IEEE ICBC'24 1570981950

OTTx: One-Time Transactions Tokens for Blockchain Services

***Abstract*—In business computer networks, secure information exchange among organizations with diverse objectives is common. Users, identified uniquely, access resources through reliable applications. Integrating applications from external organizations introduces untrusted elements and requires necessitating mechanisms to ensure privacy, integrity, and non-repudiation. This work proposes a One-Time Transactions (OTTx) protocol in blockchains. It addresses the need for secure transaction authentication, reviews the state of the art, and evaluates OTTx in a permissioned blockchain. Results show that OTTx ensures security, privacy, integrity, and non-repudiation for transactions involving external identities, with satisfactory performance and low network overhead. This contribution advances blockchain knowledge by providing an effective solution for transactions through untrusted network participants.**

***Index Terms*—blockchain, permissioned network, authentication, security, privacy, integrity, non-Repudiation, smart Contracts, one-time password.**

I. Introduction

The term blockchain refers to a decentralized and distributed ledger that records transactions in interconnected and encrypted blocks [1]. This structure offers increased security, accountability, transparency, and records cannot be retroactively altered without network consensus.

The permissioned form of blockchain, also known as consortium blockchain, is a type of blockchain designed specifically for use in private business environments [6]. In permissioned blockchain networks, maintainers can control access in a customized manner, and there is a need to allow the submission of transactions through applications of organizations that do not possess blockchain infrastructure and communicate with the network through some identity. For example, in networks with Application Programming Interfaces (APIs), organizations using these APIs do not have their blockchain infrastructure but interact with the network using the API provider identity. In this context, additional mechanisms are required to enable these external organizations to provide services to other users without tampering with their identities, and ensure integrity, non-repudiation, and data privacy in third-party transactions.

The use of authentication methods with one-time use tokens, such as *One-Time Password (OTP)* [18], in smart contracts [11] [12] [13] and integration with transaction submission [9], were proposed as solutions to enable authentication using blockchain. However, these approaches do not secure the submission of transactions through third parties, ensuring non-repudiation and data privacy.

This work proposes a protocol for one-time transactions called *OTTx (One-Time Transactions)* for blockchain networks. A specific approach is adopted, consisting of creating a dedicated smart contract for authentication functions and enabling communication between smart contracts. The study also compares the security and efficiency of the OTTx protocol with other related works and experiments, providing relevant data for future research in this area. By highlighting the potential of blockchain technology to overcome challenges associated with third-party authentication in transactions, this work aims to contribute to significant advancements in this field.

II. Related Works

While specific articles on authenticated transactions in blockchain are limited, various research on blockchain authentication forms a foundation for authenticated or one-time transactions. In one study [9], the introduction of a time-based One-Time Password (TOTP) scheme for two-factor authentication (2FA) in Hyperledger Fabric blockchains was proposed. However, the focus is primarily on the vulnerability of Hyperledger's access tokens to interception. This work maintains centralization through a proposed server and addresses implementations of the discontinued version of Hyperledger Fabric. Despite implementing transactions with tokens, it does not cover transactions sent from other identities.

In [11], authors propose an interorganizational authentication/authorization/accounting system preserving privacy using blockchain. This system, designed for a public blockchain, stores encoded passwords on the blockchain. However, the authorization scheme is limited to user access rules for network functionalities, excluding third-party identities' access to user data. The article by [12] describes an authentication method based on blockchain with a one-time password, but it only details the authentication method with OTP and permissioned blockchain, omitting secure submission of transactions from other identities.

Lastly, [13] introduces a microservices and blockchain-based protocol for enhanced one-time password authentication (*MBB-OTP*). While limited to authorizing network users, the use of shared secrets and multiple microservices increases computational expenses and vulnerabilities. Additionally, it prolongs the authentication process.

III. Background

This section covers fundamental concepts related to this work.

A. Blockchain

Blockchain, originally proposed by Nakamoto [1], is a form of Distributed Ledger Technology (DLT) that organizes timestamped data blocks in chronological order through hash values. DLT, employing cryptography, advanced algorithms, and significant computing power, revolutionizes transaction recording in a collaborative digital database [28]. This article focuses on blockchain-type DLT, emphasizing essential characteristics for the proposed solution. The cryptographic functions embedded in blockchain ensure attributes like immutability, integrity, and auditability, addressing Byzantine faults, double spending, and trust establishment in decentralized systems [23].

User authentication in this context relies on cryptographic keys, introducing an additional layer of security. Each user possesses a pair of keys: a publicly accessible public key and a confidential private key. These keys create a unique and immutable digital signature, validating transaction authenticity.

For a transaction, a user digitally signs it with their private key, and the blockchain network verifies this signature using the corresponding public key. This decentralized verification involves multiple distributed nodes.

The security of blockchain authentication hinges on users safeguarding their private keys. Proper storage in a secure environment, protected against unauthorized access, is essential.

Blockchain technology can be complemented with other methods like 2FA for enhanced security. Adding verification layers, such as authentication devices or physical tokens, strengthens transaction protection, even from untrusted third parties.

