 doi:10.20944/preprints202209.0295.v1

XOR Chain and Perfect Encryption at the Dawn of the Quantum Era

Luis Adrián Lizama-Perez[1,*[0000−0001−5109−2927]]

[1] Universidad Técnica Federico Santa María, Av. Vicuña Mackenna 3939, San Joaquín, Santiago, Chile; luis.lizamap@usm.cl

Abstract. In this work we present a new algorithm that achieves the perfect Shannon secret by means of the XOR function and a method that we call multiple key reuse. The algorithm has two execution modes: message authentication and data encryption. The XOR encryption scheme allows for batch encryption and exhibits Perfect Forward Secrecy (PFS). Furthermore, based on our fundamental algorithm, we have developed a new strategy for blockchain implementation that does not require Proof of Work (PoW), but defines a fair mechanism for miner selection and secure addition of blocks to the chain.
Since our method is mainly based on the Boolean XOR function, the strength of the cryptosystem can be directly established thanks to its mathematical properties. Due to the risk that quantum computers represent for current cryptosystems based on prime factorization or discrete logarithm, we postulate that our method represents a promising alternative in the quantum era for the security of communications between Internet of Things devices as well as Blockchain technology.

Keywords: Authentication; Encryption; Blockchain

1 Introduction

Today's global economy is enhanced by digital transactions executed through the Internet, which must be guaranteed by cryptographic services for data protection and authentication. In this scenario, public key cryptography makes user authentication possible, as well as digital signature and Blockchain protocols.

On the other side, a debate has arisen regarding the installation of backdoors in mobile application software which would allow encrypted conversations to be revealed and prevent cybercrime. It is known that in the future, homomorphic encryption will allow the processing of encrypted data without requiring its decryption and thus keep the conversations confidential. Unfortunately, homomorphic software is still in the research stage.

In this paper we will discuss several methods which we hope will further contribute to homomorphic software development, specifically to obtain homomorphic traceability. Furthermore, since our method is based on the properties of the XOR function, we claim that in the quantum era the security of the algorithm is guaranteed.

1.1 Perfect Secrecy

In the context of the mathematical theory of secret communication, as it was stated by Claude Shannon [1], it is known that the XOR logical operation is a mathematical function that allows

 doi:10.20944/preprints202209.0295.v1

perfect encryption, at least in theory, since up to this date there are no cryptosystems that operate using just the XOR encryption function.

A ciphertext exhibits perfect secrecy if the attacker's knowledge about the content of the message is the same before and after the adversary inspects the ciphertext, even though the attacker can apply unlimited computational capacity. This implies that the encrypted message does not provide the adversary with any precise information about the content of the cleartext message. This is because according to the communication theory [1, 2], to achieve perfect secrecy the encryption key must be the same size as the cleartext message. Unfortunately, there is a serious challenge to establish an arbitrarily large secret key between Alice and Bob, the parties seeking the private communication.

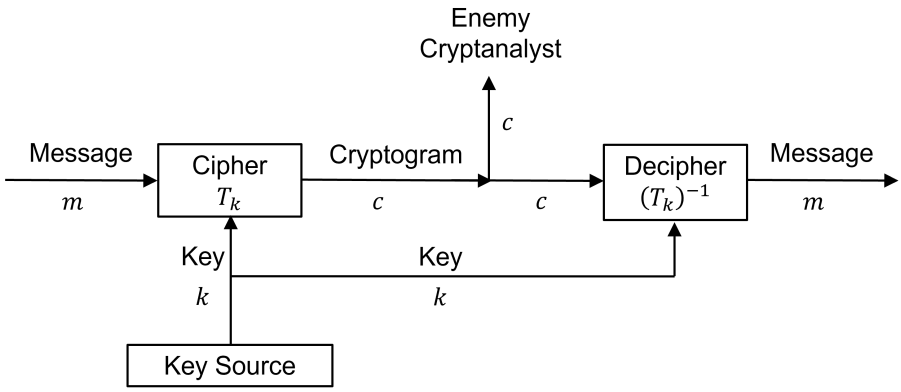

Fig. 1: The general scheme of Shannon cipher.

As long as the encryption key is at least the same size as the cleartext, the XOR function is sufficient to guarantee the confidentiality of the message. This is explained because the XOR is the only reversible Boolean operation, since $k \oplus k = 0$ the ciphering relations are written below, thus $c \oplus k = m \oplus k \oplus k = m$ where k represents the secret key and m denotes the cleartext:

— $m \oplus k = c$
— $c \oplus k = m$

Furthermore, given a ciphertext c, it could be derived from any arbitrary m and any possible k, so that there are many possible keys as there are possible ciphertexts. This property is known as perfect secrecy [3]. However, the XOR encryption system prohibits the reuse of the secret key because messages can be subjected to statistical attack since they do not contain random bits as is the case with secret keys.

In this work we will demonstrate that in order to obtain perfect secrecy it is not strictly necessary to establish the entire secret key beforehand, but rather we propose to generate and reuse multiple previously interleaved keys. However, another alternative but not functional in practice could be the use of a simple key-renewal scheme, as it is written below, because it requires the continuous establishment of a secret key between remote users through the public channel:

 doi:10.20944/preprints202209.0295.v1

— $m_1 \oplus k_1 = c_1$
— $m_2 \oplus k_2 = c_2$
— $m_3 \oplus k_3 = c_3 \ldots$
— $m_r \oplus k_r = c_r$

We will return to this point later soon, where we will sketch the idea of multiple key reuse.

1.2 XOR Indistinguishability

Perfect secrecy can be equivalently established by asserting that two pairs of random binary strings producing the same ciphertext cannot be algorithmically separated from each other because the XOR function exhibits indistinguishability between the strings. Suppose we have x_a and y_a such that $h = x_a \oplus y_a$ but we find another pair x_r and y_r that also produces h, as indicated below. We say that the pairs (x_a, y_a) and (x_r, y_r) are indistinguishable from each other because they both produce the same h. Therefore, an eavesdropper cannot determine the original components x_a and y_a because there are an arbitrary pairs of strings that produce h likewise:

— $h = x_a \oplus y_a = x_r \oplus y_r$

In other words, due to indistinguishability, given h there is no algorithm to derive (x_a, y_a). We will call to this computational problem the separability problem, and therefore cryptosystems based on it are post-quantum because it will suffice to use binary strings large enough to demand the exhaustive search of Grover's quantum algorithm [4]. In this document, we refer to being indistinguishable as the inseparability property of the XOR function.

We should note that if the user authentication would be supported just on verifying h, the inseparability property becomes irrelevant, since it will suffice for the attacker to generate a random string x_e, then she computes $h \oplus x_e$ to get y_e so that $h = x_e \oplus y_e$.

