IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 24 June 2023, accepted 29 June 2023, date of publication 4 July 2023, date of current version 10 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3292143

RESEARCH ARTICLE

A Novel Secure and Privacy-Preserving Model for OpenID Connect Based on Blockchain

BELFAIK YOUSRA[1], SADQI YASSINE[1], (Senior Member, IEEE), MALEH YASSINE[2], (Senior Member, IEEE), SAFI SAID[1], TAWALBEH LO'AI[3], (Senior Member, IEEE), AND KHALED SALAH[4], (Senior Member, IEEE)

[1]Laboratory LIMATI, FPBM, Sultan Moulay Slimane University, Beni-Mellal 23000, Morocco
[2]Laboratory LISERT, ENSAK, Sultan Moulay Slimane University, Khouribga 25000, Morocco
[3]Department of Computer Engineering, Jordan University of Science and Technology, Irbid 22110, Jordan
[4]Department of Electrical Engineering and Computer Science, Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates

Corresponding author: Maleh Yassine (yassine.maleh@ieee.org)

ABSTRACT OpenID Connect (OIDC) is one of the most widely used delegated authentication protocols in web and mobile applications providing a single sign-on experience. It allows third-party applications, called Relying Parties (RP), to securely request and receive information about authenticated sessions and end-users from an identity provider. The OIDC specification defines several parameters, including the client_id, client_secret, authorization code, access token, id token, state, and redirect_uri, as keys to the protocol operation, with significant security and privacy implications. Therefore, securing these parameters is critical to prevent attackers from impersonating legitimate entities, gaining unauthorized access, having complete control over users' accounts, and/or violating their privacy. To enhance OIDC security and preserve its users' privacy, we propose a novel model for OIDC based on the Ethereum Blockchain and the non-fungible token (ERC721) standard. To prove the robustness and safety of the proposed system, we perform a detailed security analysis formally using the most widely accepted protocols security verification tools, AVISPA and Scyther, and informally by discussing various attacks. The analysis results show that the proposed system is resilient against well-known attacks. Furthermore, we evaluate the cost and performance of the proposed solution, confirming its affordability and assuring that our approach does not impact the user experience and performance of existing OIDC-based systems. Finally, we conduct a security and privacy comparative analysis with similar existing systems, proving the superiority and efficiency of our proposed Blockchain-based OIDC system.

INDEX TERMS Authentication, blockchain, OpenID connect, privacy-preserving, security.

I. INTRODUCTION

Nowadays, the number of internet users is steadily increasing worldwide. According to the 2022 mid-year estimates of the world internet usage statistics [1], there are more than 5.47 billion active internet users out of the 7.93 billion global population. This significant growth can be explained by the increasing number of internet services and the remarkable development in network technologies. All Internet services require an authentication mechanism to accept or deny users' requests to access their functionalities. Despite its security and usability risks, password-based authentication is the most widely used authentication scheme in network systems. Generally, users cannot remember their passwords for multiple accounts, which makes them choose easily-guessable passwords, write them down, use the same password for several applications, and other poor security practices [2]. Delegated authentication protocols provide a solution to this dilemma. These protocols allow users to seamlessly log in to other services called "clients" or "Relying parties (RPs)", using their identity on an identity provider (IdP), eliminating the need for each service to have its password. OpenID Connect (OIDC) is the most widely used delegated authentication protocol today [3]. Microsoft, Amazon, AT&T, Google, Meta, PayPal,

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

Verizon, Salesforce, Oracle, Symantec, Deutsche Telekom, VMWare, IBM, WordPress, Yahoo, GitHub, and Twitter are among the top digital companies that support OIDC [4]. This protocol allows billions of users to access millions of services effortlessly and securely compared to earlier solutions [5], and it is considered the new standard of the Single Sign-On system [6], [7]. According to Zhang et al. [8], over one million websites support Single Sign-on (SSO) via OIDC.

A. RESEARCH MOTIVATION

The widespread adoption of OIDC on the web, mobile, cloud, Fog Computing, IoT, and SSO systems makes its security and privacy-preserving a must. One of the critical drawbacks of this standard is that it does not verify authenticity since it only verifies and validates the parameters contained in a request but not the identity of the sender of this request. Therefore, attackers can get unauthorized access to restricted resources due to theft, improper protection, and/or incomplete validation of tokens. OIDC specification defines several sensible parameters, including the client_id, client_secret, authorization code, access token, id token, state, and redirect_uri, used in the protocol operation and its extensions workflow. However, a significant concern arises when an adversary gains access to one or more of these parameters, as it can lead to unauthorized access to restricted resources, impersonation of the RP, falsification of users' identities, exposure of personal information, and potentially acquiring full control over accounts, at the same level as a legitimate authenticated user. Therefore, ensuring confidentiality, integrity and availability of those parameters and users' unlinkability and anonymity will significantly enhance OIDC security and privacy.

OIDC's security and privacy have been widely examined. Previous works have analysed the security and privacy threats of OIDC specification and its different implementations. Various approaches have been proposed to improve OIDC security and privacy, including changing the structure or treatment of tokens, using cryptographic mechanisms, modifying the flow steps described in the analysed specification, changing the traditional implementation way or redefining policies, agreements, reputation systems, and other similar techniques [3], [4], [9], [10], [11]. However, the combination of the above approaches is still far from ideal for overcoming the security and privacy threats explored in OIDC. There is still a need for a more efficient and trustworthy OIDC model.

B. MAIN CONTRIBUTIONS

Inspired by the above challenges, this paper proposes a novel model for OIDC based on blockchain technology and smart contracts to efficiently secure the authorization and authentication process in the OIDC protocol, prevent IdPs from tracking user activities, and preserve privacy. Blockchain is a distributed and shared database ledger that organizes an increasing number of transaction records into a cryptographically linked chain of blocks. The widespread adoption of blockchain technology in the development of various industries is due to its desirable characteristics, such as decentralization, integrity, immutability, verification, fault tolerance, anonymity, auditability, and transparency. Taking advantage of these features, our proposed blockchain-based approach meets the security and privacy requirements that OIDC needs. Our model uses Ethereum, an open-source blockchain-based platform designed for distributed data storage, smart contracts, and decentralized applications (DApps), as well as the smart contract ERC-721, also referred to as the non-fungible token NFT standard. The proposed system in the present manuscript takes the approach provided in [12] as its basis but applies to all the sensitive security parameters exchanged during authentication in the OIDC protocol instead of OAuth 2.0 access tokens. Our approach aims to ensure security and authenticity and verify the ownership of an entity requesting an OIDC security parameter or a protected resource using NFTs. However, it's important to note that NFTs provide proof of possession by mapping the ownership of the token directly to the public Blockchain address of the intended owner. Furthermore, the proposed system ensures compatibility with existing OIDC-based systems since it does not change the workflow of the OIDC standard. The main contributions of this paper are as follows:

- We propose a novel model for OIDC based on blockchain technology. In this system, we have used the Ethereum Blockchain and NFTs to uniquely identify the OIDC-sensitive parameters, prove and verify the ownership of these parameters, securely store them in the blockchain, and exchange them with legitimate entities after a robust verification process.
- We suggest some minus modifications to the OIDC's client registration and user authentication flows to preserve users' privacy and prevent entities (e.i. RPs and IdPs) from tracking their activities and linking data about their interests and preferences.
- We present a prototype implementation proof-of-concept of the proposed Blockchain-based OIDC model. Furthermore, we provide a cost and performance evaluation of the proposed system to prove its affordability and that our approach does not impact the user experience and performance of existing OIDC-based systems.
- We prove the security of our proposed system formally using AVISPA and Scyther tools and informally by discussing various attacks.
- We perform a comparative analysis of our proposed Blockchain-based OIDC approach and other similar systems.

C. ORGANIZATION OF THE PAPER

The rest of this paper is organized as follows. Sections II and III present the concepts and technologies used in this work and the OIDC's security and privacy previous works, respectively. Section IV describes the security and privacy threats of OIDC. Section V examines the network model and outlines the different phases of the proposed Blockchain-based OIDC system. Section VI presents a proof of concept

and cost and performance evaluation of the proposed solution. Sections VII and VIII, respectively, conduct a formal and informal security analysis of our proposed scheme and a comparison with similar schemes. Finally, we conclude the paper in section IX.

II. PRELIMINARIES

A. OpenID CONNECT

OpenID Connect (OIDC) is a delegated authentication protocol developed by the OpenID Foundation in November 2014. It is built on the creation of a basic identity layer on top of the authorization protocol OAuth 2.0 [13]. OIDC enables Relying Parties of all types, including Web-based, mobile, and IoT, to authenticate an end-user by getting his basic profile information (name, family_name, given_name, email, picture, etc.) from an Identity Provider. OIDC enables RPs to verify end-user identity by introducing a new type of token to OAuth2.0, notably the id token, in addition to the existing access token and authorization code. The OIDC architecture includes three roles (see Fig. 1): the end-user, the relying party, and the OpenID provider (OP). The OP is itself an identity Provider (IdP) that supports OIDC.

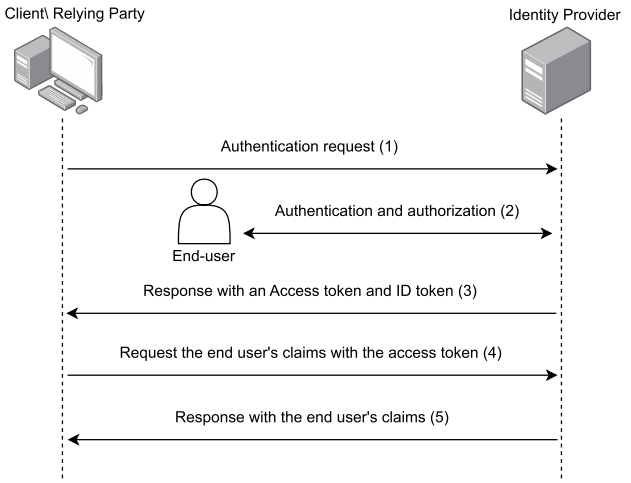


FIGURE 1. OpenID Connect typical sequence diagram.

When an end-user accesses an application (i.e. RP) that requires his authentication, the RP submits an authentication request to the IdP. Next, the IdP verifies the end-user identity and asks for his authorization. After obtaining the authorization grant, the IdP responds with an id token and access token. If the id token does not contain all the claims needed to authenticate this end-user (defined in the scope parameter of the authentication request), the RP sends a request to the IdP with the access token. If the received access token is successfully verified, the IdP respond with the end-user claims.

OIDC supports three different flows to authenticate the end-user on an RP: *Authorization code flow, Implicit flow, and Hybrid flow*. The most common OIDC authentication flow of these three is the authorization code [12], on which we will describe our proposed system.

1) OIDC CLIENT REGISTRATION

To use OIDC services for an end-user, the RP must first register with the IdP. To record a new RP at the IdP, the RP sends an HTTP POST message to the Client Registration Endpoint (a URL through which an RP can be registered at an IdP) with the Client Metadata that it chooses to specify for itself, including the redirect URI (redirect_uri) (i.e. required metadata). The redirect_uri is a URI to where send the browser, and the IdP authorization response after end-user interaction at the Idp is complete. Upon successful registration, the IdP creates a unique client identifier client_id, and optionally a client_secret to the RP, and associates it to all the registered metadata about this client [14].

2) TOKENS

OpenID Connect protocol uses three tokens that perform different activities, namely the authorization code, access token, and id token. The authorization code is a temporary code used to give RP authorization to retrieve tokens from the IdP by the RP to have an access token. The code itself is obtained from the IdP where the user gets to see what information the RP is requesting and approve or deny the request. The lifetime of this token must be short (10 mins max) to mitigate the risk of leaks and it must be used once. Furthermore, OIDC uses the access token to authorize RPs to access the user's protected resources stored at an Idp. The most common types used for access tokens are the bearer token type and the JWT type. The Bearer token is a data structure that gives any party in possession of the token rights to use it. This token can be associated with a secret key to prove the ownership of these tokens for additional security. The second type is a compact and safe way of representing the claims as JSON objects, which are encrypted and signed to ensure a secure transmission between parties [15]. The ID token is the primary extension that OIDC makes to OAuth2.0 to enable users' authentication into an RP. It is a JSON Web Token (JWT) containing information about the identity of the end-user registered on an IdP, namely claims.

3) THE AUTHORIZATION CODE FLOW

The authorization code flow (see Fig. 2) begins when an end-user requests access to an RP. Therefore, the RP prepares an authentication request and sends it to the IdP with the following parameters:

- **scope:** its basic and required value for OIDC is ''openid''. It indicates that the RP intends to use the OIDC protocol to verify the identity of a user. The scope parameter can have other values, such as ''profile'' and ''email'', which represent the group of claims needed to authenticate the end-user on the RP.
- **response_type:** it identifies the flow used. In this case, it takes ''code'' as a value.