Understanding the proposed blockchain network architecture requires exploring different blockchain categories [26]. Public blockchains, like Bitcoin and Ethereum, are fully decentralized and open. Private blockchains restrict participants to specific groups, ensuring closed and secure transaction management. Permissionless blockchains offer complete decentralization and user autonomy.

In this work, the adopted blockchain network is permissioned for authorization control [25], with hybrid features allowing public access for transaction verification and authorization records. This hybrid approach optimizes transaction encryption for data security while enabling transparency and public validation, reflecting a deliberate balance between security and transparency for a robust solution in the proposed scenario [27].

B. Authtentication Methods

Passwords continue to be the predominant method of user authentication on the internet [20]. Although various password replacement schemes have been proposed, none have managed to compete in terms of deployment ease and usability of passwords [19]. In an attempt to enhance account security and safeguard them against compromises, major service providers such as Google, Facebook, and Microsoft have implemented an optional layer of 2FA in their authentication systems. Two-factor authentication requires users to provide two different types of authentication factors, chosen from the following options:

- **Knowledge Factor**: This involves something the user knows, such as a password, PIN, or response to a security question.
- **Possession Factor**: This encompasses something the user possesses, such as a physical token, smart card, or mobile device.
- **Inherence Factor**: This refers to something inherent to the user, such as biometric data (fingerprint, facial recognition, iris scanning).

By requiring the combination of two distinct authentication factors, two-factor authentication aims to significantly strengthen user account security. It mitigates the risks associated with single-factor authentication, where compromise of a single factor (e.g., password theft) can lead to unauthorized access.

C. One-Time Passwords (OTP)

The OTP concept, initially proposed by Lamport, generates unique passwords that continuously change with time or usage. In the authentication protocol, the system acts as the OTP verifier, authenticating users on behalf of various service providers. Service providers only need to forward the OTP token (the unique password sent by the client) and accept the authentication result, reducing their burden in the registration phase. This simplifies the deployment of authentication solutions and enables users to access multiple services with a single identity.

OTP enhances security by providing unique and dynamically changing passwords, making unauthorized access difficult even if one password is compromised. Combining OTP with other authentication factors like knowledge, possession, or inherence further strengthens authentication efficiency.

It is crucial to implement OTP securely, including measures like password storage, protection against brute-force attacks, and safeguarding the OTP secret, to ensure the effectiveness of the authentication process.

The OTP scheme proposed by Lamport involves building a hash chain using a recursively applied one-way cryptographic function to a secret key or password (seed). The resulting hash values serve as OTPs in reverse order [14].

During registration, the user provides a password (pw), and the verifier stores the hash value $y_0 = H_n(pw)$ as an OTP commitment value. For authentication, the verifier uses pw to generate the OTP token $y_i = H_{n-i}(pw)$ and verifies $y_{i-1} = H(y_i)$ against the stored commitment value y_{i-1}. A successful verification updates the commitment value from y_{i-1} to y_i for the next authentication. Lamport's OTP scheme has a limitation with finite-length hash chains.

OTP provides secure authentication without storing a registered key on the server. Time-Based One-Time Password (TOTP) [22] and HMAC-Based One-Time Password (HOTP) [21] enhance OTP features and cannot be adopted on the blockchain without trusting network maintainers. Hence, this

