Journal of Computer Virology and Hacking Techniques (2024) 20:1–14
https://doi.org/10.1007/s11416-023-00482-1

ORIGINAL PAPER

Networking and cryptography library with a non-repudiation flavor for blockchain

Mohamed Rasslan[1] · Mahmoud M. Nasreldin[2] · Doaa Abdelrahman[3] · Aya Elshobaky[4] · Heba Aslan[1,3]

Received: 23 August 2022 / Accepted: 9 May 2023 / Published online: 5 August 2023

Abstract
Blockchain is currently one of the most widely discussed inventions in the information and communication technology industry. It is a pillar of the fourth industrial revolution and it is a cryptographically demanding technology that is regarded as one of the most influential topics in academia. Many blockchain platforms currently utilize third-party cryptographic libraries that offer many cryptographic primitives in order to ensure users' protection against cyber-attacks. The Networking and Cryptography library (NaCl) is an open-source library for cryptographic primitives. NaCl is known to be one of the best libraries that provide usability property. Although NaCl is easy to use and ensures: confidentiality, integrity, and authenticity, it fails to provide the non-repudiation service. In this paper, an improvement to the blockchain platforms is proposed by enhancing the NaCl library to achieve the non-repudiation property that enhances the security level of the implemented blockchain platform. In NaCl, to provide the aforementioned security services, messages are signed and then encrypted. Therefore, the non-repudiation service is not provided. In this paper, the proposed solution is based on adding a signature block to enable a non-repudiation property. First, logical analysis is conducted using the BAN logic on the NaCl library to prove that it does not provide the non-repudiation property. Subsequently, a modification to the library is proposed, and the correctness of the proposed solution is proven using BAN logic. The analysis suggests that the proposed solution fixes the aforementioned problem.

Keywords Blockchain · Cryptocurrencies · Networks · Confidentiality · Integrity · Non-repudiation · Cryptographic libraries · Security goals

1 Introduction

Since the introduction of Bitcoin in 2008 [1], the concept of blockchain, which dates back to the 1970s [2] has been employed by many developers to provide solutions for their applications. Blockchain technology and cryptocurrencies have sparked concerns related to guaranteeing users' privacy and security protection due to the deployment of algorithms that can be selected by users ("bring your own algorithm" concept). Blockchain, a public ledger where transactions are stored in blocks, is characterized by immutability, decentralization, anonymity, and auditability [3]. Data are stored in a chain of blocks, where each block is linked to the previous block through its hash. The nodes of the network (miners) are responsible for calculating the hash value [4]. Blockchain blocks are distributed to all users, leading to block immutability. This is because to alter any block, a large number of calculations must be performed. In addition, the consensus process is performed in a decentralized and distributed manner without the intervention of a third party. To guarantee data integrity, blockchain uses cryptographic algorithms such as asymmetric encryption, digital signature, and hash functions. In addition, cryptographic techniques provide the ability to trace transactions and ensure data confidentiality [3]. The following three generations of blockchain exist: blockchain 1.0, blockchain 2.0, and blockchain 3.0. However, a new generation, blockchain 4.0 is currently under development [5].

✉ Mohamed Rasslan
mohamed@eri.sci.eg

[1] Informatics Department, Electronics Research Institute, Cairo 12622, Egypt

[2] CyberistInsight, Ottawa, ON K1S 1N4, Canada

[3] Center of Informatics Science, Faculty of Information Technology and Computer Science, Nile University, Giza 12588, Egypt

[4] Alexandria University, Alexandria 21500, Egypt

While blockchain 1.0 is mainly concerned with cryptocurrency applications, blockchain 2.0 concerns whole markets and economies; it stores not only transactions but also smart contracts and applications; an example of this type of application is Ethereum. Blockchain 3.0, which is considered an evolution of blockchain 2.0, extends the technology into more applications in areas such as art, health, science, and government. Artificial intelligence has recently been introduced in blockchain 4.0, to enhance consensus efficiency, scalability, and energy efficiency and to incorporate blockchain technology into real environments. Blockchain is used in many applications, such as banking, financial transactions, supply chain management, healthcare, and education. The security requirements for these applications include confidentiality, integrity, and authentication. Confidentiality is delivered through the encryption of sensitive information. On the other hand, authentication is achieved through digital signatures. Integrity is ensured via two mechanisms. The first is the inclusion of the hash value of the preceding block and the second is the calculation of the root hash of all block transactions. Most blockchain platforms use third-party cryptographic solutions [6]. One of these solutions is the networking and cryptography library (NaCl) [7], which is a public domain library for network communication, encryption, decryption, and signatures. NaCl (pronounced "salt") and its forks (i.e. LibSodium) provide cryptographic algorithms that are utilized in well-known blockchain platforms (e.g. Apache Tuweni, Duniter, Tezos), cryptocurrencies (e.g. Zcash), and cryptocoin wallets (e.g. SilentDragon, SilentDragonLite). NaCl is the chemical abbreviation for sodium chloride, commonly known as salt which is an ionic compound with the chemical formula NaCl. Cryptographic algorithms aim to provide the following four well-known security services: confidentiality, authenticity, integrity, and non-repudiation. As mentioned in [7], the NaCl library ensures confidentiality and integrity; however, it does not provide non-repudiation, which is considered an authentication service. Non-repudiation is the main security requirement that must be fulfilled, especially in the case of disputes. This paper proposes a solution to the aforementioned drawback of the NaCl library. In NaCl, to provide the aforementioned security services, messages are signed and then encrypted. Therefore, the non-repudiation service is not provided. In the present paper, to solve this problem, another signature block is added. Furthermore, the security analysis of both the NaCl library and the proposed improvement is conducted using Burrows, Abadi and Needham (BAN) logic [8, 9]. The analysis demonstrates that the proposed modification fixes the problem. The main contributions of the present paper can be summarized as follows:

- Examine which security libraries are used in Blockchain applicationsf.
- Analyze the NaCl library using the BAN logic. The analysis shows that the NaCl library has the following primary drawback, the inability to provide a non-repudiation security service.
- Propose a solution to this problem and analyze it using BAN logic.

The remainder of this paper is organized as follows. The next section provides a literature review of different security libraries. Subsequently, a description of the NaCl library is provided, followed by a security analysis using BAN logic. The proposed solution is then described in detail and analyzed using (BAN) logic in Sect. 3. In Sect. 4, a performance analysis is illustrated. The final section concludes the paper.

2 Literature review

This section provides a brief description of the different cryptographic libraries used in modern applications. Then, a description of the NaCl library modules is detailed. Subsequently, it proposes critiques of the library and security analysis using BAN logic.

2.1 Cryptographic libraries

Blockchain is an immutable distributed digital ledger that is based on cryptographic algorithms to provide security, and uses consensus mechanism to agree upon the transaction log [10]. Cryptographic libraries are used to perform the cryptographic operations needed to provide trust in using Blockchain technology. Following are examples of cryptographic libraries that are used in literature: wolfCrypt [11], an embedded library for symmetric and asymmetric algorithms, Cifra [12] that is mainly concerned in implementing symmetric algorithms, TinyCrypt [13] and micro-ecc (uECC) [14], Crypto-JS [15], and Relic [16], which contains symmetric and asymmetric cryptographic schemes with particular support for many elliptic curves. A lightweight cryptographic library which is named Bouncy Castle is based on lightweight APIs for TLS (RFC 2246, RFC 4346) and DTLS (RFC 6347/ RFC 4347) [17]. In [18], a comparison among cryptographic libraries utilized for encryption operations performed in Blockchain is conducted. The following libraries are used in Blockchain networks: PBC library [19], Crypto++ library [20], MIRACL library [21], and Hyperledger Ursa [22].