- **client_id:** a unique identifier for the RP registered with the IdP previously.
- **redirect_uri:** is the redirection URI to which the response will be sent.
- **state:** is a recommended parameter used to maintain the state between the request and the callback. It is used to mitigate Cross-Site Request Forgery attacks.

Next, the IdP compares the value of the redirect_uri with the one registered in the RP registration phase. The process is ended if the comparison fails. The IdP then authenticates the end-user, if he is not already authenticated, and obtains his permission for the RP to access his resources stored in the IdP. Next, the IdP returns the end-user back to the RP with an authorization code and the parameter state. The state's value should be identical to the one sent in the authentication request. Next, the RP sends a token request to the IdP containing the authorization code received, redirect_uri, and the parameter "grant_type" taking as a value "authorization_code". If the code and client credentials are both successfully valid, the IdP returns an access token and an id token to the RP. The received tokens are then validated by the RP. If the id token does not contain all the claims needed to authenticate the end-user (defined in the scope parameter), the RP sends a Request to the IdP with the access token. Finally, if the access token is valid, the IdP responds with the end-user's claims to the RP.

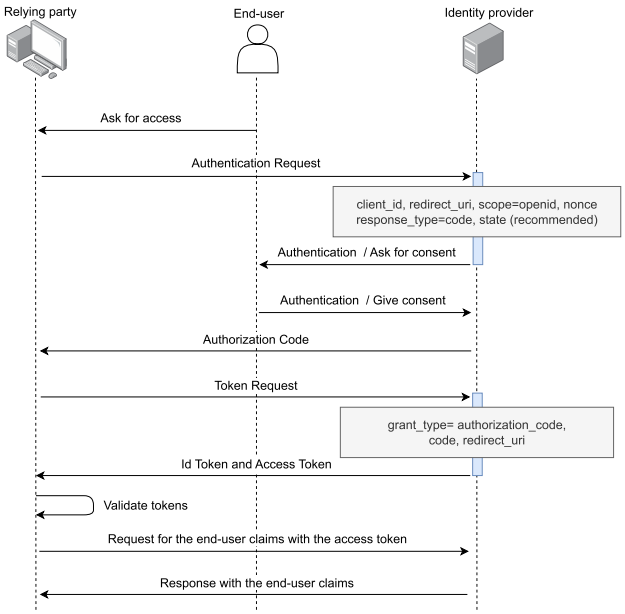


FIGURE 2. OpenID Connect authorization code flow.

B. BLOCKCHAIN TECHNOLOGY

Blockchain is a distributed, shared, and immutable database ledger that records an increasing number of transaction records into a cryptographically linked chain of blocks. This technology has come into existence in 2008 by Satoshi Nakamoto, who introduced the term blockchain for the first time as the distributed ledger underlying Bitcoin transactions. Bitcoin is the first cryptocurrency that effectively addressed the problem of double-spending on digital currency by employing a decentralized peer-to-peer network without the need for a trusted third-party [16], [17].

In recent years, Blockchain has been widely adopted in different environments including cryptocurrency, finance, Internet-of-Things (IoT), healthcare, agriculture, identity management, voting, education, etc. The wide popularity of Blockchain is due to its desirable features of decentralization, transparency, integrity, immutability, anonymity, auditability and irreversibility [18]. Blockchain eliminates the need for a centralized authority to control or decide things for an organization's transactions, as well as the need for a third-party intermediary to verify and validate the data transactions. Furthermore, a transaction can not be removed or modified once it has been recorded in the blockchain. Because of the decentralized and distributed nature of the blockchain, any modification to an involved block can impact all the subsequent blocks, since each block in the blockchain possesses the previous block's cryptographic hash value. Blockchain uses public and private keys to ensure security and prevent forgeries. The public key is the common address that everyone in the network knows, while the private key is a unique and random value known only by the user and used to sign transactions. Moreover, the blockchain protects users' privacy because transactions are carried out using the user's public address rather than his real identity.

Our proposed system is based on Ethereum, which is an open-source platform based on blockchain technology developed for distributed data storage as well as smart contracts, and deployed as a peer-to-peer network, known as being the second most popular public blockchain after Bitcoin [19], [20]. Ethereum was proposed for the first time by Vitalik Buterin in his whitepaper [21] in 2013, crowdfunded for its development in 2014, and then on 30 July 2015 Ethereum network went live. Ethereum is a next-generation smart contract and decentralized application (DApp) platform as defined in its whitepaper. DApp is a web application that functions as a central trusted authority between untrusted parties in a transaction, such as an online marketplace. The Dapp's back-end code runs on a decentralized peer-to-peer network, while the frontend code and user interfaces can be written in any language to make calls to the backend. DApps are considered decentralized because they are controlled by the logic written into the smart contract, not a user or company [22].

A smart contract [17], [23] is a self-executing program that runs on the blockchain network with no need for any intermediary. It contains a collection of functions and data that resides at a specific address on the blockchain and describes how the process will be performed and what actions will be taken once an event has occurred. Smart contracts can be considered a type of decentralized automation that facilitates, verifies, and enforces agreements in transactions, and records the results in a distributed ledger. To execute the bytecode of smart contracts in Ethereum, a decentralized virtual machine

called the Ethereum Virtual Machine (EVM) is installed in each network node [24]. This will enable nodes to agree to execute the same instructions by running the EVM code. A cost in Gas units is assigned to each instruction executed.

In the blockchain, the shared public ledger needs an efficient and secure consensus algorithm to ensure security and establish a high level of trust between participants in terms of updating or transferring data [25]. The consensus algorithm is essentially a set of rules and regulations to be followed by every participant to approve transactions and add blocks to the blockchain taking advantage of the fact that the majority of users have a common interest to keep the network honest [17]. Therefore, every transaction transmitted in the network must be approved and validated by the majority of participants based on the consensus algorithm, to add it to the block. Thus, consensus algorithms allow blockchain to preserve trust and ensure honesty between anonymous users while transmitting data in the network.

For blockchain, numerous consensus algorithms have been developed. The Proof of Work (PoW), Proof of State (PoS), and Delegated Proof of State (DPoS) consensus algorithms are the most frequently used ones in public blockchains. Proof of Work involves highly efficient and powerful nodes called miners to solve a complicated mathematical puzzle in order to add new blocks to the chain. The miner who solves the puzzle first will be rewarded with cryptocurrency. The consensus algorithm used by Bitcoin and Ethereum is PoW. The PoW consensus process has a significant drawback in that it takes a long time and a lot of energy to finish. To overcome these problems, the PoS consensus model has been proposed as a solution for others. PoS requires nodes called validators to stake their coins in the network to participate in the system and mine the next block. A validator is chosen randomly to create new blocks, share them with the network and earn rewards. Using PoS you only need to stake your money in the network, instead of doing intense computational work. If the validator tries to insert an invalid block into the chain, he will lose his stake. The Ethereum 2.0 upgrade has been activated on September 15, 2022, enabling Ethereum to validate transactions through the PoS consensus algorithm instead of the PoW consensus algorithm.