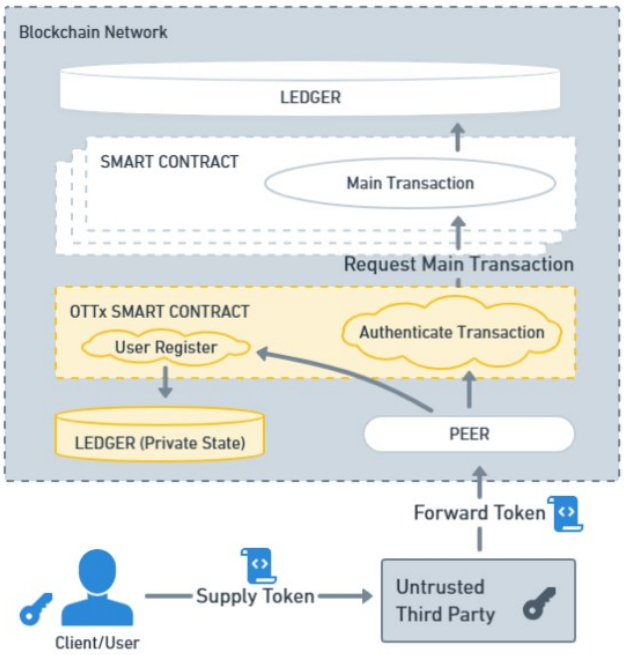

Fig. 1. OTTx protocol overview

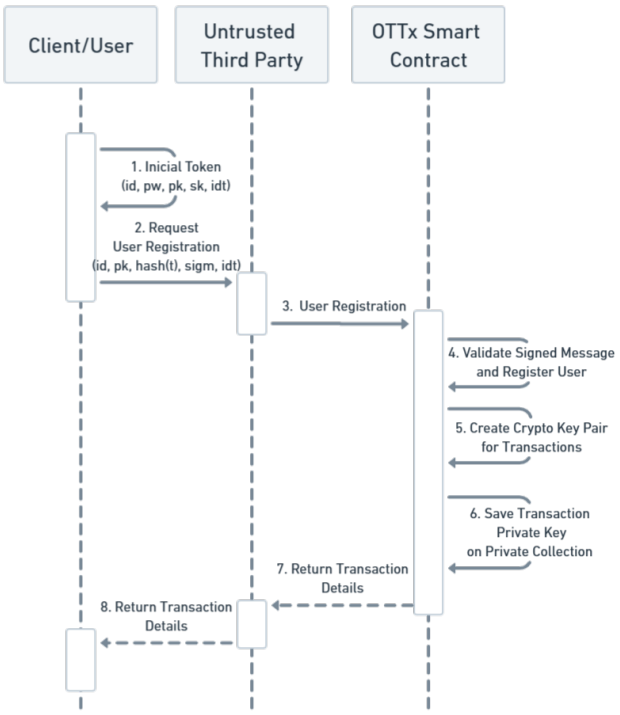

Fig. 2. OTTx User Register Sequence Diagram

work utilizes the OTP method in constructing the proposed solution.

IV. Proposed Protocol

Our proposal addresses challenges related to transaction data privacy by means of the implementation of a unified contract for authenticated transactions. We elaborate on two key aspects: a singular contract for submitting authenticated transactions and the recording of private data. These elements require implementations in the User token generation algorithm, the User registration function of the smart contract, the transaction authentication function of the contract, and network configuration. The overarching goal of this proposal is to improve security in transaction submissions involving untrusted identities by strengthening data privacy and consolidating authentication functions within a single contract.

A. Overview

The proposed scheme focuses on blockchain network modifications (Figure 1). In this permissioned blockchain network, Users generate tokens for transactions executed through Untrusted Third Parties.

Figure 2 illustrates the user registration sequence in the OTTx Smart Contract. The User, possessing public and private keys, (1) generates the initial token using the token generation algorithm, (2) requests user registration from the Untrusted Third Party, which then (3) initiates user registration with the OTTx Smart Contract.

The initial token generation involves user information, a cryptographic key pair (pk, sk), identifier (id), and password (pw). An algorithm combines these to generate a cryptographic hash as a token. Additionally, the algorithm produces the hash of the token and a signed message (sigm) containing relevant information.

To (2) request user registration, the User explicitly informs the third party of their public key (pk), identifier (id), hash of the generated token (hash(t)), signed message (sigm), and third party's identifier (idt). To prevent tampering attempts, the signed message verification ensures the provided data matches.

In step (3), the Untrusted Third Party calls the User Registration Transaction of the OTTx Smart Contract. The validation and registration phase begins with (4) verifying the signed message against the data passed to the contract. If valid, (5) the contract creates a key pair for transactions, used for asymmetric encryption in future transactions. In (6), the writing key, user public key, and token hash are stored in a private data collection accessible only to authorized peers and organizations. Finally, (7) the contract returns public transaction data, and (8) the third party must return the public key for transactions to the user.

Figure 3 displays the sequence diagram for transaction authentication. The sequence starts with (9) the User/Client encrypting transaction data (txData) using the transaction key (tpk) from the registration process. The User then (10) generates tokens for transactions, similar to registration, but including encrypted transaction data in the signed message. The User (11) requests the transaction from the Untrusted Third Party, passing necessary data. The third party (12) calls the Transaction to Authenticate Transactions of the OTTx

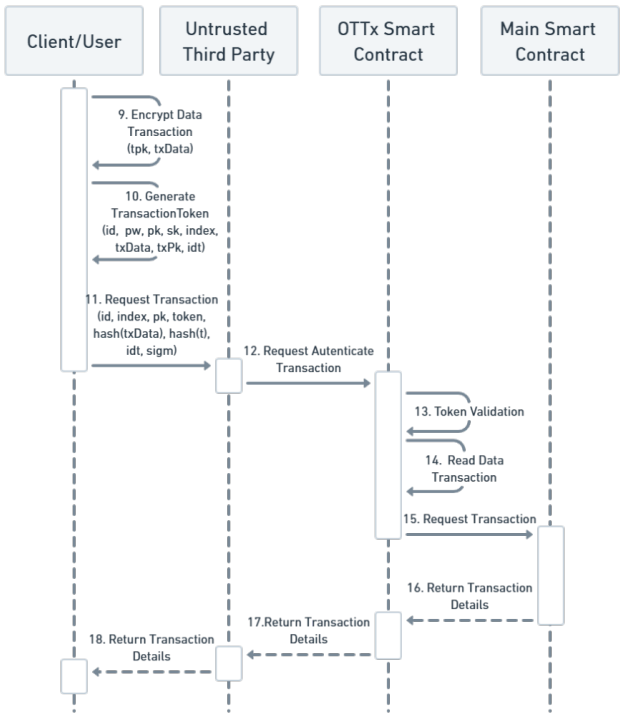

Fig. 3. OTTx Authenticate Transaction Sequence Diagram

Smart Contract for token validation and data writing in other smart contracts.