Researchers have explained that the usability and misuse resistance of application programming interface (API) implementations are critical for practical cryptographic library deployments. Many solid API design guides provide developers with the required knowledge to develop usable and misuse-resistant cryptographic implementations. In the OpenSSL cryptographic library [23], users cannot modify

their APIs design because of the backward compatibility issue. Few cryptographic libraries consider fundamental usability. However, more recent libraries, such as the Networking and Cryptography library (NaCl) and its forks, provide high levels of usability and misuse-resistance features of APIs [24, 25]. Improving the usability of API is a fundamental objective of the NaCl library. The NaCl library focuses on the most significant use-cases and manipulates use-cases that are not complex in high-level API. The NaCl cryptographic library and its forks, such as libsodium, are distinguished based on their user-oriented viewpoint and concentration on usability features. Cryptographic researchers and software developers have elaborated on the features of cryptographic libraries (e.g., the usability and misuse resistance of API designs) and their importance, which encompasses making cryptographic libraries bug-free codes. Recent libraries such as NaCl and its forks (i.e., libsodium, sodiumoxide, rust_sodium, PyNaCl, TweetNaCl, PySodium, and Keyczar) offer robust concentrations for achieving a high level of usability and misuse-resistant features of APIs design [26, 27]. The Keyczar cryptographic library allows developers to safely and easily use cryptography. The PyNaCl library binds to the NaCl library using Python. Moreover, PyNaCl was considered to be the libsodium interface using Python. Furthermore, sodium is a potential replacement for PyNaCl. NaCl and its forks are cryptographic libraries that are designed and implemented with a focus on usability (these libraries do not require the designer/developer to select cryptographic details). These libraries provide secure asymmetric and symmetric APIs. The NaCl cryptographic library and its forks (i.e., TweetNaCl and libsodium APIs) were used in the implementation of post-quantum cryptographic algorithms [28]. NaCl and its forks (i.e. LibSodium [29, 30]) provide cryptographic algorithms that are utilized in well-known blockchain platforms (e.g. Apache Tuweni [31], Duniter [32], Tezos [33]), cryptocurrencies (e.g. Zcash [34]), and cryptocoin wallets (e.g. SilentDragon [35], SilentDragonLite). In what follows, we elaborate on some blockchain platforms, cryptocurrencies, and cryptocoin wallets that deploy a fork of the NaCl cryptographic library (i.e. LibSodium). Apache Tuweni [31] is a blockchain platform that consists of a set of libraries (e.g. LibSodium which is a fork of the NaCl cryptographic library), tools that help in the development of blockchain, and decentralized software built in Java and other Java Virtual Machine (JVM) languages (e.g. Java, Kotlin, Scala, Groovy, Clojure, Fantom, Ceylon, Jython, JRuby, Frege, Xtend, Golo, Concurnaas, Yeti.) The Ethereum Virtual Machine (EVM) is implemented using Kotlin code. Duniter [32] (originally uCoin) is a blockchain platform that deploys a fork of the NaCl cryptographic library (i.e. LibSodium). It is a free software that enables users to create a new type of P2P crypto-currency using individuals and universal dividends. The Duniter network is decentralized. It uses blockchain to synchronize the state of money across nodes. In contrast to Bitcoin, Duniter has no power race. In Bitcoin, because of the CPU race, power is provided to those who own more computing power. Duniter is democratic; because every user is identified as a unique human, they can write in the blockchain in turns. When a member writes data in the blockchain, he must wait before being able to write again. This ensures that the blockchain does not end in the hand of a few users and that it does not burn too much energy. To identify users, Duniter chooses a self-regulated system by its members. This is known as the Web of Trust (WoT). Each member can certify new users. When a user receives sufficient certifications, and is not too far away from the existing members in the web of trust, he becomes a member. Bitcoin's blockchain mechanism is important for two main reasons: synchronization and security. Duniter's blockchain thus benefit from these two features. However, Duniter's blockchain is slightly different: it does not only store transactions, but also stores community activity in order to define the WoT. It also has a different Proof-of-Work (PoW) mechanism that is made possible by the WoT definition, providing a much more energy-efficient mechanism in order to compute the blockchain. Tezos [33] is an open-source blockchain platform that deploys a fork of the NaCl cryptographic library (i.e. LibSodium) and executes peer-to-peer transactions and serve as a platform for deploying smart contracts. The native cryptocurrency for the Tezos blockchain is the Tez (ISO 4217: XTZ; sign: ꜩ). The Tezos network achieves consensus using proof-of-stake. Tezos uses an on-chain governance model that enables the protocol to be amended (when upgraded proposals receive a favorable vote from the community.) Its testnet was launched in June 2018, and its main net went live in September 2018. Tezos, which was first proposed in 2014, was created by Arthur and Kathleen Breitman. Zcash [34] is a cryptocurrency aimed at using cryptography (i.e. deploy a fork of the NaCl cryptographic library (i.e. LibSodium)) to provide enhanced privacy for its users compared to other cryptocurrencies such as Bitcoin. Zcash is based on Bitcoin's codebase. It shares many similarities, such as, a fixed total supply of 21 million units. Transactions can be "transparent" and similar to bitcoin transactions. Zcash affords private transactors, the option of "selective disclosure". This allows a user to prove payment for auditing purposes. One reason to allow private transactors is to comply with anti-money laundering or tax regulations. "Transactions are auditable but disclosure is under the participant's control. While miners receive 80% of a block reward, 20% is given to the "Zcash development fund": 8% to Zcash Open Major Grants, 7% to Electric Coin Co., and 5% to The Zcash Foundation. Silent Dragon (SD) [35] is a compatible wallet (that deploy a fork of the NaCl cryptographic library (i.e. LibSodium)) and a full node that runs on Linux, Windows, and macOS for HUSH. HUSH is

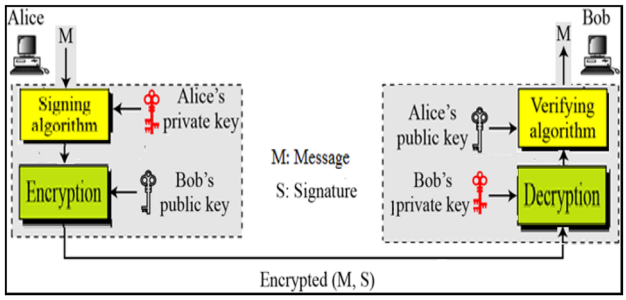

Fig. 1 NaCl library structure

the cryptocurrency of the Hush platform. SD is a one-stop solution that offers high-speed messaging and transactions. The next subsection briefly describes the NaCl library.

2.2 NaCl library

The NaCl library was first published in 2008. The first version of NaCl was released in 2009. Subsequently, the library has undergone several enhancements, such as the inclusion of both the C + + NaCl API and digital signatures. The most recent version of the NaCl library was released in 2017. The NaCl library aims to ensure strong confidentiality, integrity, and availability [7], as shown in Fig. 1. Confidentiality is provided only for the packet content. Therefore, users must incorporate anti-traffic analysis tools in their systems to hide packet lengths and timings. The library consisted of six main functions: crypto_box, crypto_box_open, crypto_box_keypair, crypto_sign, crypto_sign_open, and crypto_sign_keypair. The following paragraphs provide descriptions of these functions.

