 3.

For a better understanding, you can use the following link to check the approximate graphs of $f_I(i)$ and $e(i)$ with different values of n and k and also by using these graphs, it's possible to better understand the $evac(i, \epsilon)$ function. And since it's not easy to calculate $evac(i, \epsilon)$. (However it could be shown that it's value's complexity is from $O(n(i+1)lg(\frac{n(i+1)}{min\{\epsilon, \frac{1}{2}\}}))$.) I created a calculator to calculate it in $O(lg(n) + lg(i+1) + lg(lg(\frac{1}{min\{\epsilon, \frac{1}{2}\}})))$ the only limitation it has is since numbers are bounded in computers it can't work correctly for very large values of n and i and very small or large values of ϵ but theoretically it can calculate any $evac(i, \epsilon)$ and it's source code and it's proof of correctness could be read and downloaded from the **Goorbian Probability Distribution** directory in the project's repository. [13] [14]

Link of the visualized diagrams:
https://www.desmos.com/calculator/oh6mmliiik

Link of the project's repository:
https://github.com/bistoyek-official/Goorb-encode

3.2 It's Applications

Now suppose the number of the players is $|B|$ and when a random and necessarily new a chosen and $F(a) = b$ the size of $bank_b$ will increase by one and we can move the corresponded player to b one unit into the right, since creating a is random there is no difference with the past version of the game from probabilistic view, and even it's possible to create a bank by parallel processing it with p processors it would be created p times faster than the single processor case. And know we know the answer of the first question was asked in this section.

Note: In entire this process is assumed to produce a different random value from the generated values each time, given the huge size of the set of possible cipher-blocks and **Birthday problem** that shows it's needed to use the random generator $O(\sqrt{N})$ times to have good chance of collision, with a chance approximately equal to one collision never happens. (N is number of possible cipher-blocks that the *random_block* function can make.) [15]

Also, if the path traveled by each person can affect the probability of being chosen, this distribution can be used in many problems.

[13] This visualization was created using the website Desmos.com to read more about it:
https://www.desmos.com/about

[14] In this approximated functions used a formula for calculating $ln(x!)$ called Stirling's Formula to read more about it:
https://math.mit.edu/~rmd/440/stirlingformula.pdf

[15] For more information about Birthday problem:
https://www.oxfordreference.com/display/10.1093/oi/authority.20110803095508254

4 In The Face Of Attacks

In this section, I'll examine the resilience and robustness of the Goorbian encryption method against various types of attacks. My focus is on two distinct versions of this method: **Goorb encode 1** and **Goorb encode 2**. The reason for examining two versions is that, in this method, the key is not a simple number but some parts of an irreversible function, named the F function. Since the only necessary condition for the F function is that it achieves an uniform distribution when mapping an infinite set to a finite set, analyzing and attacking an unknown, complex function such as this presents unique challenges. Additionally, the structure of F and the cipher blocks can vary in complexity, from something as simple as $F(x) = x \, mod \, 2$, where x is an integer, to more intricate and hard-to-guess functions.

It can be shown that there are some possible cases of F which crypt analysis and statistical analysis attacks are totally ineffective and even quantum attacks are ineffective so it's nearly impossible for attackers to decrypt a random cipher-text with desired accuracy, e.g. AES is anti quantum attacks and it's possible to define series of functions that $F_i : \{0,1\}^{128} \rightarrow \{0,1\}$ that $F_i(x) = [i^{th} \text{ bit of } AES_{128 \rightarrow 128}(x \, mod \, 2^{128})]$ $(1 \leq i \leq 128)$ then there is at least one of these functions are immune against these attacks because if it was possible to guess all the functions outputs with desired accuracy then due to the production principal if someone does these attacks and with goal that achieve accuracy greater or equal to 0.9999 then attackers can break the AES because they can merge $F_i(x) \, (1 \leq i \leq 128)$ and guess the plain-text with accuracy equal to $0.9999^{128} \approx 0.987$ but given that AES cryptography is anti quantum attacks it's impossible. Therefore, I decided to subject these two versions to various tests. In two scenarios, In one it's known the structure of the function F besides the key (Let call it $F-$key), while in other scenario, there is no clue about the encryption at all. [16] [17] [18] [19] [20]

4.1 No clue scenario

Imagine there is no clue about cryptography but it is guaranteed that the key(s) of it won't change in the all of the attack time (clearly now it's an easier problem), and there is an oracle that could answer these queries:

1. for a such $a \in A$ what is $F(a)$. (costs x)
2. for b that $b \in B$ return a random $a \in A$ which $F(a) = b$. (costs y)