C. CRYPTOGRAPHIC TECHNIQUES IN ETHEREUM

Blockchain uses cryptography and consensus mechanisms to ensure security, establish trust between anonymous and untrusted parties, and prevent problems such as double-spending and the retrospective change of transaction data in a block after it has been successfully added to the chain. Blockchain uses popular cryptographic techniques such as asymmetric cryptography, hash functions, the Merkle tree, and digital signatures.

A blockchain account is an entity with a balance of cryptocurrency that can send transactions on the network. An account is consisting of two keys: public and private. The private key is stored in a digital wallet (i.e. an application that let you read your balance, send transactions, manage your account, and connect to applications). The private key is used to sign transactions, which are called digital signatures, and the public key is used to verify whether or not a transaction was signed by the sender [25]. This prevents forgeries and malicious entities from disseminating fraudulent transactions since we can always verify the sender's legitimacy. Because knowledge of a public key is required for digital signature verification, a user's public key can logically be chosen as the person's identity. It is a mechanism used to manage the identity of users in the Blockchain without disclosing their real identity.

In Ethereum, Elliptic Curve Cryptography (ECC) is used to generate the public key from the private key which is a random value made up of 64 hex characters. The public key is calculated from the private key using the following function Kb = Kr * G, where Kr is the private key, Kb is the resulting public key, and G is the Elliptic Curve base point of order n (i.e. prime number), and * is the elliptic curve multiplication, which is the operation of successively adding a point along an elliptic curve to itself repeatedly and it is irreversible. Ethereum uses the elliptic curve secp256k1 together with the Elliptic Curve Digital Signature Algorithm (ECDSA) to sign transactions. The secp256k1 curve is defined with the equation $y^2 = x^3 + 7$ and it has a security level of 256 bits, which is considered secure [26]. The base point G is specified as part of the secp256k1 standard and it is the same for all Ethereum users.

Hashes are fixed-size and one-way functions used in blockchain to securely record transactions in the blocks which are structured in a specific way using the Merkle tree. Each transaction passes through a hash function. The hash values of the transactions are paired and passed once again through the hash function until only one hash value is left, and this is called the Merkle root hash. Each block in the blockchain is made up of a header and a body. The Merkle root is stored with the hash of the previous block, a timestamp, Nonce and a difficulty target value in the block header; otherwise, the transaction data is stored in the block body. To add a block to the chain, it should contain the hash value of the previous block, which forms a secure interconnection link between the blocks and make the blockchain immutable. The Merkle tree can prevent tampering with data. This means that if an adversary tries to tamper with data, the hash value of all the blocks will change, rendering the chain invalid. Therefore, it is an efficient and easy way to detect whether the data has been tampered with.

Ethereum uses a hash function called keccak-256 with secp256k1 ECDSA signatures to securely submit transactions in the network [26]. When a user wants to send a message transaction, the transaction data will be first hashed using the keccak-256 hash function. Then, the hash value of the transaction will be signed with the user's private key using ECDSA. The user then transmits their transaction data and

the digital signature to the Blockchain network. The received digital signature will be decrypted and validated, and the transaction verified by the miner using the user's public key.

D. ERC-721 NON-FUNGIBLE TOKEN (NFT) STANDARD

The ERC-721 (Ethereum Request for Comments 721) is a non-fungible token standard that implements an API for tokens within smart contracts. This standard is based on the ERC-20 token standard and builds on two years of experience gained since that standard's inception [27]. Unlike ERC-20 tokens, ERC-721 tokens are unique and cannot be exchanged for other tokens (non-fungibility), and this is exactly why we chose to use this type of token in our model. ERC-721 tokens have a unique identifier, referred to in our system as tokenId, and can only be possessed by a single user [12]. ERC-721 standard defines several methods/functions and events that allow providing different functionalities. We call a smart contract that implements these methods and events an ERC-721 Contract, and once deployed, it would be responsible for keeping track of the created tokens on Ethereum. In this work, we assume that once a token is created, its tokenId and metadata are defined and that only the contract owner has the right to create new tokens. To associate a token with some metadata, the ERC recommends using the ERC-721 metadata extension, which specifies several functions to associate an ERC-721 token with its metadata. In our model, the tokenURI method is used to associate a tokenId to a URI where the metadata is stored. Table 1 describes the methods used in our proposed scheme.

III. RELATED WORKS

In this section, we present the previous studies that address the security and privacy concerns related to the OIDC protocol in both specification and implementation aspects.

In 2015, Fett et al. [28] proposed a privacy-respecting Single Sign-On system for the web called SPRESSO (for Secure Privacy-REspecting Single Sign-On). SPRESSO is a protocol designed to protect the privacy of SSO users by preventing IdPs from tracking users and collecting their activities. They proved that SPRESSO provides strong authentication and security properties based on an expressive Dolev-Yao-style web infrastructure model. Moreover, in 2017 authors provided an in-depth formal security analysis of OIDC and recommended adequate and effective security principles so that implementers have a high level of security respecting authentication, authorization, and session integrity when using OIDC [4]. In 2020, Zhang et al. [8] proposed EL PASSO, an asynchronous SSO system that preserves users' privacy against both IdPs and RPs. Motivated by the fact that SPRESSO only prevents IdPs from profiling users and does not protect them from colluding and malicious RPs and the synchronous nature of most SSO systems, EL PASSO uses zero-knowledge proofs, PS signatures and anonymous credentials to enable a privacy-preserving asynchronous authentication solution. The authors evaluated EL PASSO performance, latency and cost against OIDC and conducted a thorough security analysis of the EL PASSO system compared to other existing SSO systems.

In 2017, Weingärtner and Westphall [11] proposed a system that addresses privacy problems within the personally identifiable information (PII) stored in identity providers of OpenID Connect federations. This work suggests using encrypted PII data stored in the IdP with a key that only the user has. This approach is a continuation of their previous work proposed in 2014 [29], which suggests the use of user-inserted policies in IdPs to support and automate the dissemination process. Unfortunately, this task is complex and difficult for users to perform. For this reason, they propose in their new design a method for IdPs to develop and maintain dissemination policies, which would provide a better user experience by making this task transparent and getting direct feedback about their decisions in the dissemination interface.