The crypto_box: This function provides public key authenticated encryption. The first step is to sign a message "m" using the sender's secret key "sk1". This step provides authenticity and integrity. The signed message is then encrypted using the receiver's public key "pk2" to produce the ciphertext "c". This step provides confidentiality. To ensure message freshness, a nonce "n" is used. The following equation represents the encryption function.

$$c = \text{crypto\_box(m, n, pk2, sk1)} \tag{1}$$

The function input consists of the sender's secret key sk1, the receiver's public key pk2, plain packet m, and nonce n, and it outputs an authenticated ciphertext c. The crypto_box function automatically handles all necessary conversions and initializations.

crypto_box_open: Upon receiving the message, the recipient uses its secret key "sk2" to decrypt the signature. It then uses the public key of the sender "pk1" to authenticate the message, as represented in the following function:

$$\text{m} = \text{crypto\_box\_open(c, n, pk1, sk2)} \tag{2}$$

The function input consists of the sender's public key pk1, the receiver's secret key sk2, ciphertext c, and nonce n, and outputs plaintext m. The crypto_box function automatically handles all necessary conversions and initializations.

crypto_box_keypair: This is used to generate the public keys. It takes the secret key (sk) as input and outputs the public key (pk), as shown below.

$$\text{pk} = \text{crypto\_box\_keypair(sk)} \tag{3}$$

crypto_sign: In some cases, confidentiality is not required. For example, a sender may broadcast a message to many people. Therefore, the library requires only public-key signatures. The sender signs the message "m" using its secret key "sk" to produce the signature "sm," as follows:

$$\text{sm} = \text{crypto\_sign(m, sk)} \tag{4}$$

crypto_sign_open: Upon receiving the message, the receiver authenticates the message using the sender's public key "pk" and the signature "sm" as shown below:

$$\text{m} = \text{crypto\_sign\_open(sm, pk)} \tag{5}$$

crypto_sign_keypair: This works as crypto_box_keypair and is used to generate public key pairs. It takes the secret key (sk) as input and outputs the public key (pk), as shown below.

$$\text{pk} = \text{crypto\_sign\_keypair(sk)} \tag{6}$$

For more information about NaCl, the reader can refer to [36, 37] and for more information about blockchain technology, the reader can refer to [38–42]. As stated in [7], the library does not provide non-repudiation, which is the primary requirement for authentication. Therefore, the next section introduces a solution to address this problem.

2.3 Attack on NaCl library and its security analysis

In the following paragraphs, a scenario attack concerning NaCl is described and the security analysis using BAN logic is conducted. First, the scenario attack is described as follows. Assume that Alice sends the message "M" to Bob after it is signed using the secret key of Alice "sk_{alice}," and then encrypted using the public key of Bob "pk_{Bob}". The message sent to Bob is as follows:

$$\text{C1} = \{\{\text{M}\}\text{sk}_{\text{alice}}\}\text{pk}_{\text{Bob}} \tag{7}$$

After receiving the message, Bob decrypts the message using his private key and then the public key of Alice. However, Bob can re-encrypt the signed part using Eve's public key to produce the ciphertext to be sent to Eve:

$$C2 = \{M\}sk_{alice}\}pk_{Eve} \tag{8}$$

Finally, Bob sends the ciphertext to Eve as if it comes from Alice. This represents a violation of the nonrepudiation property. To confirm this scenario threat, the analysis is conducted by applying BAN logic to the NaCl library.

NaCl is typically used in blockchain applications. Authentication protocols are fundamental components of the blockchain, and it is necessary to ensure the correctness of these protocols. BAN logic uses logic to define authentication protocols. They convert each message into a logical description in a flawless form. For an effective confirmation of correctness, the certainty (belief) of the sender (signer) and receiver (verifier) should fulfill the procedure objectives. They assume that the verification step of the authentication is accomplished among the signer (Alice) and the receiver (Bob) if there is a message "X" that the receiver Bob is convinced that it has been sent by the sender, Alice. Therefore, the authentication process between Alice and Bob is accomplished if Bob | ≡ Alice | ≡ X and Bob | ≡ X, where the representation | ≡ means to believe (or be convinced by) [8]. The fundamental rules of BAN logic are as follows:

The interpretation rule

$$\frac{Bob|\equiv (Alice|\sim (X, Y))}{Bob|\equiv (Alice|\sim X),\ Bob|\equiv (Alice|\sim Y)} \tag{9}$$

This rule states that if Bob believes (convinced by the protocol) that Alice generates a message containing both X and Y, he believes that Alice generates each message distinctly.

Message Meaning Rule

$$\frac{Bob|\equiv \overset{Q-Alice}{\longrightarrow} Alice,\ Bob \triangleleft [X]_{S-Alice}}{Bob|\equiv Alice|\sim X},\ Alice \neq Bob \tag{10}$$

This rule states that if Bob believes (convinced by the protocol) that Q_{Alice} is the public key of Alice and Bob obtains a statement X that is signed by Alice's secret key S_{Alice}, this indicates that Bob believes that Alice once generated X.

Nonce Verification Rule

$$\frac{Bob|\equiv \#(X),\ Bob|\equiv Alice \sim X}{Bob|\equiv Alice|\equiv X} \tag{11}$$

This rule states that if Bob believes (convinced by the protocol) that X is a new statement and Alice once generated X, it believes that Alice believes in X.

Jurisdiction Rule

$$\frac{Bob|\equiv Alice \Rightarrow X,\ Bob|\equiv Alice|\equiv X}{Bob|\equiv X} \tag{12}$$

This rule states that if Bob believes (convinced by the protocol) that Alice has jurisdiction (authority) over X and Bob believes that Alice believes in X, Bob believes in X.

Freshness Rule

$$\frac{Bob|\equiv \#(X)}{Bob|\equiv \#(X, Y)} \tag{13}$$

This rule states that if Bob believes (is convinced by the protocol) in the freshness of X and Y, He believes in the freshness of each message. This study assumes that the statement is transmitted between the sender, Alice, and the receiver, Bob.

Authentication is accomplished between Alice and Bob if the following goals are accomplished:

$$\text{Goal 1:}\quad Bob|\equiv Alice|\equiv M_i \tag{14}$$

$$\text{Goal 2:}\quad Bob|\equiv M_i \tag{15}$$

where M_i characterizes the statement that is sent by Alice

- Alice first signs the statement (message) "M_i" using the encryption secret key. The result is then encrypted by making use of Bob's public-key of encryption in order to generate "C1".
- Alice transmits $C1 = \{\{\{M_i, n\}sk_{ea}\}pk_{eb}$ to Bob, where *n* is the nonce produced by Alice.

The analysis is completed by assuming the following:

$$Alice|\equiv \overset{pkeb}{\longrightarrow} Bob \tag{16}$$

$$Alice|\equiv \overset{pkea}{\longrightarrow} Alice \tag{17}$$

$$Bob|\equiv \overset{pkeb}{\longrightarrow} Bob \tag{18}$$

$$Bob|\equiv \overset{pkea}{\longrightarrow} Alice \tag{19}$$

$$Bob|\equiv Alice \Rightarrow M_i \tag{20}$$

$$Alice|\equiv \#n \tag{21}$$

$$Bob|\equiv \#n \tag{22}$$

Equation (16) shows that Alice is convinced that pk_{eb} is the public key for encryption. Equation (17) shows that Alice

is convinced that pk_{ea} is the public key for encryption. Equation (18) indicates that Bob is convinced that pk_{eb} is his public key for encryption: Eq. (19) shows that Bob is convinced that pk_{ea} is Alice's public key for encryption. Subsequently, Eq. (20) shows that Bob is convinced that Alice has (authority) jurisdiction over the sent statement (message). Finally, Eqs. (21) and (22) show that Alice and Bob are convinced that "n" is fresh (n changes every message). Given these assumptions, the messages transmitted during the initial phase were converted into a logical description. Finally, the basic rules of BAN logic were applied to the logical formulas. For more information about the BAN logic, refer to [8, 9]. Using the message $C1 = \{\{\{M_i,n\}sk_{ea}\}pk_{eb}$, Eq. (19) and the message meaning rule (Eq. (10)):

$$Bob|\equiv Alice|\sim (m,\ M_i) \tag{23}$$

However, *Alice* and *Bob* are convinced that n is fresh (Eqs. (21) and (22)). Thus, by applying the nonce verification rule (Eq. 11), the following equation is obtained:

$$Bob|\equiv Alice|\equiv M_i \tag{24}$$

According to the jurisdiction rule (Eq. (12)) using Eq. (20), the following result is obtained:

$$Bob|\equiv M_i \tag{25}$$

The above analysis assumes that Bob is honest. However, as previously mentioned, Bob can re-encrypt the part signed by Alice to produce C2 (Eq. (8)). The following assumptions were made in the logical analysis:

$$Eve|\equiv \stackrel{pkea}{\longrightarrow} Alice \tag{26}$$

$$Eve|\equiv Alice \Rightarrow M_i \tag{27}$$

$$Alice|\equiv \#n \tag{28}$$

$$Eve|\equiv \#n \tag{29}$$

Using message $C2 = \{\{\{M_i,n\}sk_{ea}\}pk_{Eve}$, Eq. (26) and the message meaning rule (Eq. (10)):

$$Eve|\equiv Alice|\sim (m,\ M_i) \tag{30}$$

However, *Alice* and *Eve* are convinced that n is fresh (Eqs. (28) and (29)). Thus, by applying the nonce verification rule (Eq. 11), the following equation is obtained:

$$Eve|\equiv Alice|\equiv M_i \tag{31}$$

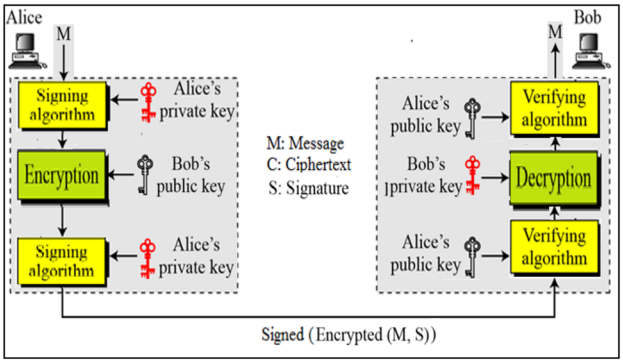

Fig. 2 The proposed modification to the NaCl library

According to the jurisdiction rule (Eq. (12)) using Eq. (27), the following result is obtained:

$$Eve|\equiv M_i \tag{32}$$

The resulting goal is not supposed to have been achieved by Alice. Thus, in the case of a dispute, Alice denies sending this message to Eve. The above attack is considered a violation of the authentication service, particularly the non-repudiation service. In the following section, a detailed description of the proposed modification is illustrated to mitigate the aforementioned security attacks.

3 Proposed modification to NaCl library

In the proposed modification, the main focus is on crypto_box and crypto_box_open functions. As mentioned in [7], the NaCl library does not achieve non-repudiation; therefore, these functions must be fixed. To avoid the attack that is described in the previous section, the message after encryption is signed. Thus, Alice sends the following message to Bob: $\{\{\{M\}sk_{alice}\}pk_{Bob}\}\ sk_{alice}$. This is illustrated in Fig. 2.

In the following subsections, a detailed description of the proposed modification is provided, followed by a BAN logic analysis.

3.1 The proposed mitigation

The proposed solution consists of three modules: gen_keypairs, crypto_authenc_sign, and crypto_dec_ver.

gen_keypairs is used to generate public and private keys for Bob to be used to verify the messages received from Alice. In addition, it is used to generate two key pairs for Alice. The first key pair is used in signature, and the latter is used for encryption. The reason separate key pairs are used for signature and encryption is to provide more security. When a private key is compromised for encryption, the attacker can

```
gen_keypairs:
Input: sk_sa, sk_ea, sk_eb
Output: pk_sa, pk_ea, pk_eb
{       pk_sa = crypto_sign_keypair(sk_sa)
        pk_ea = crypto_box_keypair(sk_ea)
        pk_eb = crypto_box_keypair(sk_eb)
}
```

Fig. 3 Pseudocode of gen_keypairs function

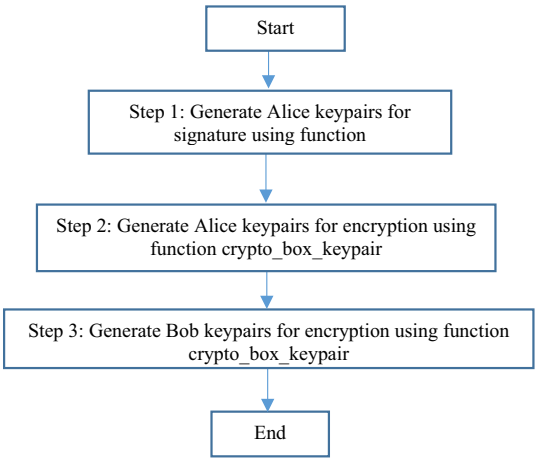

Fig. 4 Flowchart of gen_keypairs function

decrypt the transferred messages but cannot use it for signatures. First, genkeypairs generate the signature key pairs of Alice by using crypto_sign_keypair in the NaCl library. It, then, generates Alice's key pairs for encryption using the crypto_box_keypair in the NaCl library. The last step is to generate the Bob key pairs used for encryption. The pseudocode for this function (gen_keypairs) is depicted in Fig. 3. A flowchart of the gen_keypairs is shown in Fig. 4.

In Fig. 3, pk_{sa} is the public key for Alice's signature, sk_{sa} is the secret key for Alice's signature, pk_{ea} is the public key for Alice's encryption, sk_{ea} is the secret key for Alice's encryption, pk_{eb} is the public key for Bob's encryption, and sk_{eb} is the secret key for Bob's encryption.

crypto_authenc_sign is used to perform the cryptographic operations at the sender. First, the sender (Alice) signs the message "M" using her secret key for encryption "sk_{ea}," and encrypts the output using the public key for encryption of the receiver (Bob) "pk_{eb}" to produce the output ciphertext "c". This is performed using crypto_box in the NaCl library. To provide non-repudiation, a second step is performed. In this step, Alice signs the output ciphertext to produce "sm" using its secret signature key "sk_{sa}".

```
crypto_authenc_sign:
Input: M, n, pk_eb, sk_ea, sk_sa
Output: sm
{c = crypto_box(M,n,pk_eb,sk_ea)
sm = crypto_sign(c,sk_sa)
}
```

Fig. 5 Pseudocode of crypto_authenc_sign function

```
crypto_dec_ver:
Input: sm, pk_sa, n, pk_ea, sk_eb
Output: M
{c = crypto_sign_open(sm,pk_sa)
 M = crypto_box_open(c,n,pk_ea,sk_eb)
}
```

Fig. 6 Pseudocode of crypto_dec_ver function

This is performed using the crypto_sign function of the NaCl library. The pseudocode for this function (crypto_auth_sign) is depicted in Fig. 5.

crypto_dec_ver is used to perform cryptographic operations at the receiver. First, the receiver (Bob) verifies Alice's signature using the public key of the signature of Alice "pk_{sa}". This is performed using the crypto_sign_open function of the NaCl library. To retrieve M, Bob decrypts the output using its secret key of encryption "sk_{eb}," and then verifies Alice's signature using the public key of encryption of Alice "pk_{ea}". This is performed using the crypto_box_open function of the NaCl library. The pseudocode for this function (crypto_dec_ver) is depicted in Fig. 6. A flowchart of both crypto_authen_sign and crypto_dec_ver functions are shown in Figs.7 and 8.