[16] For more information about crypt analysis attacks:
https://www.geeksforgeeks.org/cryptanalysis-and-types-of-attacks/

[17] For more information about statistical analysis attacks:
https://link.springer.com/chapter/10.1007/978-3-030-95161-0_5

[18] For more information about quantum attacks:
https://coingape.com/everything-you-need-to-know-about-quantum-attacks/

[19] For more information about AES:
https://www.geeksforgeeks.org/advanced-encryption-standard-aes/

[20] For more information about post quantum (anti quantum attacks) cryptography:
https://www.nist.gov/cybersecurity/what-post-quantum-cryptography

Using this two queries there is possible to do any crypt-analysis and statistical–analysis attacks, and since we have no data about the F function a combined attack with one of the crypt-analysis attacks and machine learning attacks (machine learning attacks are in statistical attacks category) could be a nice try, so I design some of these attacks and attacked to these two versions to test their security of each key, source codes of attacks are available in project's repository in **Goorb Lab** directory and, here are the results of the tests. [21]

Using the **Goorb Lab**, my simple cryptography lab, nine different combined attacks were conducted on the **Goorb encode 1** and **Goorb encode 2** algorithms, which were set with **key-1** and **key-0**, respectively. Each of these attacks was repeated under various conditions and states to evaluate the actual performance of the cryptography algorithms against diverse threats. Below, using the provided links, the performance of the encryptions against these attacks can be observed: (For readers using the printed version who cannot access the hyperlinks, the performance results can be accessed at: https://github.com/bistoyek-official/Goorb-encode/tree/main/Goorb%20Lab/Encryption/Performance)

Goorb encode 1 performance:

NN-Linear	**NN-Differential**	**NN-Known-plain-text**
results	results	results
CNN-Linear	**CNN-Differential**	**CNN-Known-plain-text**
results	results	results
ANN-Linear	**ANN-Differential**	**ANN-Known-plain-text**
results	results	results

Goorb encode 2 performance:

NN-Linear	**NN-Differential**	**NN-Known-plain-text**
results	results	results
CNN-Linear	**CNN-Differential**	**CNN-Known-plain-text**
results	results	results
ANN-Linear	**ANN-Differential**	**ANN-Known-plain-text**
results	results	results

[21] **Goorb Lab**'s source code is available in **Goorb Lab** directory in the project's repository, this attack simulator can help people to test their encryption algorithms or their attack algorithms much easier.
Link of the Goorb Lab's source code:
https://github.com/bistoyek-official/Goorb-encode/tree/main/Goorb%20Lab
Disclaimer: This project developed by **bistoyek R.I.C.** is intended solely for educational and research purposes. It should not be used for any malicious or illegal activities. **Goorb Lab** and it's Author are not responsible for any misuse of the information provided or any consequences resulting from it's application.

4.2 Known $F-$key scenario

In the previous section, it was demonstrated that even with a fixed key, there exist algorithms based on this method that exhibit strong resistance against relatively powerful attacks. This indicates that such methods maintain high security even with a fixed key. It will now be shown that there exist keys that can transform these algorithms into post-quantum algorithms, even if the underlying function $F-$key is not post-quantum secure.

For example, in cases where the entire dynamic part of F (the key), is at the end of the structure of F, and $F-$key is a function from A to $S = \{0,1\}^k$ which $256 \leq k$ and $2^k \, mod \, |B| = 0$, a key could be adjusted such that our encryption becomes post-quantum secure, given that $F-$key uniformly distributes A over the set S.

One of the ways to construct a key that makes the encryption post-quantum is to use the following unsolved problem:

In the context of functions $g : \{0,1\}^{256} \to \{0,1\}$ where the output is the XOR of bits from a hash function based on the SHA-256 algorithm applied to the input, it is known that no classical or quantum algorithm currently has been discovered that can determine the exact form of such functions merely from their input-output pairs, due to their inherent complexity and the principles of cryptographic hashes security.

Considering the infinite number of such functions g that exist (Because there is infinite number of cryptographic secure hashes based on SHA-256), if $|B| = 2^l$ ($l \leq 256$), it's possible to utilize this unsolved problem to construct keys that meet the condition. Specifically, it's possible to choose l functions like g and set the key like below:

$$F(a) = \text{key}(x) = \sum_{i=0}^{l-1} g_i(x \, mod \, 2^{256}) \cdot 2^i$$

$$\text{where } (F - \text{key})(a) = x \text{ and } x \in S$$

Note: To obtain a desirable result, for any two $g_i(x)$ and $g_j(x)$ $(0 \leq i < j < l)$ the probability that the output is 1 for a random x must be independent events, and for each function in this functional series the probability that the output equals 1 is 50%. For the case $l = 1$, the creation of such series is obvious, e.g. one of the ways is $g_0(x) =$ XOR of all bits of SHA-256(x).