Mainka et al. [9] classified OIDC attacks into single-phase and cross-phase attacks and introduced two new attacks in OIDC, namely identity provider confusion attack and malicious endpoints attack. They also evaluated the officially referenced OIDC libraries and found 75% of them vulnerable to at least one single-phase Attack. They have provided PrOfESSOS: an open-source implementation for a fully automated evaluation-as-a-Service platform for SSO services. PrOfESSOS introduces a generic approach to improve the security of OIDC implementations by systematically detecting vulnerabilities.

In 2018, Asghar et al. [30] proposed a privacy-preserving identity and access management system that prevents IdPs from profiling users' behaviour and controls users' personal information disclosure. PRIMA enables a user to get digital credentials stored locally in his device after registering with an IdP, which allows users to control access, private data shared, and revocate from a service at any time. The authors have also conducted a performance analysis of the PRIMA system and a comprehensive evaluation to prove its feasibility.

In 2018, Deeptha and Mukesh [31] proposed the EOIDC system, which enhances the current OpenID Connect authorization code flow to meet the high-level security and reliability requirement of mission-critical applications, such as bank and government services. EOIDC includes features such as end-to-end encryption, message signing, and message replay protection to ensure the authenticity and integrity of user authentication and authorization messages, as well as protection against impersonation and replay attacks and other types of security threats. The authors have also performed a security analysis and performance evaluation to prove the efficiency and safety of the EOIDC system.

Navas and Beltrán [10] performed a depth threat model of OIDC specification and its current implementations and summarized the main impacts and consequences of each identified threat on security and privacy. They also identified a number of mitigations, countermeasures and remediation options for both specification and implementation. In 2019, Li et al. [32] developed OAuthGuard, an OAuth 2.0 and OIDC

TABLE 1. The ERC-721 and ERC-721 metadata extension methods used in this model.

Methods	Purpose
ownerOf (tokenId)	Returns the address owner of the tokenId token.
transferFrom (add_from, add_to, tokenId)	Transfers tokenId token from add_from to add_to.
approve(add_to, tokenId)	Gives permission to add_to to transfer tokenId token to another account.
tokenURI (tokenId)	Returns a URI that points to a JSON file that conforms to the token's metadata.
_safeMint(add_to, tokenId, TKmedata)	Safely mints tokenId and transfers it to add_to, with its metadata.
_burn(tokenId)	Destroys tokenId. The approval is cleared when the token is burned.

vulnerability analyzer and defender to protect users' security and privacy even when RPs improperly implement OAuth 2.0 or OIDC protocols. They demonstrated its effectiveness by using it to search for five OAuth 2.0 or OIDC security and privacy vulnerabilities on the top 1000 websites accepting Google sign-in. In 2020, they performed a systematic analysis of the OAuth2.0 and OIDC systems' user access privacy properties [33]. They also proposed possible mitigations to make the protocols truly respect privacy.

In 2020, Hammann et al. [3] proposed two solutions to address a huge privacy threat in OIDC: IPs can track users and link the data about which RPs the user visits. They also gave formal security analysis using the Tamarin security protocol verification tool for OIDC standard and their extensions.

In 2020, Baum et al. [34] proposed PESTO which is a solution that aims to address the single point of failure threat in SSO systems, in which a compromised IdP can impersonate users and control all their accounts associated with this IdP. PESTO is a distributed SSO system with proactive security that uses cryptographic techniques and proactive secret sharing to distribute trust among multiple servers and securely recover servers after a compromise. In the PESTO approach, even a compromised password is not enough to obtain access to the user's accounts since each access token must be signed by all servers, and they can reject the request if they discover suspicious access patterns. The authors have also performed a thorough security analysis of the PESTO system against existing SSO systems.

In 2022, Mir et al. [35] proposed DAMFA, a Decentralized Anonymous Multi-Factor Authentication scheme for single sign-on systems. DAMFA uses multifactor authentication, Blockchain network, non-interactive zero-knowledge proof and cryptographic techniques such as pseudo-random functions, hash functions, signatures and credentials and secret-sharing schemes to address three main challenges in web SSO authentication. These challenges include the theft of authentication data stored by IdPs, user tracking and control over their sensitive data, and system and data availability. The authors have conducted a security analysis, performance evaluation and comparison of DAMFA against the existing SSO schemes.

The aforementioned works discussed the different security and privacy threats targeted at OIDC protocol [3], [4], [9], [10], [33] and proposed mitigations and new systems to overcome OIDC-based systems' security and privacy challenges [11], [8], [28], [30], [31], [34], [35]. Those schemes are based on cryptographic techniques, anonymous credentials, distributed verification techniques, and/or Blockchain network properties to provide a highly secure authentication system while maintaining users' privacy. Indeed, none of these systems provides an approach that significantly enhances OIDC security and users' privacy while maintaining the features already available in OIDC. In contrast to the above works, our proposed scheme is the first to employ Blockchain to enhance OIDC security and privacy-preserving without changing the OIDC protocol specification or workflow, making our solution compatible with the existing OIDC-based systems.

IV. OIDC THREAT MODEL

In this section, we provide the security and privacy threats of OpenID Connect protocol specification and its common implementations [3], [4], [9], [10].

A. SECURITY THREATS

1) TOKENS REPLAY

The improper protection of the authorization code, access token and id token, and/or the incorrect validation of these tokens and their audience presents a significant threat in OIDC. If an adversary obtains legitimate tokens, he can replay a token to access RP resources if it has not yet expired or if the expiration process has been handled improperly. He can also use the code to obtain tokens at the IdP (if the audience of the authentication and token requests are not checked). Or even worse, replay the id and access tokens intended for an RP to gain access to the resources of another RP that is not appropriately verifying the token's audience (i.e. verify whether this RP is truly the intended receiver of the token).

2) MANUFACTURE OF FAKE TOKENS

Because of the improper and faulty validation of id tokens, an adversary may be able to access RP resources by providing fake tokens tailored to his purposes. Fake tokens can be created by making simple modifications to the parameters of a legitimate token. To avoid the token being rejected because of an invalid signature, the adversary can use signature bypassing to modify the token's data without invalidating the signature, enforce the RP to employ incorrect keys during the signature verification or remove the signature by relying on an RP policy to accept tokens without signatures at all.

Furthermore, a malicious IdP can create fake tokens from scratch. If the RP has not properly examined the token's parameters, it will be validated, and a spoofing attack will occur.