The transaction validation phase begins with (13) validating tokens and the signed message, checking the user identifier's existence and correct index. If no errors, (14) the contract searches the private collection for the reading key to decrypt the transaction data, which is (15) sent directly to the main contract.

The OTTx protocol was implemented and tested using Hyperledger Fabric [7] v2.0 in a private permissioned setup. Developed in *Node.js*, the protocol involves network configurations, Client/User token generation algorithms, and a singular smart contract for authentication. Organizations needing access to private data must be designated during network creation.

B. Token Generation

For the User, algorithms for token generation are required. These functions are based on the works of [14] and [12], utilizing the *SHA256 hash* functions and the *RSA-SHA256* signature algorithm. The implementation and testing of the client-side were done in *JavaScript*, using the *Node.js* runtime environment and the *crypto*[1] and *keypair*[2] packages.

```
1   Input: id, pw, pk, sk, idt, index, txData, tpk
2   Output: token, hashToken, signedMessage
3   if index == 0 then
4       token = hash({ id, index, pw, sk })
5       hashToken = hash(token)
6       message = { id, pk, hashToken }
7       signedMessage = sign(sk, message);
8       return { token, hashToken, signedMessage }
9   else
10      nextToken = hash({ id, index, pw, sk })
11      hashToken = hash(nextToken);
12      hashTxData = publicEncrypt(pkt, txData);
13      message = { id, pk, hashToken, hashTxData,
            idt };
14      signedMessage = sign(sk, message);
15      token = hash({ id, index: index - 1, pw, sk
            });
16      return { token, hashToken, signedMessage };
17  end if
```

Listing 1. OTTx Tokens Generation Algorithm

To generate tokens, the user must always pass the current index value as a parameter so that the correct tokens are generated. If the User does not have a record yet, they should pass the value zero (0) for the index to create the initial token. Therefore, the token generation function shown in Code Listing 1 begins by checking the index value to decide whether to generate an initial token (index = 0) or not. For initial tokens, the function generates a hash (*SHA256* [16]) by combining the user's identifier (id), index (index), password (pw), and private key (sk). After generating this first hash (token), the function generates a new hash (hashToken), this time passing the previous hash as a parameter. Thus, the second generated hash (hashToken) will be stored in the blockchain to be used as a verifier in the next request and should be updated by the next hash in the next call. In addition to these two hashes, the function generates a signed message (signedMessage) by combining the user's identifier (id), public key (pk), and the hash of the token (hashToken) using the *RSA-SHA256* algorithm [17]. The signed message will be used to maintain the integrity of information publicly passed to the smart contract.

For index values greater than zero (0), the token generation function shown in Code Listing 1 also generates, as previously described, a new token (nextToken), the hash of this new token (hashToken), and the signed message (signedMessage). However, this time, the hash is generated using asymmetric encryption [10] (publicEncrypt) of the transaction data (hashTxData), using the public key for transactions (pkt) generated in the registration transaction. The encrypted transaction data (hashTxData) is included in the signed message along with the identifier of the untrusted third party (idt). This way, the untrusted third party cannot read the transaction data, and the token cannot be sent by another identity, preventing replay attacks.

C. Smart Contract

Another contribution of this work is the implementation of a custom smart contract for the transaction authentication process. To achieve this, the contract must include functions for user registration and token validation, as well as facilitate communication with other contracts.

```
1   Input: id, pk, hashToken, signedMessage, idt
```

[1]https://www.npmjs.com/package/crypto-js
[2]https://www.npmjs.com/package/keypair

```
2  Output: true or false
3  if id exists then
4      return false
5  else
6      message = { id, pk, hashToken, idt}
7      if V(pk, signedMessage, message) === true
           then
8          txPublicKey, txPrivateKey =
               generateKeyPairSync()
9          save private {id, pk, txPublicKey,
               txPrivateKey}
10         save {id, pk, hashToken, index = 1,
               txPublicKey}
11         return true
12     else
13         return false
14     end if
15 end if
```

Listing 2. OTTx Smart Contract User Registration Algorithm

Code Listing 2 presents the algorithm for the user registration function in the authentication smart contract. The function takes the user identifier (id), their public key (pk), the hash of the initial token (hashToken), the signed message, and the identifier of the untrusted third party (idt). With this data, the contract checks if the user identifier already exists. If not, it verifies the validity of the signed message with the user's public key (V(pk, signedMessage, message)). If validated, the contract generates new keys for asymmetric encryption of transaction data (generateKeyPairSync), and the private key is securely saved using the Hyperledger Fabric's functionality to create private data collections, along with the user's identifier (id), public key (pk), transaction public key (txPublicKey), and transaction private key (txPrivateKey). Subsequently, the contract records the user's identifier (id), public key (pk), hash of the token (hashToken), index (index) set to one (1), and the transaction public key (txPublicKey) in the blockchain. The privately saved data is not exposed to any party outside the authorized organizations at the time of network creation. With these registered data, the user can start sending transactions for authentication in the contract.

```
1  Input: index, id, pk, token, hashToken,
       hashTxData, signedMessage, idt
2  Output: true or false
3  if id exists then
4      get {HASHTOKEN, INDEX} where ID = id
5      message = {id, pk, hashTxData, hashToken, idt
           }
6      if HASHTOKEN = hash(token) and verifyMessage(
           pk, signedMessage, message) === true
           then
7          if INDEX === index
8              get private {TXPRIVATEKEY} where ID =
                   id
9              txData = privateDecrypt(txPrivateKey,
                   hashTxData)
10             invokeChaincode(txData.chaincode,
                   txData.args, txData.channel)
11             update {id, pk, nextToken, index,
                   txPublicKey}
12             return true
13         else
14             return false
15         end if
16     else
17         return false
18     end if
19 else
20     return false
21 end if
```