1.3 Stream Cipher

A stream cipher is a symmetric-key cipher in which a pseudo-random encryption stream digit is combined with a data stream binary digit, that is, one byte at a time. Stream ciphers are mainly used when speed and simplicity of the system are prioritized. Next we will briefly describe the most outstanding algorithms.

— HC-256 generates keystream from a 256-bit secret key and a 256-bit initialization vector [5]. It has been designed to provide bulk encryption in software at high speeds while permitting strong confidence in its security.
— Grain is based on two shift registers and a nonlinear output function. The key size is 80 bits and no attack faster than exhaustive key search has been identified [6].
— Rabbit is based on iterating a set of coupled non-linear functions. Rabbit is characterized by a high performance in software [7].
— Mutual Irregular Clocking Keystream generator (MICKEY) uses irregular clocking of shift registers, with some novel techniques to balance the need for guarantees on period and pseudo-randomness against the need to avoid certain cryptanalytic attacks [8].
— The core of Salsa20 is a hash function with 64-byte input and 64-byte output. The hash function is used in counter mode as a stream cipher: Salsa20 encrypts a 64-byte block of plaintext by hashing the key, nonce, and block number and xor'ing the result with the plaintext [9].

— Trivium, a stream cipher inspired by block cipher design principles that replaces the building blocks used in block ciphers by equivalent stream cipher components [10].
— Sosemanuk is key length is variable between 128 and 256 bits. It accommodates a 128-bit initial value. Any key length is claimed to achieve 128-bit security. The Sosemanuk cipher uses both some basic design principles from the stream cipher SNOW [11] and some transformations derived from the block cipher Serpent [12].

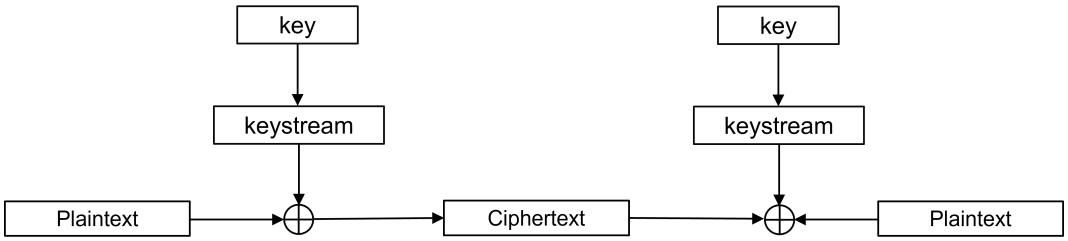

Fig. 2: The main components of the stream cipher.

1.4 Proposal of Our Approach

In previous works, we have developed digital signature methods based on hash functions [13] and HMAC [14]. This time we will base our algorithms on the properties of the XOR boolean function. We claim that our method can be used to construct the following cryptographic primitives, then we will proceed to explain how we achieved them.

— **Message Authentication:** It is infeasible for the eavesdropper to impersonate a message as coming from another one. The algorithm makes it possible to verify that the current message is chained to all previous messages starting from the user's identification message. To be discussed in section 2.
— **Message Encryption:** Message are encrypted so that it is infeasible for the eavesdropper to impersonate a message. To be discussed in section 3.
— **XOR Chain:** The authentication model has allowed us to define a proof-of-work technique and a new approach to blockchain implementation that we have called XOR chain. For the conceptualization of the XOR chain, we are going to first introduce a game based on hash functions that we have named Crypto Bingo. This topic is covered in section 4.

Section 3 also contains a description of an attack scenario and other important properties of the encryption method: Perfect Forward Secrecy (PFS), Group Encrypted Communication and Batch Encryption.

2 Message XOR Authentication

Let three binary strings be k_0, k_1 and k_2. We derive x_{01}, x_{02} and x_{12} by performing the following operations $x_{01} = k_0 \oplus k_1$, $x_{02} = k_0 \oplus k_2$ and $x_{12} = k_1 \oplus k_2$. The additive group $\mathbb{Z}_{\oplus}$ is defined by the numbers x_{01}, x_{02} and x_{12} because the following operations can be directly verified:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0295.v1

$$x_{01} \oplus x_{02} = x_{12}$$
$$x_{02} \oplus x_{12} = x_{01}$$
$$x_{01} \oplus x_{12} = x_{02}$$

If we define a random variable $k_0{}'$ and compute $k_1{}'$ such that $k_0{}' \oplus k_1{}' = x_{01}$. Then $k_2{}'$ is chosen such that $k_0{}' \oplus k_2{}' = x_{02}$. Now, we want to find out if $k_1{}' \oplus k_2{}' = x_{12}$. To establish it, we know that the numbers x_{01} and x_{02} imply the existence of x_{12} due to the closure of the additive group. Since $k_0{}'$, $k_1{}'$ and $k_2{}'$ define the same group $\mathbb{Z}_{\oplus}$, we have $k_1{}' \oplus k_2{}' = x_{12}$. Since $k_0{}'$ was defined randomly, this implies that for every chain $k_0{}'$ exist $k_1{}'$ and $k_2{}'$ that define the same group $\mathbb{Z}_{\oplus}$. Suppose Alice has defined k_0, k_1 and k_2 that she uses to compute x_{01}, x_{02} and x_{12}. If Alice keeps hidden k_0, k_1, and k_2 but she publicly shares x_{01}, x_{02}, and x_{12}, an adversary can derive another set of numbers $k_0{}'$, $k_1{}'$ and $k_2{}'$ which return x_{01}, x_{02} and x_{12} as well. However, due to the indistinguishability principle, Eve cannot derive the original Alice's set k_0, k_1, and k_2. Thus, we can establish an advantage for Alice which leads us to derive our authentication model.

2.1 XOR Authentication Algorithm

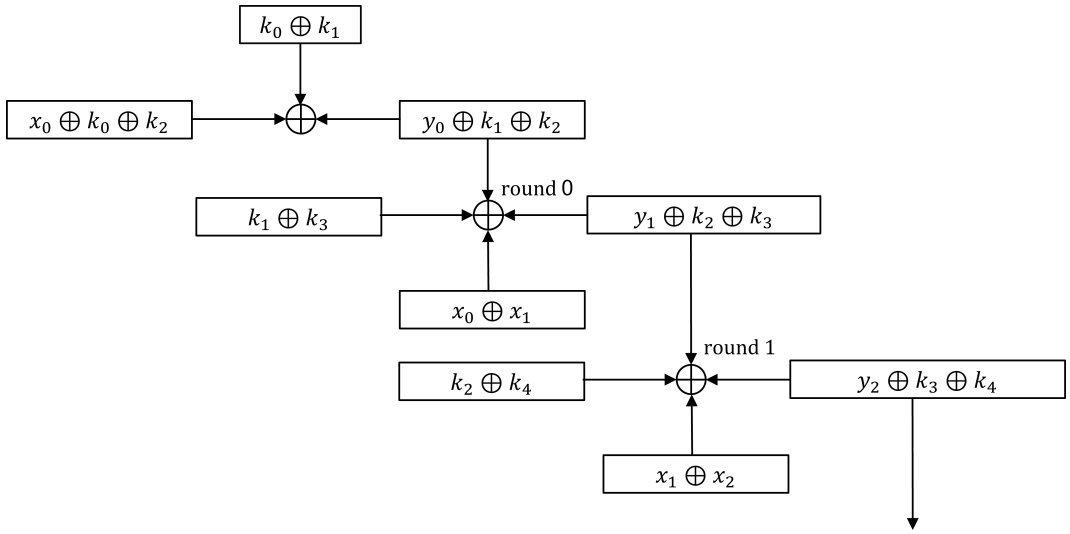

Fig. 3: Bob's process diagram for authenticating Alice.