3.2 Implementation of the modified NaCl library

The modified NaCl library is implemented using the C programming language and NaCl libraries. The code shown in Fig. 9 is used to compute the secret and public keys of the sender (Alice) for the encryption and signature. In addition, it generates the secret and public keys of the receiver (Bob) for encryption. The program output is as follows:

Step 1 (Fig. 4)**:** Alice's secret key for encryption:

```
0x7b,0x31,0x5b,0x94,0xfc,0xa6,0x66,0x85,0x15,0x3d,0xb1,0x5b,
0x58,0x89,0xc6,0xa6,0xc,0xa4,0xef,0x5e,0x42,0xb1,0xc7,0xe0,0x78,
0xd1,0xd2,0xd4,0x52,0x08,0xa2,0x46
```

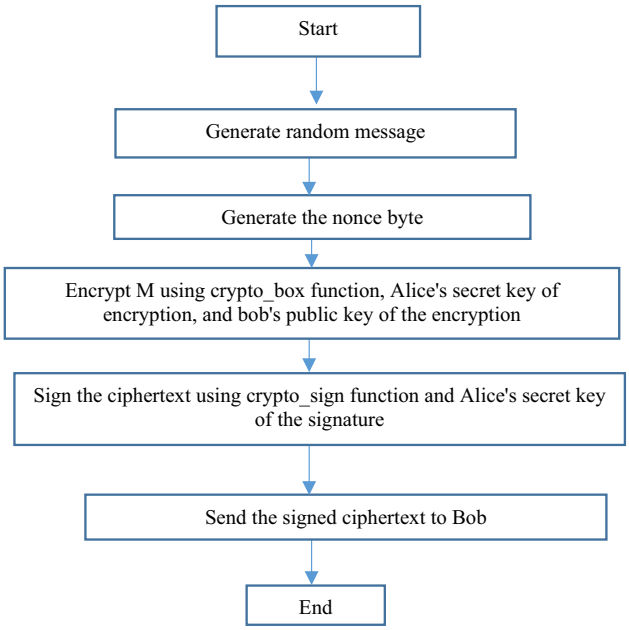

Fig. 7 Flowchart of crypto_authenc_sign function

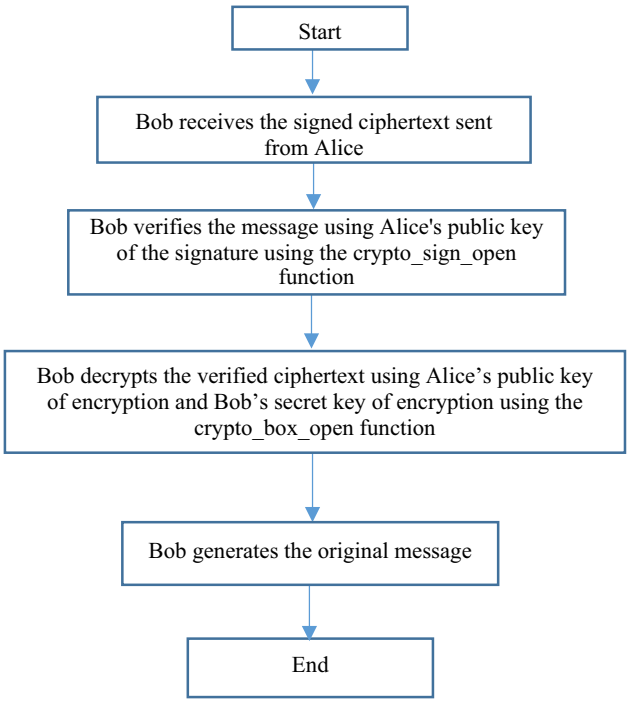

Fig. 8 Flowchart of crypto_dec_ver function

Alice's public key for encryption:

0xd4,0x12,0x0d,0x05,0x52,0x6d,0x68,0xe4,0xb5,0xb1,0xe6,0x8a,
0x9e,0x65,0x76,0x17,0x50,0xb4,0x1d,0x46,0xa9,0x10,0xb8,0xc7,
0x15,0x0c,0x9e,0xd8,0xdd,0xed,0xef,0x00

Step 2 (Fig. 4)**:** Alice's public key for signature:

0x9c,0x75,0xd8,0xb9,0xbe,0x75,0xf9,0x5e,0x91,0xcd,0x2d,0x9b,
0x12,0xc8,0x54,0xc7,0x30,0x77,0xb8,0x6f,0x9f,0xca,0x8f,0xf3,0x05,
0xe6,0x54,0x33,0xf0,0x78,0xc4,0xc8

Alice's secret key for signature:

0x08,0x5d,0xbb,0x34,0x00,0xba,0x8c,0x36,0x61,0xb8,0xa9,0x70,
0x5c,0xc4,0x7b,0x91,0x6e,0x4c,0x52,0x70,0xf8,0x2a,0x2a,0x52,
0x7b,0x1b,0xfd,0x26,0x80,0x41,0x19,0x69,0xb4,0xc9,0xcf,0x43,
0x61,0x89,0x30,0xbb,0x60,0x69,0x26,0x86,0xe4,0x1a,0x90,0xfe,
0x39,0xd9,0xdb,0xfb,0x8d,0x03,0x92,0xe7,0x9d,0x76,0x52,0x00,
0x59,0x8c,0x9a,0xd0

Step 3 (Fig. 4)**:** Bob's secret key for encryption:

0x51,0x65,0xca,0x35,0x22,0x34,0x6b,0x26,0xee,0xc7,0x64,0xce,
0xaa,0x4f,0x2b,0x10,0x21,0x65,0x50,0x3c,0x93,0xcd,0x1f,0x3f,0x7f,
0x90,0x8b,0x26,0x1b,0x0e,0x43,0xf9

Bob's public key for encryption:

0xff,0xca,0x3c,0x2c,0xee,0x6b,0x3e,0x60,0xc6,0x78,0x1c,0x6d,0x30,
0xb7,0xe1,0xad,0x0c,0xce,0x12,0x9e,0xb7,0xeb,0x7c,0x2c,0x32,
0xa4,0x70,0xf0,0x03,0x71,0x22,0x5a

The code shown in Fig. 10 is used to implement the crypto_authenc_sign function. This function is used by the sender (Alice) to sign the message using its secret key for encryption and then encrypts the output using the public key of encryption of the receiver. Finally, it signs an authenticated message using its secret key for the signature before sending the output to Bob. The generated random message before encryption and signing is as follows:

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xe2,0xc3,0x2e,0x3d,

0xbd,0x31,0xcc,0x7d,0x4c,0x63

The output is as follows after applying crypto_authenc_sign function:

0xce,0xd8,0x39,0x39,0x1b,0xee,0xb4,0xf2,0xc2,0xd0,0x9d,0xc8,
0xa8,0x5d,0xa9,0x70,0x45,0x0f,0xc9,0xbf,0x06,0x1e,0x03,0x27,
0x5b,0xc7,0x9c,0xc6,0xed,0x91,0x71,0x15,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x54,0xcc,0x4b,0x48,0x6b,0xaf,0x66,0x8a,0x02,0xe5,0xc0,0x46,
0xd4,0x7e,0xfa,0xdd,0x18,0x35,0x0a,0x6a,0x54,0x3e,0x3e,0xad,
0xdb,0xca,0x15,0x9a,0x17,0xa8,0x64,0x1b,0x24,0x16,0xd7,0x64,
0x4d,0x92,0x37,0xd4,0xbe,0x4a,0xb8,0xfd,0x9d,0xa9,0xe8,0xb6,
0xc0,0x93,0x1f,0xe3,0x1c,0xcb,0x23,0xe0,0xfc,0x05

The code shown in Fig. 11 is used to implement the crypto_dec_ver function. This function is used by the

Fig. 9 The C code used in the implementation of gen_keypairs function

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "crypto_box.h"
#include "randombytes.h"
#include "crypto_sign.h"
#include "crypto_hash.h"
#include "crypto_scalarmult_curve25519.h"

void gen_keypairs (unsigned char *ASpk, unsigned char *ASsk, unsigned
char *ACpk,
                unsigned char *ACsk, unsigned char *BCpk, unsigned char
*BCsk)
{
        int i=0;
        unsigned char h[crypto_hash_BYTES];
        crypto_sign_keypair(ASpk,ASsk);
        crypto_hash(h,ASpk,sizeof ASpk);
            for(i=0; i < crypto_box_SECRETKEYBYTES ; i++)
            {
                ACsk[i]=h[i];
            }
            crypto_scalarmult_curve25519_base(ACpk,ACsk);
            crypto_box_keypair(BCpk,BCsk);
}
```