So even if the structure of $F-$key is known to a hacker, keys can still be created for a set of $F-$key structures that make the encryption post-quantum secure. Therefore, even with the knowledge of the $F-$key structure, the security of the encryption cannot be threatened by a hacker.

5 Conclusion

5.1 Goorbian encryption's Applications

In **Goorb encode** project, I introduced and evaluated a novel encryption method called **Goorbian encryption**. One of the key aspects of this method is the security of the encryption banks and the ability to approximate the cost of constructing cipher-texts banks using **Goorbian Probability Distribution**. This analysis helped us better understand the computational cost required to achieve the desired state for a particular encryption scheme.

The **pseudo-asymmetric** property of keys in **Goorbian encryption** allows us to achieve the **Raz** encryption protocol. This protocol significantly enhanced the security of communications. The developed messaging application, **Goorb post**, provided a secure and dynamic environment for data exchange where users could easily change their encryption keys and algorithms.

In the simple cryptography laboratory, **Goorb Lab**, I specifically examined the security of two encryption algorithms, **Goorb encode 1** and **Goorb encode 2**, that utilized this new style of encryption. These tests demonstrated that the **Goorbian encryption** method can provide significant security.

The applications of **Goorbian encryption** highlight its high potential and can serve as a foundation for future studies and developments in the field of information security and cryptography. This project not only presented a new and effective method in cryptography but also provided practical protocols and tools for real-world applications. It was also shown that many F functions in encryption can be equipped with post-quantum keys.

Suggestions for Future Researches:

1. **Development and Optimization of Algorithms:**
   - Develop and improve the performance and efficiency of encryption algorithms based on Goorbian encryption.
   - Create and optimize powerful and efficient F functions and keys.
2. **Industrial and Commercial Applications:**
   - Assess practical applications of Goorbian encryption in various industries such as cybersecurity, cryptocurrencies, and telecommunications.
   - Study real-world use cases and implement this method in existing systems.
3. **Examination and Analysis of Emerging Threats:**
   - Study and evaluate emerging threats and propose solutions to counter them.
   - Analyze the impact and potential outcomes of new threats on the security of Goorbian encryption.

5.2 Source Codes

The following section provides links to the source codes related to our project. These repositories contain the implementation details of the **Goorbian encryption** methods and associated tools. You can explore and analyze the code to gain a deeper understanding of the algorithms and their functionalities:

- **Goorb encode 1:**
  https://github.com/bistoyek-official/Goorb-encode/tree/main/Goorb%20encode%201
- **Goorb encode 2:**
  https://github.com/bistoyek-official/Goorb-encode/tree/main/Goorb%20encode%202
- **Goorb post:**
  https://github.com/bistoyek-official/Goorb-encode/tree/main/Goorb%20post
- **Goorb Lab:**
  https://github.com/bistoyek-official/Goorb-encode/tree/main/Goorb%20Lab

Link of **Goorb encode** project's repository:
https://github.com/bistoyek-official/Goorb-encode

6 References

[1] https://ssldragon.com/blog/public-key-cryptography

[2] https://networkencyclopedia.com/sha-256-unmasked-deciphering-cryptographic-hash-functions

[3] https://games.skillz.com/guides/bubble-shooter

[4] https://github.com/bistoyek-official/Goorb-encode

[5] https://web.stanford.edu/class/cme241/lecture_slides/david_silver_slides/MDP.pdf

[6] https://ocw.mit.edu/courses/6-042j-mathematics-for-computer-science-fall-2010/eb5072ea046760bb81e5ce6bc1c97b37_MIT6_042JF10_chap20.pdf

[7] https://www.desmos.com/about

[8] https://math.mit.edu/∼rmd/440/stirlingformula.pdf

[9] https://www.oxfordreference.com/display/10.1093/oi/authority.20110803092254

[10] https://www.geeksforgeeks.org/cryptanalysis-and-types-of-attacks/

[11] https://link.springer.com/chapter/10.1007/978-3-030-95161-0_5

[12] https://coingape.com/everything-you-need-to-know-about-quantum-attacks/

[13] https://www.geeksforgeeks.org/advanced-encryption-standard-aes/

[14] https://www.nist.gov/cybersecurity/what-post-quantum-cryptography