Now, we will introduce the XOR authentication model whose security is discussed in the next subsection. The Alice's public and private keys to achieve authentication are written in Table 1 where k_0, k_1 and k_2 are random binary strings.

In this protocol h_r constitutes the hash code of the message m_r. The message m_0 is assumed to be the credential containing identification data of user Alice. For message m_r, Alice obtains $h_r = f(m_r)$ where f denotes the hash function and she generates a random string x_r, then she proceeds to compute $y_r = x_r \oplus h_r$. The authentication process is represented in Figure 3 and is

Table 1: Alice Private/Public key definition in Lizama's XOR authentication.

Private Key	Public Key
$\{k_0,\ k_1,\ k_2\}$	$\{x_0 \oplus k_0 \oplus k_2,\ y_0 \oplus k_1 \oplus k_2,\ k_0 \oplus k_1\}$

Table 2: Alice database (DB) contains the XOR sum of transactions. Integrity of the DB is verified because h_{r+1} can be derived from $h_r \oplus h_{r+1}$ and h_r from 0 to any row r.

#	First Sum Term	Second Sum Term
0	h_0	$x_0 \oplus x_1$
1	$h_0 \oplus h_1$	$x_1 \oplus x_2$
2	$h_1 \oplus h_2$	$x_2 \oplus x_3$
	...	
r	$h_{r-1} \oplus h_r$	$x_r \oplus x_{r+1}$

completed after Bob verifies Eq. 2. We will denote the terms in Figure 3, as center term c_j, down term d_j, left term l_j and right term r_j, so, we state Eq. 1 for $j \geq 1$. The initial authentication step is represented in Figure 3 and will be discussed next.

While the index j is intended to denote an arbitrary round beyond the initial authentication, the index r will be used to denote the user authentication from the beginning. Thus, round 0 e.g. $r = 0$, is the first authentication process, as seen in Figure 3. However, there is an initial authentication step where Bob gets the Alice's public key from her database. Then Bob XORs the three components of the key to get $x_0 \oplus k_0 \oplus k_2 \oplus y_0 \oplus k_1 \oplus k_2 \oplus k_0 \oplus k_1 = x_0 \oplus y_0 \oplus x_1 \oplus y_1$, but remember that $h_0 = x_0 \oplus y_0$ and $h_1 = x_1 \oplus y_1$ (see Table 1). Also, h_0 and $h_0 \oplus h_1$ are allocated in the Alice DB, from here Alice is correctly authenticated.

$$\begin{aligned} c_j &= y_j \oplus k_{j+1} \oplus k_{j+2} \\ l_j &= k_{j+1} \oplus k_{j+3} \\ r_j &= y_{j+1} \oplus k_{j+2} \oplus k_{j+3} \\ d_j &= x_j \oplus x_{j+1} \end{aligned} \tag{1}$$

To be authenticated in the round 0 Alice computes h_0 and chooses two random numbers x_0 and x_1, then she registers h_0 and $x_0 \oplus x_1$ in her public database (see Table 2). The execution involves the following steps.

1. Alice chooses a new random number x_2 to get $x_1 \oplus x_2$. She computes h_1 from m_1 and updates her DB uploading $h_0 \oplus h_1$ and $x_1 \oplus x_2$ as shown in Table 2.
2. Alice sends to Bob:
   – $\{k_1 \oplus k_3\}$, $\{y_2 \oplus k_2 \oplus k_3\}$ and the message m_1.
2. Bob retrieves $\{x_0 \oplus x_1\}$ and $\{h_0 \oplus h_1\}$ from Alice's DB.
3. Bob XORs h_0, $\{x_0 \oplus x_1\}$, $\{k_1 \oplus k_3\}$, $\{y_1 \oplus k_2 \oplus k_3\}$ and $\{y_0 \oplus k_1 \oplus k_2\}$ (see Figure 3) and he verifies that h_1 be equal to the computed value, that is $h_1 == f(m_1)$. This condition is stated

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0295.v1

in Eq. 2 for $r \geq 1$. Bob also checks that the received $h_0 \oplus h_1$ is equal to the data retrieved from Alice's DB.

$$\begin{aligned} h_r \;==\; & h_{r-1} \oplus \\ & \oplus\; x_{r-1} \;\oplus\; x_r \;\oplus\; k_r \;\oplus k_{r+2} \oplus \\ & \oplus\; y_r \;\oplus\; k_{r+1} \oplus\; k_{r+2} \oplus \\ & \oplus\; y_{r-1} \;\oplus\; k_r \;\oplus\; k_{r+1} \end{aligned} \tag{2}$$

2.2 First Security Analysis

Suppose the eavesdropper Eve finds the triplet $\{k_0{}', k_1{}', k_2{}'\}$, and the pair $\{x_0{}', y_0{}'\}$, so that those numbers meet the next relations, where $y_r = h_r \oplus x_r$ for $r = 0, 1$:

— $h_0 \oplus x_0 \oplus k_1 \oplus k_2 = h_0 \oplus x_0{}' \oplus k_1{}' \oplus k_2{}'$
— $k_1 \oplus k_3 = k_1{}' \oplus k_3{}'$
— $h_1 \oplus x_1 \oplus k_2 \oplus k_3 = h_1{}' \oplus x_1{}' \oplus k_2{}' \oplus k_3{}'$

Assume that in round 0 Eve tries to mount a Man In The Middle (MITM) attack to the protocol depicted in Figure 3. Eve intercepts Alice's left term $k_1 \oplus k_3$ then she replaces it with $k_1{}' \oplus k_3{}'$. Also, she blocks Alice's right term $h_1 \oplus x_1 \oplus k_2 \oplus k_3$ and she substitutes it with $h_1{}' \oplus x_1{}' \oplus k_2{}' \oplus k_3{}'$. Bob XORs the right and the left terms, the DB term $x_0 \oplus x_1$ and the previous term as shown in Figure 4. However, on the other side, Eve gets $h_0 \oplus h_1{}' \oplus x_0 \oplus x_0{}' \oplus x_1 \oplus x_1{}'$ because $x_0 \neq x_0{}'$ and $x_1 \neq x_1{}'$. Otherwise if $x_0 = x_0{}'$ and $x_1 = x_1{}'$ Bob could derive $h_0 \oplus h_1{}'$. Remember that Alice has stored $x_0 \oplus x_1$ in her public DB.