3) IMPERSONATION OF LEGITIMATE ENTITIES

This threat occurs because of the adversary's ability to intercept the authentication flows impersonating legitimate entities RP/IdP. The adversary can manipulate the redirect_uri parameter using a malicious RP to retrieve information like authorization codes, access tokens, and id tokens from a legitimate IdP if the redirect_uri is not properly validated. Furthermore, an adversary would be able to obtain a valid authorization code if he has control of a malicious IdP and the client_id parameter of the target RP is the same in both the legal and the malicious IdP. This threat differs from threats based on replaying valid or manufactured tokens in that the adversary utilizes tokens or codes for the first time and has not been used by the legitimate user even once.

4) COMPROMISED IdP ACCOUNT

In single sign-on systems, IdPs are considered keys to the kingdom of users' accounts. If an attacker can compromise an IdP account, he can impersonate the user at any RP that supports this IdP. Thus, it causes unwanted actions like reading the victim's messages, sending new messages, changing personal information, etc. The adversary can compromise the IdP account by obtaining the victim's IdP password, which can be accomplished by installing malicious software on the target user's device, spying on the network, or employing phishing techniques. Furthermore, the attacker can get control of an IdP account by hijacking the IdP session cookie straight from the browser, sniffing it when it is delivered across an unsecured network, employing malicious software, or relying on XSS attacks.

B. PRIVACY THREATS

1) USER DATA LEAKAGE

This threat is triggered by privileged entities (IdPs and RPs) who share users' Personally Identifiable Information (PII) with third parties without their awareness. The IdPs and RPs often don't give end-users the resources they need to understand how their PII is shared or distributed. Even though IdPs and RPs are reliable agents, adversaries may still have varying degrees of access to their infrastructure. Therefore, if sensitive data is transferred through non-secure communication without the appropriate encryption, PII can be retrieved from these infrastructures.

2) USER TRACKING

IdPs can track and correlate end-user activity across many apps, services, and resources, allowing them to obtain detailed information about user behaviour, habits and interests. The subject identifier attribute in the id token uniquely identifies the end-user, allowing the IdP to link data about which RPs the user accesses. Furthermore, the IdP may monitor end users' movements over time, routines, and relationships with others via their devices, mainly utilizing data from cell towers, GPS satellites, location logs from various applications, Wi-Fi history, IP addresses, etc.

C. ADVERSARY MODEL

To better assess the security of the proposed scheme, we apply the widely used Dolev-Yao (DY) threat model [36]. In this model, an attacker has complete control over the network and can intercept, modify, analyse, and/or delete messages sent over public channels (if he knows the required keys). Furthermore, the adversary can guess a user's identity or password, but not both of them simultaneously. He can also steal a legitimate user's device and its stored values and attempt various attacks, such as impersonation, man-in-the-middle, replay attacks, etc.

V. PROPOSED SYSTEM

This section begins by examining the network model utilized in our proposed system. Next, it presents the notation and assumptions that must be taken into account within our system. Finally, a comprehensive explanation of the system model for our proposed scheme is provided following the most commonly used OIDC flow known as the authorization code flow. Additionally, we outline the suggested modifications to be made in the OIDC core specification and its client registration extension.

A. NETWORK MODEL

The Network model of our proposed Blockchain-based OIDC system is shown in Fig. 3. As with the standard OIDC protocol, our system revolves around three key entities: the user, the RP, and the IdP. The RP and IdP also act as Ethereum nodes in the Blockchain network, where the IdP assumes ownership of the smart contract responsible for generating non-fungible tokens. The process typically begins with the user attempting to access the resources of an RP that requires authentication. The IdP serves as the trusted intermediary responsible for authenticating the user and providing identity information to the RP. Therefore to authenticate the user, the RP initiates a request to the IdP to get the user's consent to access their identity information on the IdP by redirecting the user browser to the IdP. It is important to note that, in the OIDC specification, the authentication request should be transmitted securely via TLS. However, due to the incorrect implementation of OIDC and mistakes made by RP developers [32], this request is transmitted over an insecure channel. Thus, an attacker can intercept such a request and uses a malicious RP to generate a manipulated similar request with legitimate variables to gain a valid access token from the IdP. To mitigate this security concern, we use Blockchain to verify the authenticity of the RP sending this request using the smart contract. The user then is presented with a login page provided by the IdP where they can enter their credentials. Upon successful authentication, the IdP generates an

authorization grant (i.e. a code or an access token depending on the OIDC flow used) and then redirects the user browser back to the RP through a series of HTTP-based interactions, typically utilizing redirects. This kind of communication is susceptible to attacks such as replay, man-in-the-middle, impersonation, fake token manufacturing, sniffing, phishing, etc., [10].

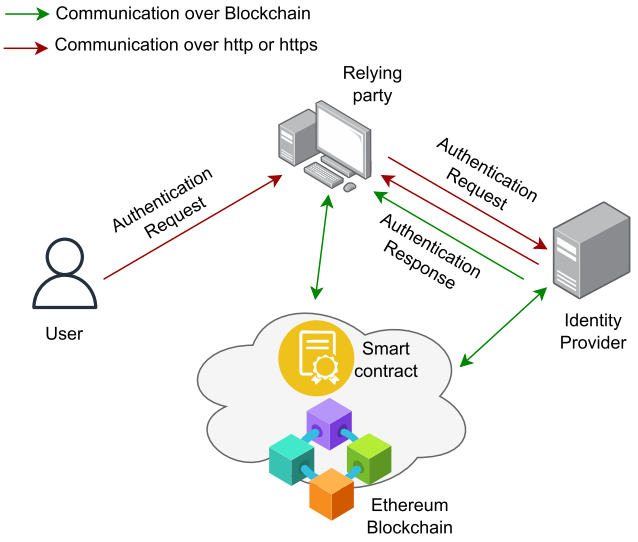


FIGURE 3. Network model of the Blockchain-based OIDC system.

Therefore, to secure these tokens and ensure the authenticity of the entity receiving these tokens, we use the smart contract to generate NFTs equivalent to each token, which are then transmitted securely over the blockchain to the attended RP. The RP can then use this code or token to make token requests to the IdP to obtain an ID token and access token. The ID token contains the user's identity information, while the access token allows the RP to access protected resources on behalf of the user. The IdP typically responds to the RP's token request in the OIDC protocol by using HTTP-based communication, which is not secure. As in the previous step, our system ensures the validity and authenticity of the entity sending the token request. Then, the IdP creates the access token and ID token and their equivalent NFTs. Next, the IdP transmits only the access token over HTTP/HTTPS channel and transfers both NFTs to the RP over the blockchain. This will ensure that the Id token is only used by the legitimate RP. Depending on the scope, the RP may still need additional identity information to authenticate the user. In this case, the RP uses the access token to send a user-claims request to the IdP. Based on the same process, our system verifies the authenticity of the RP sending this request before responding with the user's information.