Listing 3. OTTx Smart Contract Transactions Algorithm

Code Listing 3 illustrates the implementation of the function for token authentication in the smart contract. This function is responsible for verifying the validity of tokens, updating user data, and invoking the original transaction requested by the user. The function checks if the user identifier exists, if the sent token hash (hash(token)) is equal to the one stored in the blockchain (HASHTOKEN), verifies the signed message, and ensures the correct token index. If all validations pass, the contract retrieves the user's private transaction key (TXPRIVATEKEY) and decrypts the transaction data to send it to the requested original contract using the function to invoke other contracts on the Hyperledger Fabric network (invokeChaincode). Finally, the contract updates the user's data registered in the blockchain to include the hash of the next token (hashToken) and increments the index by one unit.

V. Security Analysis

The previous section presented the implementation for authenticated transactions in a permissioned blockchain. Next, an evaluation of these implementations is carried out. This assessment encompasses aspects of security and efficiency of the solution. Additionally, improvements related to previous works on authentication are listed and discussed. The evaluations demonstrate that the proposal put forth by this work has made the solution more competitive than previous ones, as well as more secure and versatile for various scenarios.

This section aims to analyze the effectiveness of the proposed solution concerning different types of attacks and security challenges. The evaluation is divided into five subsections, each addressing a specific aspect of the solution. The first subsection, "Adversarial Model," presents the adversarial model that encompasses the limitations of the security analysis. The second subsection, "Resistance to Replay Attacks," explores how the solution handles attacks attempting to reuse tokens in subsequent identities or requests. The third subsection, "Resistance to Brute-Force Attacks," discusses how asymmetric encryption and the token hash generation process protect against attacks attempting to guess the private key. The fourth subsection, "Token Forgery," examines the solution's ability to prevent token forgery, considering the security properties of encryption and hash functions. The fifth subsection, "Data Confidentiality in Transactions," addresses user data protection through asymmetric encryption and restricted access to private keys. Finally, the sixth subsection, "Identity Recertification," explores the implementation of a centralized contract for registration and authentication, ensuring identifier uniqueness and preventing recertification attacks.

A. Adversary Model

The proposed protocol assumes that an adversary has access to the blockchain network and can obtain stored information,

including the hash of private transaction data in both used and unused tokens. However, the adversary cannot compromise or obtain the privately stored private key on the client securely. To ensure message integrity and prevent man-in-the-middle attacks, communications are protected by SSL/TLS [15]. The adversary also cannot modify transaction data stored in the blockchain, but their goal is to read or modify requested transaction data before it is validated by blockchain nodes.

The proposal aims to ensure transaction data privacy, resistance to brute-force attacks, no leakage of transaction data in contract-to-contract communication, tamper-proof transaction data, and resistance to dictionary attacks on private data. These criteria will be used to evaluate the proposed solution and measure its effectiveness against potential adversary attacks.

This work utilized the Hyperledger Fabric v2.0 framework to implement the network and conduct tests only in the configuration of a private permissioned blockchain network. It also did not explore the use of different private keys for private data in each transaction, and it does not describe solutions for the scenario where the user does not receive or loses access keys.

VI. Evaluation of Proposed Solution

This section aims to analyze the effectiveness of the proposed solution concerning different types of attacks and security challenges. The evaluation is divided into five subsections, each addressing a specific aspect of the solution. The first subsection, "Resistance to Replay Attacks," explores how the solution handles attacks attempting to reuse tokens in subsequent identities or requests. The second subsection, "Resistance to Brute-Force Attacks," discusses how asymmetric encryption and token hash generation protect against attacks attempting to guess the private key. The third subsection, "Token Forgery," examines the solution's ability to prevent token forgery, considering the security properties of encryption and hash functions. The fourth subsection, "Data Confidentiality in Transactions," addresses user data protection through asymmetric encryption and restricted access to private keys. Finally, the fifth subsection, "Identity Recertification," explores the implementation of a centralized contract for registration and authentication, ensuring identifier uniqueness and preventing recertification attacks.