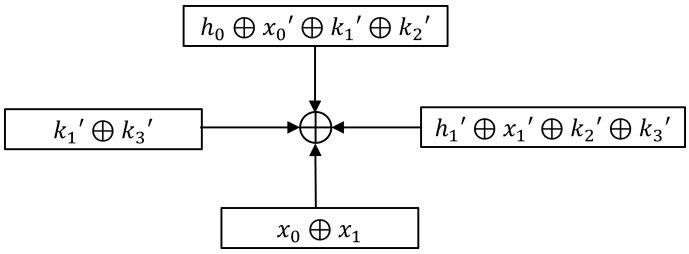

Fig. 4: Security analysis for a Man In The Middle (MITM) attack. Eve gets $h_0 \oplus h_1{}' \oplus x_0 \oplus x_0{}' \oplus x_1 \oplus x_1{}'$.

2.3 Second Security Analysis

Suppose Alice is unable to store the term $x_r \oplus x_{r+1}$ in her DB as indicated in Table 2 where $r \geq 0$. Therefore she sends it directly to Bob, thus in Equation 1 we have $l_j = k_{j+1} \oplus k_{j+3} \oplus x_j \oplus x_{j+1}$.

In the previous subsection, we assumed Eve could adjust the left and right term in Figure 4. Now, we will go deeper to discuss this scenario. As commented before, the attacker is allowed to find $\{k_0{}',\ k_1{}',\ k_2{}'\}$ and $\{x_0{}', y_0{}'\}$ so that such numbers meet the following relations.

 doi:10.20944/preprints202209.0295.v1

— $k_0{}' \oplus k_1{}' = k_0 \oplus k_1$
— $k_0{}' \oplus k_2{}' = k_0 \oplus k_2$
— $x_0{}' \oplus k_0{}' \oplus k_2{}' = x_0 \oplus k_0 \oplus k_2$
— $y_0{}' \oplus k_1{}' \oplus k_2{}' = y_0 \oplus k_1 \oplus k_2$
— $h_0{}' = h_0$

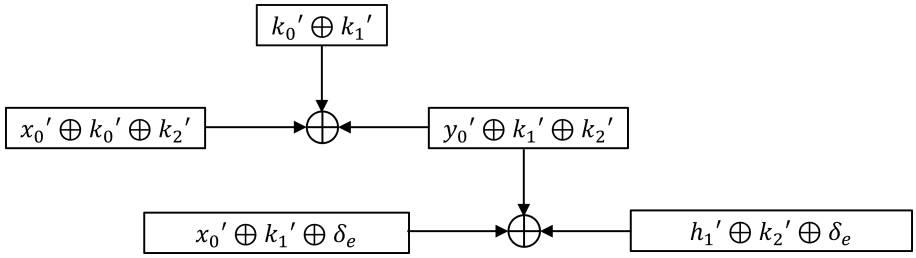

Fig. 5: We write $h_1{}'$ as $y_1{}' \oplus x_1{}'$ then $\delta_e = x_1{}' \oplus k_3{}'$.

Nevertheless the following inequalities hold: $k_0{}' \neq k_0$, $k_1{}' \neq k_1$, $k_2{}' \neq k_2$, $x_0{}' \neq x_0$ and $h_1{}' \neq h_1$. What we want to determine is whether Eve, using her own numbers, could produce Alice's left term: $l_0 = k_1 \oplus k_3 \oplus x_0 \oplus x_1$ and right term: $r_0 = y_1 \oplus k_2 \oplus k_3$ as represented in Figure 5.

Assume Eve choose $x_1{}'$ and $k_3{}'$ for the next round. By choosing them, Eve should get Alice's numbers. Let $\delta_e = x_1{}' \oplus k_3{}'$, then as can be seen in Figure 5 Eve could adjust one of the two numbers but not both. Let's say she adjusts δ_e to get the left term $l_0 = k_1 \oplus k_3 \oplus x_0 \oplus x_1$, but doing this causes the inequality of the right term $r_0 = y_1 \oplus k_2 \oplus k_3$. As a result, Eve's presence in the middle of the protocol becomes detectable.

2.4 Efficiency

For the quantum era is recommended that the size of the hash code should be at least 256 bits to be used. Thus, the recommended size of the private key $\{k_0, k_1, k_2\}$ achieves $3 \cdot 256 = 768$ bits. Also, the size of the public key $\{x_0 \oplus k_0 \oplus k_2, y_0 \oplus k_1 \oplus k_2, k_0 \oplus k_1\}$ amounts $3 \cdot 256 = 768$ bits. These key sizes are really small compared to current public key cryptosystems.

In addition, a database of each user must be publicly maintained, which contains the XOR sum of hash codes and the Alice keys $x_r \oplus x_{r+1}$. Then the size of the packet sent by Alice, each round, achieves 512 bits: $\{x_{r-1} \oplus x_r \oplus k_r \oplus k_{r+2}\}$ and $\{y_r \oplus k_{r+1} \oplus k_{r+2}\}$ plus the size of the message m_r itself.

Preliminary tests carried out on the cocalc.com platform show execution times of only a few milliseconds. A prototype is currently being implemented to determine the precise timing of key generation as well as authentication/encryption processes.

2.5 Key Renewal

Two reasons could lead to the renovation of the authentication chain produced by Alice's DB.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0295.v1

1. The length of the chain has become too long, thus authenticating a user DB can become computationally costly (see the caption of Table 2).
2. If Alice acts as a server, she could maintain an authentication chain with each user separately.

To meet both requirements, Alice executes the following procedure: she keeps the keys k_0 and k_1 but she updates k_2, what is achieved choosing randomly other binary string. As a result, the root key $\{k_0 \oplus k_1\}$ remains unchanged but the other two keys $\{k_0 \oplus k_2\}$, $\{k_1 \oplus k_2\}$ must be substituted (see Table 3).

Table 3: Renewal strategy for XOR authentication mode.