B. NOTATION AND ASSUMPTIONS

Within our system, we assume that both identity providers (IdPs) and relying parties (RPs) are trustworthy entities with no malicious intent. This implies that the IdP only sends valid and correct tokens, whereas the RP only registers redirect_uris linked to its domain and receives valid signed tokens from the approved IdP. Furthermore, we assume that all tokens are of the JSON Web Token (JWT) format and have not yet expired. The end-user's identity is classified as Personally Identifiable Information (PII), as it encompasses a set of attributes (claims) used to identify the user, such as name, surname, email address, phone number, photo, etc.

TABLE 2. Description of different notations used in our blockchain-based OIDC system.

Notation	Description
OIDC	OpenID Connect.
IdP	Identity provider.
RP	Relying Party.
U	End-user.
JWT	JSON Web Token.
AC	Authorization code.
AccTK	Access token.
IdTK	Id token that contains the user claims.
client_id	Unique identifier of the RP.
client_secret	Secret value used by clients to authenticate to the IdP.
client_add	Required parameter in our system takes as value the RP's Ethereum address.
redirect_uri	Redirection URI to which the response will be sent.
RP_metadata	RP's metadata.
ErcTK	ERC-721 non-fungible token.
tokenId	Unique identifier of ERC-721 token.
TKmetadata	ERC-721 token metadata.
Contract_IdP	ERC-721 smart contract owned by the IdP.
URIclaims	End-user claims' URI mapped to the tokenId of the ERC-721 token.
EthRP	Address Ethereum of the RP.
RegRequ	RP registration request.
RegResp	Registration response.
AuthRequ	Authentication request.
TkRequ	Token request.
ClRequ	End-user's claims request.
scope	Required parameter in OIDC in which we specify the group of claims needed to authenticate an end-user.
response_type	Required parameter in which we indicate the flow used.
state	A recommended parameter used to mitigate Cross-Site Request Forgery (CSRF) attacks.
iss	Issuer of the token.
sub	Subject of the token.
aud	Audience that this token is intended for.
exp	Token expiration time.
jti	Unique identifier of the token.

In our system, each IdP owns an ERC-721 smart contract, referred to as Contract_IdP, and is the only entity able to create new ERC-721 tokens. Each ERC-721 token (ErcTK) is identified with a unique identifier, referred to tokenId, and some metadata. The metadata associated with an ErcTK of our system is the base64url code of a JWT. To associate

a token with its metadata, we use the tokenURI(tokenId) method of the ERC-721 metadata extension, which returns the URI token metadata. Furthermore, a URI is used to uniquely identify the end-user's claims, referred to as the URIclaim. When an RP requests a specific URIclaim from the IdP, it is expected that the RP has already confirmed the owner of such claims. We also assume that each RP has a pair of keys, a public key and a private key, used to conduct transactions on the Ethereum blockchain. The address Ethereum of an RP is denoted as EthRP. Table 2 summarizes the notations used in our system.

C. BLOCKCHAIN-BASED OIDC SYSTEM DESIGN

The proposed solution aims to enhance security and user privacy-preserving in OIDC systems incorporating the Ethereum blockchain and NFTs. This approach ensures the integrity, availability, and confidentiality of OIDC security parameters and restricts their access to authorized entities through a robust verification process. Our system is divided into three phases; the RP registration on the IdP, the end-user authentication on the IdP, and the end-user U authentication on the RP. The proposed system model is shown in Fig. 4, illustrating the various components and their interactions within the system.

1) RP REGISTRATION ON THE IdP

To register a new client (i.e. RP) with the IdP, the client initiates a registration request to the IdP. This request includes RP metadata (RP_metadata) that the client chooses to provide for itself during the registration process. Upon receiving a valid registration request, the IdP prepares a response containing a newly-created client identifier (client_id) and, optionally, a (client_secret), along with all the registered data about this client. Our solution proposes adding a required parameter called ''client_add'' to the IdP registration response. This parameter must take as value the client's Ethereum address (EthRP). Its value is required in the following steps to verify the ownership of the client_id. Next, the IdP generates an ERC-721 token where its id (tokenId) matches the value of the client_id parameter, and its metadata (TKmetadata) sets equal to the base64url encoding of the registration response, which is of a JSON data type. Finally, the IdP sends his response to the client over HTTP/HTTPS channel and transmits the created ErcTK token to the client's Ethereum address EthRP over the blockchain network using the transferFrom() method of the ERC-721 smart contract.

Algorithm 1 RP Registration on the Idp

Input: RegRequ.
Output: RegResp, ErcTK.

1. RP chooses its RP_metadata
2. RP generates a RegRequ contains RP_metadata
3. RP sends the RegRequ to IdP
4. IdP creates {client_id, client_secret}
5. IdP generates a RegResp containing: {client_id, client_secret, client_add, RP_metadata}
6. IdP creates ErcTK where: $tokenId \leftarrow client\_id$ and $TKmetadata \leftarrow based64url\{RegResp\}$.
7. IdP sends RegResp to the RP and ErcTK to the EthRP

2) END-USER AUTHENTICATION ON THE IdP

When an end-user (U) visits a client application that requires his authentication, the client sends an authentication request to the IdP. In this step, we propose adding the parameter client_add as a required parameter in the authentication request. This addition will enable the IdP to verify the authenticity of the client who sent this request by executing the ownerOf() method with the client_id value obtained in the request as input. If the returned address matches the EthRP provided in the authentication request's client_add parameter and all the other parameters are validated, the IdP authenticates the end-user and obtains his consent for this client. Next, the IdP generates an authorization code and creates an ERC-721 token equivalent to this code. Then, the IdP redirects the end-user with this code to the client over HTTP/HTTPS channel and transmits the created ErcTK' token to the EthRP over the blockchain network using the transferFrom() method. The authorization code is a unique parameter with a short lifetime (10 minutes maximum), which neither OAuth2.0 nor OIDC specified its type. In our system, we use JWT as the type of code.

Algorithm 2 End-User Authentication on the IdP

Input: AuthRequ.
Output: AC, ErcTK'.