```
void crypto_Enc_Sign(unsigned char *c,unsigned char *m,unsigned long long mlen, unsigned char *n,unsigned char *BCpk,unsigned char
*ACsk,
unsigned char *sm, unsigned long long *smlen, unsigned char *ASsk)
{
            int i=0;
            unsigned long long smlenR;
            clock_t begin1 = clock();
            crypto_box(c,m,mlen + crypto_box_ZEROBYTES,n,BCpk,ACsk);
            clock_t end1 = clock();
            double time_spent1 = (double)(end1 - begin1)*1000 / CLOCKS_PER_SEC;
            clock_t begin2 = clock();
            crypto_sign(sm,&smlenR,c,mlen + crypto_box_ZEROBYTES,ASsk);
            clock_t end2 = clock();
            double time_spent2 = (double)(end2 - begin2)*1000 / CLOCKS_PER_SEC;
            *smlen=smlenR;
}
```

Fig. 10 The C code used in the implementation of crypto_authenc_sign function

```
void crypto_Dec_Ver(unsigned char *m2,unsigned long long mlen,unsigned char *n,unsigned char *ACpk,unsigned char *BCsk,
unsigned  char *sm, unsigned long long smlen,unsigned char *ASpk)
 {
            int i=0;
   unsigned long long tmlength=0; // the length of the verified message
   unsigned char tm[10000]; // verified message array
   clock_t begin3 = clock();
            // verify the signed message
            crypto_sign_open(tm,&tmlength,sm,smlen,ASpk);
            clock_t end3 = clock();
            double time_spent3 = (double)(end3 - begin3)*1000 / CLOCKS_PER_SEC;
            clock_t begin4 = clock();
  if (crypto_box_open(m2,tm,mlen + crypto_box_ZEROBYTES,n,ACpk,BCsk) == 0)
   {
            printf(" \n \n ciphertext succeed  verification\n");
   }