Public Key			
1	$x_0 \oplus k_0 \oplus k_1$	$y_0 \oplus k_0 \oplus k_2$	$k_1 \oplus k_2$
2		$y_0 \oplus k_0 \oplus k_2'$	$k_1 \oplus k_2'$
3		$y_0 \oplus k_0 \oplus k_2''$	$k_1 \oplus k_2''$

This is feasible and safe because, as stated before, an adversary cannot derive k_0, k_1 neither k_2 despite she has access to the XOR sums $\{k_0 \oplus k_1\}$, $\{k_0 \oplus k_2\}$ and $\{k_1 \oplus k_2\}$. The resulting public keys are written in Table 3. Any user who wants to authenticate Alice only needs to verify that the root key $x_0 \oplus k_0$ keeps unchanged. In the first renewal case, the last message of the original chain will become the first message of the new chain after the execution of the key renewal algorithm. For the second renewal case, Alice maintains a separated chain of communication with each user of the system. Such situation is depicted in Figure 6.

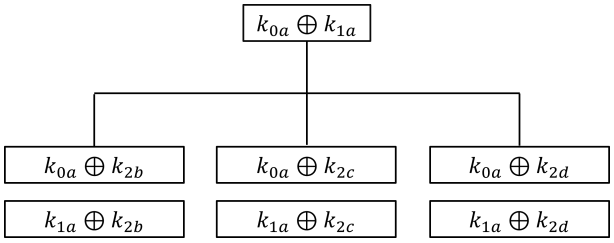

Fig. 6: User i can keep an authenticated conversation with user j using the keys $k_{0_i} \oplus k_{1_i}$, $x_{0_i} \oplus k_{0_i} \oplus k_{2_j}$ and $y_{0_i} \oplus k_{1_i} \oplus k_{2_j}$.

3 Message XOR Encryption

We will now define a new type of data stream encryption that can also be converted to a block encryption scheme, in which the encryption key acquires the same size as the data to be sent, although this size may be arbitrarily large.

 doi:10.20944/preprints202209.0295.v1

3.1 Multiple Key Reuse

Our encryption method relays in a mechanism to generate the encrypted message that we call multiple key reuse which is symbolically established by Eq. 3 for $r \geq 0$. According to this equation, the encrypted messages can be written as $c_2 = m_2 \oplus x_2 \oplus x_1 \oplus x_0$, $c_3 = m_3 \oplus x_3 \oplus x_2 \oplus x_1$, $c_4 = m_4 \oplus x_4 \oplus x_3 \oplus x_2$, $c_5 = m_5 \oplus x_5 \oplus x_4 \oplus x_2$ and so on. Messages m_0 and m_1 were encoded using the initial secret keys as we will explain soon. As can be deduced from this equation, every key is used tree times. Eq. 3 holds for $r \geq 0$ where x_{r+2}, x_{r+1} and x_r represent secret random numbers. In the following sections we will provide arguments about the security of this approach.

$$c_{r+2} = m_{r+2} \oplus x_{r+2} \oplus x_{r+1} \oplus x_r \tag{3}$$

3.2 XOR Encryption Algorithm

In this scenario Alice wants to send secret messages to Bob. Let's assume that Alice and Bob share an initial secret key because they have possibly performed a secret key establishment algorithm. Actually, they share three secret keys $k_{0ab}, k_{1ab}, k_{2ab}$, then they compute XOR between the secret keys $\{k_{0ab} \oplus k_{1ab}, k_{0ab} \oplus k_{2ab}, k_{1ab} \oplus k_{2ab}\}$. In this explanation we will denote them simply as k_0, k_1, k_2.

Users run the protocol depicted in Figure 7, so that Alice (or Bob) starts sending the first encrypted message m_0 which satisfies the relation $m_0 = x_0 \oplus y_0$. As stated by Equation 4, in this scenario there is no a database of any user so we have removed the term d_j from Equation 1 but appending the terms $x_j \oplus x_{j+1}$ into l_j.

In this protocol we establish the substitution rule $k_j = x_{j-3}$ which implies that in r_j the term k_{j+2} must be substituted by x_{j-1} and k_{j+3} by x_j while in c_j instead of k_{j+1} it must be written x_{j-2} and k_{j+1} by x_j for $j \geq 1$. Applying this rule to l_j gives $x_{j-2} \oplus x_{j+1}$.

$$\begin{aligned} c_j &= y_j \oplus k_{j+1} \oplus k_{j+2} = x_{j-2} \oplus x_{j-1} \oplus y_j \\ l_j &= k_{j+1} \oplus k_{j+3} \oplus x_j \oplus x_{j+1} = x_{j+2} \oplus x_{j+1} \\ r_j &= y_{j+1} \oplus k_{j+2} \oplus k_{j+3} = x_{j-1} \oplus [x_j] \oplus y_{j+1} \end{aligned} \tag{4}$$

In r_j of Equation 4 we have enclosed in brackets the term $[x_j]$ because it will be appended by Bob after he receives $x_{j-1} \oplus y_{j+1}$ (see Figure 7). The XOR computation between c_j, l_j and r_j results in $c_j \oplus l_j \oplus r_j = x_j \oplus y_j \oplus x_{j+1} \oplus y_{j+1}$.

 doi:10.20944/preprints202209.0295.v1

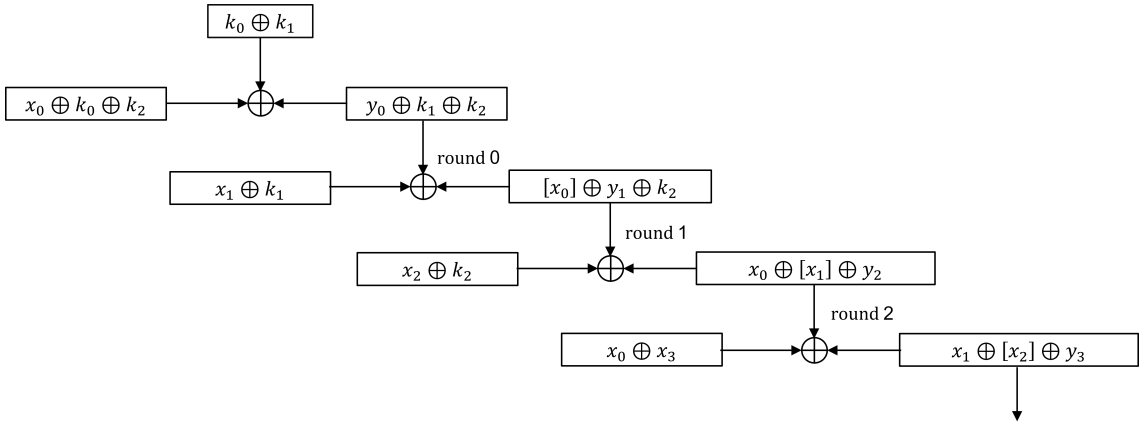

Fig. 7: To recover an encrypted message Bob must able to get k_j and k_{j+3} as they are written in r_j of Eq. 5 (see Figure 3). To derive k_j and k_{j+3} Bob uses the relation $k_{j+2} = x_{j-1}$ for $j \geq 1$, then he XORs it to the incoming number from the right hand side. We have highlighted the term x_{j-1} enclosing it in brackets.