A. Resistance to Replay Attacks

Replay attacks can occur when the attacker has access to used or unused tokens, attempting to reuse them in other identities or subsequent requests. However, the proposed solution resists such attacks due to the verification of the signed message in the contract (lines 5-6 of Code Listing 3), containing the untrusted third party's identifier (idt). Since the attacker lacks the private signing key, they cannot tamper with the message. Used tokens are not validated by the blockchain since the smart contract expects the hash of the next token (hashToken) and the corresponding signed message (sigm) for each verification (lines 5-6 of Code Listing 3).

B. Resistance to Brute-Force Attacks

Asymmetric encryption ensures that deriving the private key from the public key and signed message is computationally infeasible. The combination of using the password (pw) and the private key (sk) to generate token hashes stored in the blockchain (hashToken = hash(hash(id,index,pw,sk))) makes guessing the private key computationally unfeasible, ensuring resistance to brute-force attacks (lines 4-5 and 10-11 of Code Listing 1).

C. Token Forgery

To forge a token, the adversary needs to pass the contract's verifications, satisfying the current token hash (hash(token)) and the signed message verification (verifyMessage(pk, signedMessage, message)) (lines 5-6 of Code Listing 3). However, due to the security characteristics of asymmetric encryption and the hash function, the adversary cannot deduce the required token for generating the hash, as well as the private key (sk) for creating the signed message. Assuming it is impractical to alter values maintained by the blockchain network maintainers, if the adversary tries to forge the next token by maliciously acquiring a valid token and changing the next token's hash value (hashToken), the smart contract will not validate the request. This is because the signed message also contains the hash of the next token, making it impossible to generate the signed message without the user's private key, rendering the system resistant to forgery attacks.

D. Data Confidentiality in Transactions

User transaction data is asymmetrically encrypted (line 12 of Code Listing 1) with keys generated exclusively for this purpose (line 8 of Code Listing 2). To obtain user transaction data, the adversary may attempt to acquire the private key used to decrypt this data. However, the private key is stored in a private data collection of the blockchain network (txPrivateKey) and ensures that only peers from pre-configured organizations can access this data (line 9 of Code Listing 2). Decryption of this data occurs only within the smart contract, and the data is transmitted transiently to avoid inclusion in blockchain blocks, making it challenging to leak this information.

E. Identity Recertification

An adversary may impersonate the user's identity in contracts where the user has not yet registered and take possession of transactions from that contract using the user's identifier. This work's proposal implements a centralized contract only for registration and authentication methods to centralize identifier and public key records, preventing user registrations with the same identifier if already registered in the contract (line 3 of Code Listing 2). To make the system immune to recertification attacks, the proposal implements contract-to-contract communication, as the contract must call the requested contract in the original transaction request (line 10 of Code Listing 3).

VII. Comparative Analysis

Table I provides a comparative analysis of the proposed OTTx protocol and other relevant works, highlighting key features and distinctions.

Concerning token authentication, OTTx, [9], [12], and [13] address this functionality, whereas [11] does not. This underscores OTTx's alignment with advancements in token authentication.

TABLE I
Comparison of the Proposal and Related Works (Author)

	OTTx	**[9]**	**[11]**	**[12]**	**[13]**
Token Authentication	X	X		X	X
Authenticated Transaction Submission	X	X			
Resistance to Token Forgery	X	X		X	X
No Fully Trusted Third Party Required	X		X	X	
Non-repudiation of Data Inserted by Other Identities	X				
Reduced Dependency on CA	X				X
Resistance to Reuse Attacks	X			X	X
Single Contract for Authentication	X				
Transaction Data Privacy	X				
Addressed Blockchain	HL Fabric v2.x	HL Fabric v1.x	Ethereum	HL Fabric v2.x	HL Fabric v2.x and Ethereum

Authenticated transaction submission, a crucial feature, is supported by OTTx and [9], distinguishing them from other works that do not specify this aspect. This capability enhances system security and reliability.

Ensuring resistance to token forgery is pivotal in authentication systems. OTTx, [9], [12], and [13] address this concern, while [11] does not.

A notable advantage of OTTx is the elimination of the need for a fully trusted third party, shared by [11], [12], and [13], but not by [9]. This absence reduces reliance on third parties, enhancing authentication security.

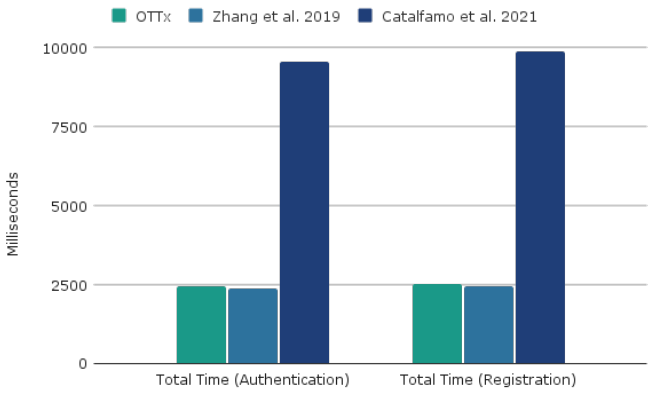

Fig. 4. Execution Time Comparison of Proposals

The ability to ensure non-repudiation of data inserted by other identities is a unique feature of OTTx, providing additional traceability and accountability benefits.