   else {printf(" \n \n ciphertext fails verification\n");}
   clock_t end4 = clock();
   double time_spent4 = (double)(end4 - begin4)*1000 / CLOCKS_PER_SEC;
}
```

Fig. 11 The C code used in the implementation of crypto_dec_ver function

receiver (Bob) to verify and retrieve the original message. The program output is as follows after verification and decryption:

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xe2,0xc3,0x2e,0x3d,0xbd,0x31,0xcc,0x7d,0x4c,0x63

3.3 Logical analysis of the modified NaCl library using BAN logic

In this subsection, the logical analysis of the modified NaCl library is, first, introduced. Burrows et al. [8] introduced a logic analysis method called BAN logic which analyzes the authentication protocols, all primitives' shared keys, and public keys are formalized. BAN logic consists of a set of rules. These rules describe and investigate information exchange protocols. Specifically, BAN logic enables users to define the dependability and security of transmitted information. BAN logic begins with the notion that transmitted information is unprotected and subject to tampering and public observation. BAN logic enables one to simply reason cryptographic protocols using a formal methodology. The foundation of BAN logic is the certainty (belief) of a contended party in the certainty of a formalized prescription. A formalized prescription is incorrect in the general sense of certainty [9]. As mentioned above, authentication protocols are essential components in the blockchain, and it is necessary to ensure the correctness of these protocols. BAN logic uses logic to define authentication protocols. The steps applied to the initial NaCl library were performed for the modified proposal. First, each message was converted into a logical description. For an effective confirmation of correctness, the certainty (belief) of the sender (signer) and receiver (verifier) should fulfill the procedure objectives. BAN logic assumes that the verification step of the authentication is accomplished among the signer (Alice) and the receiver (Bob) if there is a message "X" such that the receiver Bob is convinced that it has been sent by the sender, Alice. Therefore, the authentication process between Alice and Bob is accomplished if Bob | ≡ Alice | ≡ X and Bob | ≡ X, where the representation | ≡ means to believe (or be convinced by). Authentication is accomplished between Alice and Bob if the following goals are accomplished:

$$\text{Goal 1:} \quad Bob|\equiv Alice|\equiv M_i \tag{33}$$

$$\text{Goal 2:} \quad Bob|\equiv M_i \tag{34}$$

where M_i characterizes the statement that is sent by Alice

- Alice first signs the statement (message) "M_i" using the encryption secret-key. Then, the result is encrypted by making use of Bob's public-key of encryption to generate "c". Subsequently, c is signed using Alice's secret signature key.
- Alice transmits $\{\{\{M_i, n\}sk_{ea}\}pk_{eb}\}sk_{sa}$ to Bob, where n is a nonce produced by Alice.

The analysis is completed by assuming the following:

$$Alice|\equiv \xrightarrow{pkeb} Bob \tag{35}$$

$$Alice|\equiv \xrightarrow{pkea} Alice \tag{36}$$

$$Alice|\equiv \xrightarrow{pksa} Alice \tag{37}$$

$$Bob|\equiv \xrightarrow{pkeb} Bob \tag{38}$$

$$Bob|\equiv \xrightarrow{pkea} Alice \tag{39}$$

$$Bob|\equiv \xrightarrow{pksa} Alice \tag{40}$$

$$Bob|\equiv Alice \Rightarrow M_i \tag{41}$$

$$Alice|\equiv \#n \tag{42}$$

$$Bob|\equiv \#n \tag{43}$$

Equation (35) shows that Alice is convinced that pk_{eb} is the public key for encryption. Equation (36) indicates that Alice is convinced that pk_{ea} is the public key for encryption. Equation (37) shows that Alice is convinced that pk_{sa} is the public key for the signature. Equation (38) demonstrates that Bob is convinced that pk_{eb} is his public key for encryption: Eq. (39) shows that Bob is convinced that pk_{ea} is Alice's public key for encryption. Equation (40) shows that Bob is convinced that pk_{sa} is Alice's public signature key. Subsequently, Eq. (41) shows that Bob is convinced that Alice has (authority) jurisdiction over the sent statement (message). Finally, Eqs. (42) and (43) show that Alice and Bob are convinced that "n" is fresh (n changes with every message). Given these assumptions, the messages transmitted during the initial phase were converted into a logical description. Finally, the basic rules of BAN logic were applied to the logical formulas. For more information about the BAN logic, refer to [8, 9]. Using the message $\{\{\{M_i,n\}sk_{ea}\}pk_{eb}\}sk_{sa}$, Eq. (40) and the message meaning rule

$$Bob|\equiv Alice|\sim (n,\ M_i) \tag{44}$$

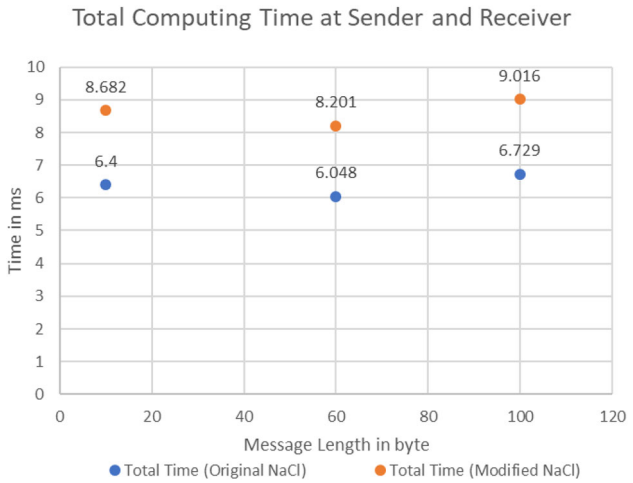

Fig. 12 Total computing time at sender and receiver

However, *Alice* and *Bob* are convinced that n is fresh (Eqs. (42) and (43)). Thus, by applying the nonce verification rule, the following is obtained.

$$Bob|\equiv Alice|\equiv M_i \tag{45}$$

According to the jurisdiction rule using Eq. (41), the following result is obtained:

$$Bob|\equiv M_i \tag{46}$$

From Eqs. (45) and (46), it is proven that the proposed modification to the NaCl library achieves the objectives of authentication (non-repudiation property) without pitfalls. This is due to the fact that Bob will not be able to re-sign the message rather than Alice. The use of BAN logic proves that the initial NaCl library cannot resist nonrepudiation attacks. On the other hand, the proposed modification fixes this issue and is proven by logical analysis.

4 Performance analysis

In this section, Fig. 12 and Table 1 provide a comparison of the execution times for both communicating parties (sender and receiver) in the case of the original NaCl and modified NaCl. The test environment consists of the following (the processor: 1.6 GHz Dual-Core Intel Core i5, Memory: 8 GB 1600 MHz DDR3, Development language: C, The IDE: Eclipse, the calculated time is in milliseconds (ms), and the calculated message length is in bytes). The modified NaCl consumes more time which is expected as adding a new block of signature will increase the execution time by a range of 50% since the time is calculated for three blocks instead of two. However, the modified NaCl library achieves the non-repudiation security goal, as shown in Table 2. In Table 1, the total time represents the sum of time needed for the cryptographic operations at both the sender and the receiver.

Table 1 Total computing time at sender and receiver

/Message length (byte)	Total time (original Nacl) sender and receiver (ms)	Total time (modified NaCl) sender and receiver (ms)
1	5.406	8.08
2	5.176	7.271
3	5.684	7.836
5	5.895	8.349
6	5.339	7.596
7	5.725	8.497
9	6.535	8.869
10	6.4	8.682
30	6.118	8.853
50	5.422	8.005
100	6.729	9.016
400	4.618	6.736
1000	4.968	7.05
5000	4.982	7.124
9000	4.865	7.434

Table 2 Security goals in original NaCl and modified NaCl

Security Goal	Confidentiality	Integrity	Non-Repudiation
Original NaCl	Yes	Yes	No
Modified NaCl	Yes	Yes	Yes

Example of the running code while the message length is 10 bytes:

Alice's private key for encryption:

0x0b,0xc7,0x3e,0xae,0xfb,0xd5,0x40,0xc2,0x48,0x9f,0xa3,0x62,0xe2,
0xb2,0xaa,0xb7,0xc6,0x73,0x42,0xd7,0xd3,0xd3,0xfd,0x12,0x24,
0x47,0x64,0xee,0x30,0x7b,0x57,0xbb

Alice's public key for encryption:

0x97,0x69,0x63,0x32,0xa7,0x7c,0xd6,0x4e,0xe4,0xd5,0x4c,0x8c,
0xd1,0x2f,0x8d,0x6f,0x8c,0x50,0xd0,0x16,0xd2,0x3c,0xe0,0x7e,
0x42,0x33,0xa1,0xf7,0x98,0x25,0x5c,0x47

Bob's private key for decryption:

0xba,0xb3,0x3e,0xdc,0x5f,0x96,0x62,0x88,0xaf,0x07,0xaf,0xa5,0xd3,
0x92,0x79,0x0e,0xb6,0x2d,0x7b,0xee,0xa0,0x41,0xc0,0x5b,0x1a,
0x0e,0x5f,0x2f,0x76,0x5f,0x2c,0x53

Bob's public key for decryption:

0x25,0x3c,0x87,0x2d,0x02,0x1d,0xd1,0x3c,0x7e,0x0b,0x56,0x30,
0xee,0x30,0xe9,0x17,0xce,0xeb,0xa8,0x2c,0x80,0x03,0x30,0x99,
0xb2,0xc2,0x53,0x52,0xaa,0xd9,0xd7,0x4d

Alice's public key for signature:

0xdb,0x73,0x6a,0x22,0xf0,0xb1,0xbd,0xad,0xf8,0x8c,0x14,0xe7,
0x85,0x85,0x79,0x59,0x57,0xa4,0x45,0x7f,0x7a,0x81,0x0b,0xed,
0xc3,0x66,0x1c,0x59,0x94,0x45,0xdd,0xd8

Alice's private key for signature:

0x68,0x17,0xb2,0xcb,0xae,0x41,0xdb,0x57,0xf3,0x64,0x97,0x9b,
0x17,0xc0,0x44,0x54,0xe7,0x57,0xc8,0x37,0x79,0x88,0x20,0x00,
0xf9,0x08,0x02,0x0b,0x41,0x87,0xc9,0x76,0xf0,0x96,0x14,0x05,
0x09,0xbf,0x12,0x61,0x70,0xac,0xfa,0x17,0x7d,0x86,0x1f,0xa9,
0xe2,0xf0,0xc9,0x92,0x25,0x5e,0x87,0x5c,0xe2,0x8f,0xe3,0x58,
0xad,0xac,0x5a,0x64

The generated random message before encryption and signing:

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x96,0x45,0x46,
0xf8,0x7d,0x1c,0xd6,0x3a,0x00

At the sender, the consumed time for signing a message (of length 10 bytes) using Alice's secret key of encryption followed by encryption using Bob's public key of encryption is 0.115 ms. Then, the time of signing the message using Alice's secret key of signature is 1.992 ms. The sent message is as follows:

0x82,0x8c,0x49,0x27,0xbe,0xe5,0x59,0xe3,0xca,0xf6,0xc3,0x0c,0x84,
0xf6,0x0d,0x19,0xfd,0x49,0xff,0x1e,0x7d,0xcf,0xfc,0xe9,0xa7,0xa5,
0xe5,0xf6,0x28,0xae,0x16,0x2c,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xb0,0x26,
0x3b,0xc3,0x69,0x51,0x71,0x8a,0xf1,0x39,0x2f,0xee,0x76,0x2c,
0x20,0x6a,0x40,0xe2,0xef,0x52,0x95,0xc1,0xbe,0x4a,0xfc,0xf9,
0x1e,0xd6,0x64,0xd3,0xbb,0x7c,0x09,0xb1,0xce,0x3c,0xfd,0x8c,
0x1f,0x13,0x75,0xe7,0xd1,0x92,0xf8,0xe2,0x5a,0x5a,0xfd,0x7c,
0x12,0x38,0x67,0xda,0xa5,0xb0,0x74,0x08

At the receiver, the consumed time for verifying the received message using Alice's public key of signature is 0.1189 ms. Then, the time of decrypting the message using Bob's private key and verifying using Alice's public key of encryption is 5.963000 ms. The output message is:

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x96,0x45,0x46,
0xf8,0x7d,0x1c,0xd6,0x3a,0x00

Therefore, the total consumed-time for a message of length 10 bytes at both the sender and the receiver sides is 8.462000 ms.

5 Conclusion and future work

Blockchain provides solutions for several applications. Blockchain has the ability to provide the following characteristics: decentralization, anonymity, privacy, trust, and immutability. This makes it a solution for most inventions in the information and communication technology industry. It is a pillar of the fourth industrial revolution and it is a cryptographically demanding technology that is regarded as one of the most influential topics in academia. The security services that need to be delivered to design a secure application are confidentiality, integrity, authentication, and non-repudiation. These security services are provided using cryptographic algorithms that are the base of Blockchain functionality. Many blockchain platforms currently utilize third-party cryptographic libraries that offer many cryptographic primitives in order to ensure users' protection against cyber-attacks. NaCl and its forks are the most widely used cryptographic libraries. NaCl is known to be one of the best libraries that provide usability property. While NaCl is easy to use and guarantees: confidentiality, integrity, and authenticity, it fails to provide a non-repudiation service. In this paper, an improvement to the blockchain platforms is proposed by enhancing the NaCl library to achieve the non-repudiation property that is widely used in the Blockchain platform. In order to provide the aforementioned security services, NaCl operation is based on signing and encrypting the messages. Therefore, the non-repudiation service is not provided as proved by applying the BAN logic rules to the library. To deliver the non-repudiation property, we propose a solution that is based on adding a signature block. The modified library is analyzed using the BAN logic that proves the correctness of the proposed solution. The analysis suggests that the proposed solution fixes the aforementioned problem. Adding non-repudiation to a blockchain platform keeps law and order when it comes to technology misuse. In this paper, we introduce the modification that adds non-repudiation to blockchain platforms that are built on the NaCl library and its forks and rely on developers to revisit these forks in order to implement the suggested modification. Adding a signature block increases the execution time of the proposed modification since the time is calculated for three blocks instead

of two. In order to improve the performance of the implementation of the modified NaCl library, parallelization and pipelining techniques are suggested to be deployed in future work.

Author contributions MR: Conceptualization and Idea establishment; Funding Acquisition; Writing—Review & Editing; Coding: Formal Analysis; Methodology. MMN: Conceptualization and Idea establishment; Funding Acquisition; Writing—Review & Editing; Coding; Formal Analysis; Methodology. DA: Idea establishment; Funding Acquisition; Writing—Review & Editing; Coding; Formal Analysis; Methodology. AE; Coding. HA; Funding Acquisition; Writing—Review & Editing; Formal Analysis. The authors read and approved the final manuscript.

Funding The authors received no funding for this study.

Declarations

Conflict of interest The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf.
2. Wong, E.: Retrieving dispersed data from SDD-1: a system for distributed databases. In: 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, University of California, Berkeley, CA, USA, pp. 217–235 (1977)
3. Zheng, Z., Xie, S., Dai, H., Chen X., Wang, H.: An overview of blockchain technology: architecture, consensus, and future trends. In: 6th IEEE International Congress on Big Data, Honolulu, pp. 557–564 (2017)
4. Casinoa, F., Dasaklisb, T.K., Patsakisa, C.: A systematic literature review of blockchain-based applications: current status, classification and open issues. Telematics Inform. **36**(1), 55–81 (2019)
5. Colomo-Palacios, R., Sánchez-Gordón, M., Arias-Aranda, D.: A critical review on blockchain assessment initiatives: a technology evolution viewpoint. Softw. Evolut. Process **32**(11), 1–11 (2020)
6. Storublevtcev, N.: Cryptography in blockchain. In: International Conference on Computational Science and Applications ICCSA2019, Saint Petersburg, Russia, pp 495–508 (2019)
7. Bernstein, D.J., Lange, T., Schwabe, P.: The security impact of a new cryptographic library. In: International Conference on Cryptology and Information Security in Latin America LATINCRYPT 2012: Progress in Cryptology—LATINCRYPT 2012, Santiago de Chile, Chile, pp. 159–176 (2012).
8. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. In: Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, London, UK, pp. 233–271 (1989)
9. Wessels, J.: Application of BAN-logic. Technical report, CMG Public Sector B.V (2001). http://www.win.tue.nl/ipa/archive/springdays2001/banwessels
10. Gamage, H.T.M., Weerasinghe, H.D., Dias, N.G.J.: A survey on blockchain technology concepts, applications, and issues. SN Comput. Sci. **1**, 114 (2020). https://doi.org/10.1007/s42979-020-00123-0
11. wolfSSL Inc. wolfCrypt Embedded Crypto Engine. https://www.wolfssl.com/products/wolfcrypt/. Last Accessed 21 Feb 2023
12. Cifra. A collection of cryptographic primitives targeted at embedded use. https://github.com/ctz/cifra. Last Accessed 21 Feb 2023
13. Intel Corporation. TinyCrypt Cryptographic Library. https://github.com/intel/tinycrypt. Last Accessed 21 Feb 2023
14. Ken MacKay. micro-ecc. http://kmackay.ca/micro-ecc/. Last Accessed 21 Feb 2023
15. https://github.com/brix/crypto-js. Last Accessed 21 Feb 2023
16. D. F. Aranha, C. P. L. Gouvˆea, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic. Last Accessed 21 Feb 2023
17. https://bouncycastle.org/java.html. Last Accessed 21 Feb 2023
18. Ferrag, M.A., Shu, L.: The performance evaluation of blockchain-based security and privacy systems for the internet of things: a tutorial. IEEE Internet Things J. **8**(24), 17236–17260 (2021)
19. PBC Library. http://crypto.stanford.edu/pbc. Last Accessed 21 Feb 2023
20. Crypto++ Library. https://www.cryptopp.com/. Last Accessed 21 Feb 2023
21. Miracl Library. https://github.com/miracl/MIRACL. Last Accessed 21 Feb 2023
22. Hyperledger Ursa. https://www.hyperledger.org/projects/ursa. Last Accessed 21 Feb 2023
23. OpenSSL https://github.com/openssl/openssl. Last Accessed 21 Feb 2023
24. Acar, Y., Backes, M., Fahl, S., Garfinkel, S., Kim, D., et al.: Comparing the usability of cryptographic APIs. In: 2017 IEEE Symposium on Security and Privacy (SP), California, CA, USA, pp. 154–171 (2017). https://doi.org/10.1109/SP.2017.52
25. Grabovský, M.: Measuring the usability of cryptographic libraries. Bachelor's Thesis, Masaryk University Faculty of Informatics, Brno, Czech Republic, Spring (2018)
26. Keck, P.: Analysing and improving the crypto ecosystem of Rust. Master's thesis, Institute of Software Technology, University of Stuttgart, Stuttgart, Germany (2017)
27. Silde, T.: Comparative study of ECC libraries for embedded devices. Technical report, Norwegian University of Science and Technology (2019)
28. Balamurugan, C., Singh, K., Ganesan, G., Rajarajan, M.: Code-based post-quantum cryptography. Preprints 2021, 2021040734. https://doi.org/10.20944/preprints202104.0734.v1 (2021)
29. Zinzindohou, J., Bhargavan, K., Protzenko, J., Beurdouche, B.: HACL*: A verified modern cryptographic library. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1789–1806 (2017)
30. https://github.com/jedisct1/libsodium. Last Accessed 21 Feb 2023
31. https://tuweni.apache.org/docs/org.apache.tuweni.crypto.sodium/-lib-sodium/index.html. Last Accessed 25 Feb 23.
32. https://duniter.org/. Last Accessed 25 Feb 2023
33. https://pytezos.org/contents.html. Last Accessed 25 Feb 2023
34. https://z.cash/. Last Accessed 25 Feb 2023

35. https://github.com/MercerWeiss/SilentDragon. Last Accesses 25 Feb 2023
36. Patnaik, N., Hallett, J., Rashid, A.: Usability smells: an analysis of developers' struggle with crypto libraries. In: Proceedings of the Fifteenth Symposium on Usable Privacy and Security (sponsored by USENIX), Santa Clara,California, CA, USA, pp. 245–258 (2019)
37. Wohlwender, J., Huesmann, R., Heinemann, A., Wiesmaier, A.: Cryptolib: comparing and selecting cryptography libraries. (long version of EICC 2022 publication) (2022). arXiv:2203.16370.
38. Namasudra, S., Sharma, P.: Achieving a decentralized and secure cab sharing system using blockchain technology. IEEE Trans Intell Transp Syst (2022)
39. Lopez, M.A., Terron, S., Lombardo, J.M., Gonzalez-Crespo, R.: Towards a solution to create, test and publish mixed reality experiences for occupational safety and health learning: training-MR (2021)
40. Namasudra, S., Sharma, P., Crespo, R.G., Shanmuganathan, V.: Blockchain-Based Medical Certificate Generation and Verification for IoT-Based Healthcare Systems. In: IEEE Consumer Electronics Magazine, vol. 12, no. 2, pp. 83–93, 1 March 2023, https://doi.org/10.1109/MCE.2021.3140048.
41. García-Peñalvo, F., Vázquez-Ingelmo, A., García-Holgado, A., Sampedro-Gómez, J., Sánchez-Puente, A., Vicente-Palacios, V., Dorado-Díaz, P.I., Sánchez, P.L.: Application of artificial intelligence algorithms within the medical context for non-specialized users: the CARTIER-IA platform (2021)
42. Sangjukta, D., Namasudra, S.: MACPABE: Multi-Authority-based CP-ABE with efficient attribute revocation for IoT-enabled healthcare infrastructure. Int. J. Netw. Manag. (2022)