In this encryption scheme, instead of sending the hash value of the message h_r as in the authentication mode, Alice uses a piece of the message to compute y_r as $m_r \oplus x_r$ where x_r is a random string and m_r is the cleartext message. In the round r Bob applies the cipher XOR rule depicted in Figure 8, which gives $x_{r+2} \oplus y_{r+2} \oplus x_{r+3} \oplus y_{r+3} = m_{r+2} \oplus m_{r+3}$. Then, m_{r+3} is recovered according to Eq. 5 for $r \geq 2$. Rounds $r = 0$ and $r = 1$ are run according to the Figure 7.

1. For message m_r where $r \geq 2$, Alice generates a random string x_r, then she computes $y_r = x_r \oplus m_r$. She sends to Bob $\{x_{r-2} \oplus x_{r+1}\}$ and $\{x_{r-2} \oplus x_{r-1} \oplus y_r\}$.
2. Bob XORs x_r to the incoming number on the right hand side of Figure 7, thus he gets $\{x_{r-1} \oplus [x_r] \oplus y_{r+1}\}$.
3. As stated by Eq. 5 Bob XORs $\{x_{r-2} \oplus x_{r+1}\}$, $\{x_{r-1} \oplus [x_r] \oplus y_{r+1}\}$ and $\{x_{r-2} \oplus x_{r-1} \oplus y_r\}$, where the last term has been stored at Bob's side from the previous round. This XOR operation yields $m_r \oplus m_{r+1}$, then Bob is able to derive m_{r+1} (see Figure 8).

$$\begin{aligned} m_{r+1} \;==\; & m_r \oplus \\ & \oplus\; x_{r-2} \;\oplus\; x_{r+1} \oplus \\ & \oplus\; x_{r-1} \;\oplus\; [x_r] \;\oplus\; y_{r+1} \oplus \\ & \oplus\; x_{r-2} \;\oplus\; x_{r-1} \;\oplus\; y_r \end{aligned} \tag{5}$$

In practice Bob must be capable to derive x_{j-1} for $j \geq 1$ as indicated by the rule $k_{j+2} = x_{j-1}$. The relations that allow Bob to obtain each x_{j-1} are $k_0 \oplus k_2 \oplus x_0 \rightarrow x_0$, $k_1 \oplus x_1 \rightarrow x_1$, ..., $x_r \oplus x_{r+3} \rightarrow x_{r+3}$.

 doi:10.20944/preprints202209.0295.v1

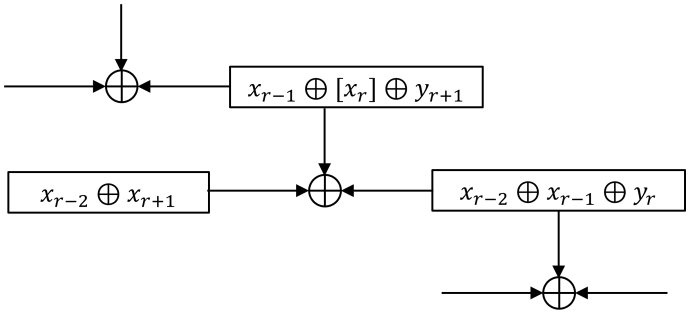

Fig. 8: It is represented the cipher XOR rule of Eq. 5 $\{x_{r-2} \oplus x_{r+1}\} \oplus \{x_{r-1} \oplus [x_r] \oplus y_{r+1}\} \oplus \{x_{r-2} \oplus x_{r-1} \oplus y_r\} = m_r \oplus m_{r+1}$.

3.3 Attack Scenario

Let us analyze an scenario which could be exploited by an eavesdropper to implement the following attack: Eve captures the numbers sent by Alice through the public channel. Eve computes the sum of the incoming numbers arriving in Figure 7 from the right (denoted as s_r), where we have substituted $y_r = m_r \oplus x_r$.

$$
\begin{aligned}
&s_0 = m_0 \oplus k_1 \oplus k_2 \oplus x_0 \\
&s_1 = m_0 \oplus m_1 \oplus k_1 \oplus x_0 \oplus x_1 \\
&\dots \\
&s_r = m_0 \oplus \dots \oplus m_r \oplus k_1 \oplus k_2 \oplus x_{r-1} \oplus x_r
\end{aligned}
$$

Now, the sum terms are XORed with the term $y_0 \oplus k_1 \oplus k_2 = m_0 \oplus x_0 \oplus k_1 \oplus k_2$.

$$
\begin{aligned}
&{s_0}' = 0 \\
&{s_1}' = m_1 \oplus x_1 \oplus k_2 \\
&\dots \\
&{s_r}' = m_1 \oplus \dots \oplus m_r \oplus x_0 \oplus x_{r-1} \oplus x_r \oplus k_2
\end{aligned}
$$

Eve also computes the XOR sum of the left hand input terms (denoted t_r) including the term $k_0 \oplus k_1$.

$$
\begin{aligned}
&t_0 = x_0 \oplus k_1 \oplus k_2 \\
&t_1 = x_0 \oplus x_1 \oplus k_2 \\
&\dots \\
&t_r = x_{r-2} \oplus x_{r-1} \oplus x_r
\end{aligned}
$$

If Eve XORs $m_1 \oplus \dots \oplus m_5 \oplus x_0 \oplus x_4 \oplus x_5 \oplus k_2$ and $x_3 \oplus x_4 \oplus x_5$ she gets $m_1 \oplus \dots \oplus m_5 \oplus x_0 \oplus x_3 \oplus k_2$. Although the term $x_0 \oplus x_3$ is sent by Alice through the public channel, it cannot be exploited by Eve to obtain $m_1 \oplus \dots \oplus m_5$ due to the encryption key k_2 that appears in Eve's relation.

3.4 Perfect Forward Secrecy

Suppose the key x_t has been exposed and therefore has been compromised. As a consequence, all messages for $r \geq t$ will be exposed because $x_r \oplus x_{r+3}$ is sent over the public channel. However, since the keys x_r for $r < t$ have no dependency on x_t then secrecy of messages where $r < t$ is preserved and the cryptosystem is said to exhibit Perfect Forward Secrecy (PFS).

 doi:10.20944/preprints202209.0295.v1

3.5 Group Encrypted Communication

Provided a group of users share a secret key, the encrypted conversation within the group is possible because each user derives the current plaintext message of the conversation as shown in Figure 7.