Reduced dependency on the Certificate Authority (CA) is an advantage shared by OTTx and [13], enhancing system robustness and reliability.

Resistance to reuse attacks, addressed by OTTx, [9], [12], and [13], safeguards against replay attacks and reuse of authentication information. [11] does not address this concern.

OTTx's use of a single contract for authentication, not found in other works, simplifies the process, offering efficiency and centralization.

Lastly, OTTx addresses transaction data privacy, an aspect not covered by other works.

In summary, the comparative analysis underscores OTTx's distinct advantages. Its combination of features places it in a favorable position for a secure and efficient authentication protocol.

Tests will compare the transaction authentication time in the blockchain with and without the proposed methods. The evaluation will determine if the proposal significantly impacts system efficiency. Data and graphs depicting the runtime of different approaches will be presented, alongside a comparative analysis with related works. The results will highlight the proposed solution's contributions to system performance and efficiency.

Figure 4 compares the proposals of [13] and [12] with the OTTx approach. Notably, [12]'s runtime results are similar, but it does not handle transaction data as part of the authentication process. [13]'s significantly higher total runtime is attributed to multiple intermediary services.

The evaluation used an *Intel® Core™ i5-7300HQ CPU @ 2.50GHz × 4, 16 GB RAM* running *Ubuntu Server* 22.04. Tests simulated transaction processes with *Nodejs* scripts, considering a block interval of two (2) seconds and 1000 Bytes transactions.

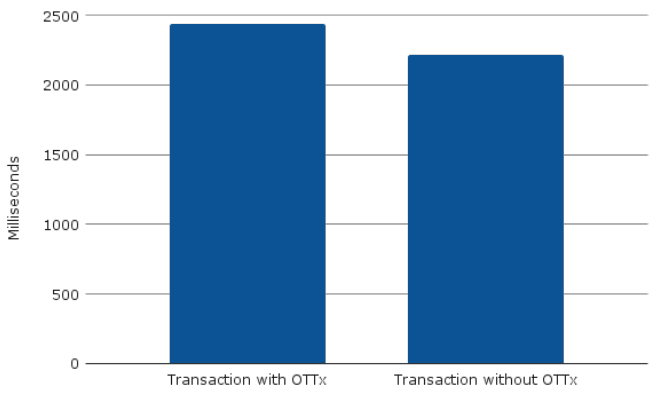

Fig. 5. Execution Time Comparison of Transactions with and without OTTx

Figure 5 shows the comparison between the proposed OTTx and a network without authenticated transactions. A slight performance loss, around 100 milliseconds, is observed, but it is not significant.

In a broader evaluation, it's important to consider limitations, such as testing in a controlled environment. Future works can address real-world deployment challenges and variations in hardware and network response times.

VIII. Conclusion and Future Work

This section summarizes the findings of the proposed OTTx protocol, a secure and efficient method for transaction submission in blockchain networks. OTTx ensures security, network access control, and efficient transaction authentication, eliminating reliance on untrusted third parties.

A. Conclusion

The OTTx protocol introduces a secure and efficient approach to sending transactions through third-party identities on blockchain networks. It addresses challenges related to security, network access control, and transaction authentication, providing a simple and secure method for submitting authenticated transactions. Through implemented components, including authentication methods, smart contract protocols, and secure token generation, the protocol enables third-party identities to submit transactions for others identities.

The protocol's implementation demonstrated feasibility and effectiveness, proving its efficiency compared to other blockchain authentication methods. Preliminary evaluations indicate comparable or superior efficiency, with a minimal additional transaction time (approximately 100 ms). This underscores the balance between performance and security, emphasizing the system's technical feasibility and tangible value in securing sensitive transaction environments.

The main contribution lies in enabling external applications to interact reliably and efficiently with the blockchain network through diverse identities, ensuring integrity, privacy, and non-repudiation.

B. Future Work

While significant results have been achieved, it is crucial to acknowledge limitations and challenges for real-world implementation. Future research should explore possibilities and address the following topics:

1) Public and Private Key Management for Transactions: Enhancing system security and flexibility involves developing mechanisms for users to manage cryptographic keys, offering greater control over transaction confidentiality.

2) Record Keeping and Synchronization: To meet AAA system security requirements, efficient methods for recording and synchronizing transaction authentication information, including token usage records, authentication indices, third-party identity identifiers, and token timestamps, need to be developed. This ensures transparency and reliability in the OTTx.

These research directions provide opportunities to enhance and expand the OTTx protocol, addressing specific challenges and exploring new possibilities. Progress in these areas will strengthen the security and reliability of authenticated transaction submissions in both permissioned and permissionless blockchain networks.

References

[1] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," March 2009. Available at: https://bitcoin.org/bitcoin.pdf.

[2] Melanie Swan, "Blockchain: Blueprint for a new economy," O'Reilly Media, Inc., 2015.

[3] Nick Szabo, "Formalizing and securing relationships on public networks," First Monday, 1997.