3.6 Batch Encryption

Instead of sending each piece of message m_r, one by one over the network, Alice can send several messages into a single batch packet. If the network traffic conditions allow it, the communication could be commuted to send again individual fragments. In any case, the amount of key material is equal to the size of the whole transferred message. See the relations listed below which correspond to the right input numbers in Figure 7.

$$y_0 \oplus k_1 \oplus k_2 = m_0 \oplus x_0 \oplus k_1 \oplus k_2$$
$$y_1 \oplus x_0 \oplus k_2 = m_1 \oplus x_0 \oplus x_1 \oplus k_2$$
$$\dots$$
$$y_r \oplus x_{r-2} \oplus x_{r-1} = m_r \oplus x_{r-2} \oplus x_{r-1} \oplus x_r$$

Messages from m_0 to m_r are encrypted with the keys x_0 to x_r plus the initial secret keys. Since the whole plaintext message is encrypted with a key that reaches the same size, we satisfy the criteria of Shannon's perfect encryption.

4 XOR Chain

The XOR authentication model discussed so far can be used to define a system of transaction-linked blocks to maintain a secure database of such network transactions. The most used system for this purpose is Blockchain, in which Bitcoin miners perform a Proof of Work (PoW) that consists of finding a number such that concatenated with the hash code of the previous block produces a hash code prefixed with a predetermined number of zeros. The miner is then authorized to establish the next block into the blockchain. The block contains the root code of the Merkle tree used to identify the transactions group. As a final step, the miner receives a reward in bitcoins. Unfortunately, to find the required nonce, miners need to compute a large number of hashes per unit of time, which entails a large energy cost.

To avoid such computational effort and the required electrical power consumption, the random selection of the miner could be proposed. However, doubts could prevail about the integrity of the random selection algorithm. We will reformulate this scenario through the XOR chain algorithm. But before introducing XOR chain, we will discuss in the next section a similar scheme from which we have developed the XOR chain algorithm that we have named Crypto Bingo.

4.1 Crypto Bingo

We define Crypto Bingo in the context of a game scenario. Let us start by stating that each player, say i, registers into the game by posting his binary number s_i that each user choose randomly. For the round j each player i computes $g_{ij} = f^{x_{ij}}(w_{ij})||y_{ij}$, here $w_{ij} = s_r||{h_{ij}}'$ and ${h_{ij}}'$ is the hash code that belongs to the previous winning player, x_{ij} is a random number and y_{ij} is the hash code of their Merkle tree which identify the transactions group of r. Let us go deeper into this algorithm.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0295.v1

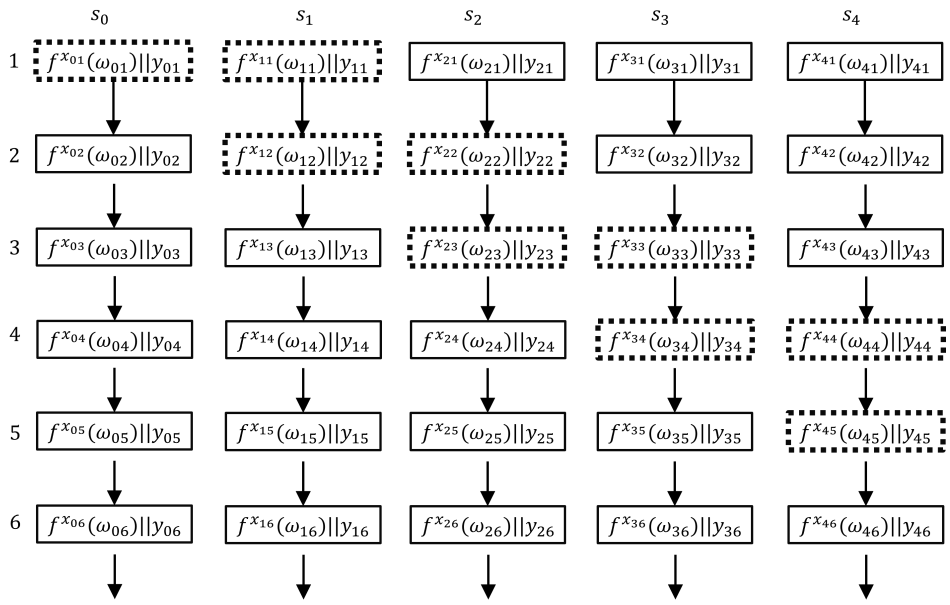

Fig. 9: Players register $h_{ij} = f(g_{ij})$, then they announce $g_{ij} = f^{x_{ij}}(w_{ij})||y_{ij}$ and (x_{ij}, y_{ij}).

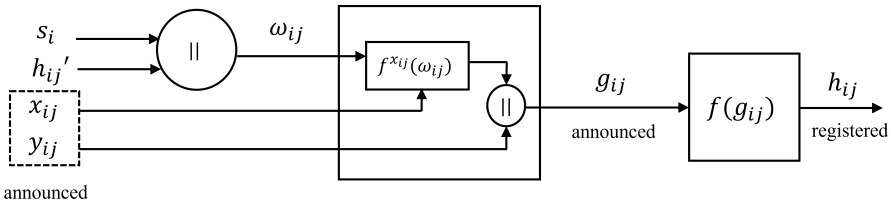

Fig. 10: The diagram process to compute h_{ij}.

— In the first round, the root player computes and registers h_{01} into the game, where $h_{01} = f^{x_{01}}(w_{01})||y_{01}$ and $w_{01} == s_0$ (instead of $s_0||h_{01}{}'$ because there is no a previous winner). In addition, players compute and register h_{i1} into the game. Then the root player announces g_{01} and the player whose g_{i1} is at the minimum Hamming distance denoted as δ, wins and acquires the right to place his block into the chain. Let s_1 be the winning player with h_{11}. To be verified the root player publishes (x_{01}, y_{01}) while player 1 announces (x_{11}, y_{11}). Then, all players verify that they correspond to h_{01} and h_{11}, respectively.

— Next, in round 2, s_1 computes and register h_{12} in the game, where $h_{12} = f^{x_{12}}(w_{12})||y_{12}$ and $w_{12} == s_1||h_{11}$. All players compute and publish h_{i2}. Then s_1 announces g_{12} and the player who gets δ wins and owns the block chain. Player 1 is verified after he publishes (x_{12}, y_{12}), also the verification of the winning player 2 is performed using (x_{22}, y_{22}) (see Figures 9 and 10).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 September 2022 doi:10.20944/preprints202209.0295.v1

As can be deduced, verifying the winning player's given h_{ij} and (x_{ij}, y_{ij}) can be efficiently performed. The players do not require a high computational effort and the expenditure of energy is minimal. More important, no player can break the rules of the game since everyone publishes their registration number w_i since the beginning of the game.