[4] Huaimin Wang, Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, "Blockchain challenges and opportunities: a survey," International Journal of Web and Grid Services, vol. 14, pp. 352-375, October 2018. DOI: 10.1504/IJWGS.2018.10016848.

[5] Turki Alghamdi, Nadeem Javaid, "A Comprehensive Survey on Security, Privacy and Authentication in Blockchain," 2022.

[6] Christian Cachin et al., "Architecture of the Hyperledger Blockchain Fabric," in Proceedings of the Workshop on distributed cryptocurrencies and consensus ledgers, Chicago, IL, 2016.

[7] Elli Androulaki et al., "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," Proceedings of the Thirteenth EuroSys Conference, EuroSys '18, Porto, Portugal, 2018. DOI: 10.1145/3190508.3190538.

[8] Vitalik Buterin, "A next-generation smart contract and decentralized application platform," 2014.

[9] Woo-Suk Park, Dong-Yeop Hwang, Ki-Hyung Kim, "A TOTP-Based Two Factor Authentication Scheme for Hyperledger Fabric Blockchain," in Proceedings of the 2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN), 2018. DOI: 10.1109/ICUFN.2018.8436784.

[10] Gustavus J. Simmons, "Symmetric and asymmetric encryption," ACM Computing Surveys (CSUR), vol. 11, no. 4, pp. 305-330, 1979.

[11] Peggy Joy Lu, Lo-Yao Yeh, Jiun-Long Huang, "An Privacy-Preserving Cross-Organizational Authentication/Authorization/Accounting System Using Blockchain Technology," in Proceedings of the 2018 IEEE International Conference on Communications (ICC), 2018. DOI: 10.1109/ICC.2018.8422733.

[12] Mingli Zhang, Liming Wang, Jing Yang, "A Blockchain-Based Authentication Method with One-Time Password," in Proceedings of the 2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC), 2019. DOI: 10.1109/IPCCC47392.2019.8958754.

[13] Alessio Catalfamo, Armando Ruggeri, Antonio Celesti, Maria Fazio, Massimo Villari, "A Microservices and Blockchain Based One Time Password (MBB-OTP) Protocol for Security-Enhanced Authentication," in Proceedings of the 2021 IEEE Symposium on Computers and Communications (ISCC), 2021. DOI: 10.1109/ISCC53001.2021.9631479.

[14] Chang-Seop Park, "One-time password based on hash chain without shared secret and re-registration," Computers and Security, 2018. DOI: 10.1016/j.cose.2018.02.010.
[15] Rolf Oppliger, "SSL and TLS: Theory and Practice," Artech House, 2016.
[16] Quynh H. Dang, "Secure hash standard," Quynh H. Dang, 2015.
[17] Devrim Unal, Abdulla Al-Ali, Ferhat Ozgur Catak, Mohammad Hammoudeh, "A secure and efficient Internet of Things cloud encryption scheme with forensics investigation compatibility based on identity-based encryption," Future Generation Computer Systems, 2021. DOI: https://doi.org/10.1016/j.future.2021.06.050.
[18] Leslie Lamport, "Password Authentication with Insecure Communication," Communications of the ACM, vol. 24, no. 11, pp. 770–772, November 1981. DOI: 10.1145/358790.358797.
[19] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, Frank Stajano, "The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes," in Proceedings of the 2012 IEEE Symposium on Security and Privacy, 2012. DOI: 10.1109/SP.2012.44.
[20] Joseph Bonneau, Sören Preibusch, "The Password Thicket: technical and market failures in human authentication on the web," 2013.
[21] David M'Raihi, Mihir Bellare, Frank Hoornaert, David Naccache, Ohad Ranen, "Hotp: An hmac-based one-time password algorithm," 2005.
[22] David M'Raihi, Salah Machani, Mingliang Pei, Johan Rydell, "Totp: Time-based one-time password algorithm," 2011.
[23] A.-D Liu, X.-H Du, N. Wang, S.-Z Li, "Research Progress of Blockchain Technology and Its Application in Information Security," Journal of Software, vol. 29, pp. 2092-2115, July 2018. DOI: 10.13328/j.cnki.jos.005589.
[24] Dmitry Kogan, Nathan Manohar, Dan Boneh, "T/Key: Second-Factor Authentication From Secure Hash Chains," 2017. arXiv:1708.08424.
[25] Michael, J., Cohn, A., Butcher, J. R. (2018). "Blockchain technology." The Journal, 1(7), 1-11.
[26] Shrivas, M. K., Yeboah, T. (2019). "The disruptive blockchain: types, platforms, and applications." Texila International Journal of Academic Research, 3, 17-39.
[27] Lin, I.-C., Liao, T.-C. (2017, September). "A survey of blockchain security issues and challenges." International Journal of Network Security, 19(5), 653-659. doi: 10.6633/IJNS.201709.19(5).01
[28] G. Benedict, "Challenges of DLT-Enabled Scalable Governance and the Role of Standards," in Journal of ICT Standardization, vol. 7, no. 3, pp. 195-208, 2019, doi: 10.13052/jicts2245-800X.731.