4.2 XOR Chain Algorithm

In this section, we will apply the authentication model for building a blockchain system. Suppose all players act as Eve according to the rules described in subsection 2.3 Second Security Analysis where Alice sends $x_j \oplus x_{j+1}$ over the network and Eve can adjust the left term but misfit the right term by an amount of δ_e as indicated in Figure 5.

$$\begin{aligned} c_j &= y_j \oplus k_{j+1} \oplus k_{j+2} \\ l_j &= k_{j+1} \oplus k_{j+3} \oplus x_j \oplus x_{j+1} \\ r_j &= y_{j+1} \oplus k_{j+2} \oplus k_{j+3} \end{aligned} \tag{6}$$

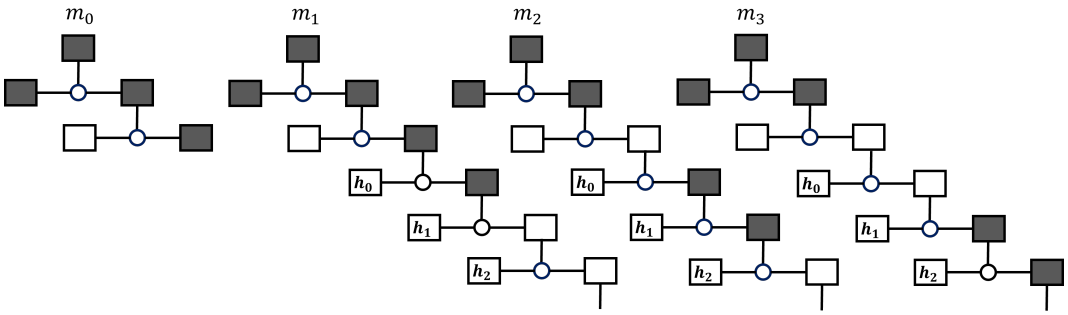

Fig. 11: The XOR Chain model. In the first round Miner m_1 computes the keys for the next round such that the resulting left term is equal to h_0. Black boxes represent that they contain the same (or closer) number.

In this explanation, we will refer to Alice as the root player m_0 (see Figure 11).

1. Using his numbers k_0, k_1, k_2 and x_0, y_0 the root player denoted as m_0 publishes c_0, l_0 and r_0 computed as $c_0 = y_0 \oplus k_1 \oplus k_2$, $l_0 = k_1 \oplus k_3 \oplus x_0 \oplus x_1$, $r_0 = y_1 \oplus k_2 \oplus k_3$. All players select their own set of numbers (k_0', k_1', k_2', k_3', x_0', x_1', y_0' and y_1'), so that they throw up the root numbers c_0, l_0, r_0.
2. In the second round the root player m_0 computes and publishes $l_1 = x_1 \oplus x_2 \oplus k_2 \oplus k_3$, $r_1 = y_2 \oplus k_3 \oplus k_4$ according to Eq. 6 where $r = 3$. Then, each player chooses k_3' so that their left term l_1' matches the root left term l_1, then he publishes both results: left term $l_1' = x_0' \oplus k_1' \oplus \delta_w$ and right term $r_1' = h_1' \oplus k_2' \oplus \delta_w$ where $\delta_w = x_1' \oplus k_3'$ (see Figure 5).
3. The player who achieves the pre-specified Hamming distance wins. For simplicity, this situation is illustrated in Figure 11 as the player m_1 who is at the right of the root node. The winner places the hash code from the root of their Merkle tree on the blockchain.

 doi:10.20944/preprints202209.0295.v1

4. The winner m_1 computes the keys for the next round, choosing k_4 such that the resulting left term l_2 is equal to h_0 as indicated by Eq. 7 where $l_{r-2} = h_{r-4}$. This action prevent the winning player from favoring another player. The process to choose the next winner is repeated.

$$l_{r-2} = \ h_{r-4} \ = \ k_{r-1} \ \oplus \ k_{r+1} \ \oplus x_{r-2} \ \oplus \ x_{r-1} \tag{7}$$

At network time n, the current winning node contains the valid historical list of system blocks: $h_0, h_1, \ldots, h_n$ (see Figure 11).

5 Conclusions

In this work we have introduced a new approach to achieve the perfect secret using the XOR function and applying a mechanism that we call multiple key reuse. We have discussed two modes of the basic algorithm: message authentication and encryption. We argue that our method exhibits Perfect Forward Secrecy (PFS) and batch encryption to adapt to network traffic conditions. Since the crypto system only requires the XOR function, the execution time only takes a few milliseconds.

More importantly, we have taken advantage of the basic authentication system to define a new approach to implementing blockchain that does not require proof of work. Our method relies on another algorithm called Crypto Bingo that we have previously developed. The security of the system is fundamentally based on XOR operations, which is why it can be established without great complexity. Therefore, we believe that our scheme represents a safe, efficient and simple alternative to the current Blockchain technology in the era of the Internet of Things (IoT) and quantum computing.

 doi:10.20944/preprints202209.0295.v1

Bibliography

[1] C. E. Shannon, "Communication theory of secrecy systems," *The Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.

[2] A. Feutrill and M. Roughan, "A review of shannon and differential entropy rate estimation," *Entropy*, vol. 23, no. 8, p. 1046, 2021.

[3] T. Shimeall and J. Spring, *Introduction to information security: a strategic-based approach*. Newnes, 2013.

[4] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 212–219, ACM, 1996.

[5] H. Wu, "A new stream cipher hc-256," in *International Workshop on Fast Software Encryption*, pp. 226–244, Springer, 2004.

[6] M. Hell, T. Johansson, and W. Meier, "Grain-a stream cipher for constrained environments." estream, ecrypt stream cipher," 2005.

[7] M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansen, and O. Scavenius, "Rabbit: A new high-performance stream cipher," in *International workshop on fast software encryption*, pp. 307–329, Springer, 2003.

[8] S. Babbage and M. Dodd, "The stream cipher mickey 2.0," *ECRYPT Stream Cipher*, 2006.

[9] D. J. Bernstein, "Salsa20 specification," *eSTREAM Project algorithm description, http://www. ecrypt. eu. org/stream/salsa20pf. html*, 2005.

[10] C. De Canniere, "Trivium: A stream cipher construction inspired by block cipher design principles," in *International Conference on Information Security*, pp. 171–186, Springer, 2006.

[11] P. Ekdahl and T. Johansson, "Snow-a new stream cipher," in *Proceedings of First Open NESSIE Workshop, KU-Leuven*, pp. 167–168, Citeseer, 2000.

[12] E. Biham, R. Anderson, and L. Knudsen, "Serpent: A new block cipher proposal," in *International workshop on fast software encryption*, pp. 222–238, Springer, 1998.

[13] L. A. Lizama-Perez, "Digital signatures over hash-entangled chains," *SN Applied Sciences*, vol. 1, no. 12, p. 1568, 2019.

[14] L. A. Lizama-Pérez, "Digital signatures over hmac entangled chains," *Engineering Science and Technology, an International Journal*, p. 101076, 2021.