1. RP generates an AuthRequ containing {client_id, client_add, redirect_uri, scope, response_type, state}
2. RP sends AuthRequ to the IdP to authenticate U
3. IdP verifies the ownership of the request using the ownerOf() method ownerOf(client_id) =? client_add.
4. IdP verifies all other parameters according to OIDC specification
5. IdP validates AuthRequ
6. IdP authenticates U if he is not already authenticated.
7. IdP obtains the U's consent for the RP
8. IdP generates AC where: { $iss \leftarrow Contract\_IdP$, $sub \leftarrow EthRP$, $aud \leftarrow Contract\_IdP$, jti, $exp \leq 10min$}
9. IdP creates ErcTK' where: {$tokenId \leftarrow jti$ and $TKmetadata \leftarrow based64url\{AC\}$.
10. IdP sends AC to the RP and ErcTK' to the EthRP

3) END-USER AUTHENTICATION ON THE RP

Upon receiving an authorization code (AC), the client can use this code to send a token request (TkRequ) to the IdP. The IdP first(1) verifies who generates this code by examining if the ''iss'' claim value of the AC matches the Contract_IdP, (2) verifies if the ''aud'' claim value matches the Contract_IdP, (3) verifies if the ownership of the code

is the legitimate client or not using the ownerOf() method which takes as input the ''jti'' claim of the AC, and checks whether the returned address matches the EthRP included in ''sub'' claim of the AC, and finally (4) verifies if the returned metadata of the tokenURI() method corresponds to the base64url code of AC JWT value. Moreover, the IdP verifies the redirect_uri parameter and validates the TkRequ request. Upon successful validation, the IdP creates an Id token (IdTK), an access token (AccTK), and two ERC-721 tokens (ErcTk'') and (ErcTkÂ°), equivalent to the IdTK and AccTK, respectively. Next, the IdP transfers only the AccTK to the client over HTTP/HTTPS channel and the created ErcTk'' and ErcTkÂ° tokens to the EthRP over the blockchain network. The RP can get the Id token (e.i. of JWT form) by encoding the string output of the tokenURI() method of the received ErcTk'' to the JSON form. If the client needs additional claims to authenticate the end-user, it sends a request ClRequ to the IdP with the AccTK. Next, the IdP validates this token by examining (1) if the ''iss'' claim value matches the Contract_IdP, (2) if the URIclaim included in the ''aud'' claim corresponds to the URI of the end-user claims, (3) if the ownership of the token is the legitimate client or not using the ownerOf() method with the ''jti'' claim of the AccTK as input, and verifies whether the returned address corresponds to the EthRP given in the AccTK's sub claim, and finally (4) verifies if the returned metadata of the tokenURI() method corresponds to the base64url code of AccTk value. If the token is successfully validated, the IdP responds with end-user claims.

Algorithm 3 End-User Authentication on the RP

Input: TkReq, ClRequ.
Output: AccTK, IdTk, ErcTK", ErcTKÂ°.

1. RP generates TkRequ containing {AC, redirect_uri'}
2. RP sends TkReq to the IdP
3. IdP verifies iss =? Contract_IdP
4. IdP verifies aud =? Contract_IdP
5. IdP verifies ownerOf(jti) =? EthRP included in the sub
6. IdP verifies tokenURI(jti) =? based64url{AC}
7. IdP verifies redirect_uri' =? redirect_uri
8. IdP validates TkRequ
9. IdP creates AccTK and IdTk where AccTk: {$iss' \leftarrow Contract\_IdP$, $sub' \leftarrow EthRP$, $aud' \leftarrow URIchaim$, jti', exp' } and IdTK: {$iss'' \leftarrow Contract\_IdP$, $sub'' \leftarrow EthRP$, $aud'' \leftarrow client\_id$, jti'', exp''}
10. IdP creates ErcTK" and ErcTKÂ° where ErcTK": {$tokenId \leftarrow jti''$} and $TKmetadata \leftarrow based64url\{IdTK\}$} and ErcTKÂ°: {$tokenId \leftarrow jti'$ and $TKmetadata \leftarrow based64url\{AccTK\}$
11. IdP sends AccTK to the RP and {ErcTK", ErcTKÂ°} to the EthRP
12. RP generates a ClRequ containing the AccTK
13. RP sends ClRequ to the IdP
14. IdP verifies iss' =? Contract_IdP
15. IdP verifies aud' =? URIclaim
16. IdP verifies ownerOf(jti') =? EthRP included in sub'
17. IdP verifies tokenURI(jti') =? based64url{AccTK}
18. IdP validates AccTK
19. IdP sends U's claims to the RP

VI. PROOF OF CONCEPT AND EVALUATION

A. PROOF OF CONCEPT

In this section, we provide a proof of concept for our proposed blockchain-based OIDC solution. We initially developed an ERC-721 smart contract in Solidity, the primary programming language in Ethereum, that contains all the methods used in our system (specified in Table 1). Then, we compile it using one of the most commonly used development environments for smart contracts on Ethereum, Remix IDE. Next, we employed Ganache, a local personal blockchain serving as our web3 provider and the environment where to deploy our smart contract and run our tests. Ganache provides pre-funded accounts, each equipped with 100 Ether. From these accounts, we selected two: one representing the IdP and serving as the ERC-721 contract owner, while the other representing the RP. Then, we adjusted the gas limit and gas price within the Ganache network to match their current values of 30M and 54 gwei, respectively. Fig. 5 (a) depicts the account balance associated with the IdP, which takes $0 \times 906 \ldots$F1919 as its address, and Fig. 5 (b) illustrates the transaction log where the IdP calls the constructor to deploy the smart contract.

Moreover, we have used an educational sandbox for web3, including drag-and-drop programming and open-source building blocks, called ETH.build, to set up a prototype of the proposed system and run our tests. In the ETH.build sandbox, we have used Ganache as the Ethereum blockchain where we run out our tests. We have connected the ABI of our compiled ERC-721 smart contract, the Ganache blockchain, and the IdP address as the inputs of the contract component. Then, we will see all the functions used in the smart contract pop out as outputs. Furthermore, we have used a JSON web token (JWT) with its different attributes to test the process of transferring the security parameters of OIDC, which are of type JWT, on the Ethereum network. We have used the _safeMint() method to safely mint a new token. The tokenId input takes as a value the jti attribute of the JWT, the _data input takes as a value the byte32 code of the JWT, and the to address is the contract owner (i.e. IdP address). Next, we transfer this token from the IdP address to the RP address on the Ethereum network, using the transferFrom() method. We can use the approve(), ownerOf(), and tokenURI() methods to, respectively, approve the token to another address, verify the token's owner, and obtain the metadata URI of the token.

B. COST EVALUATION

Our proposed system requires the invocation of smart contract functions, which generate a computational effort for which a fee is required. Gas is the fee necessary to complete

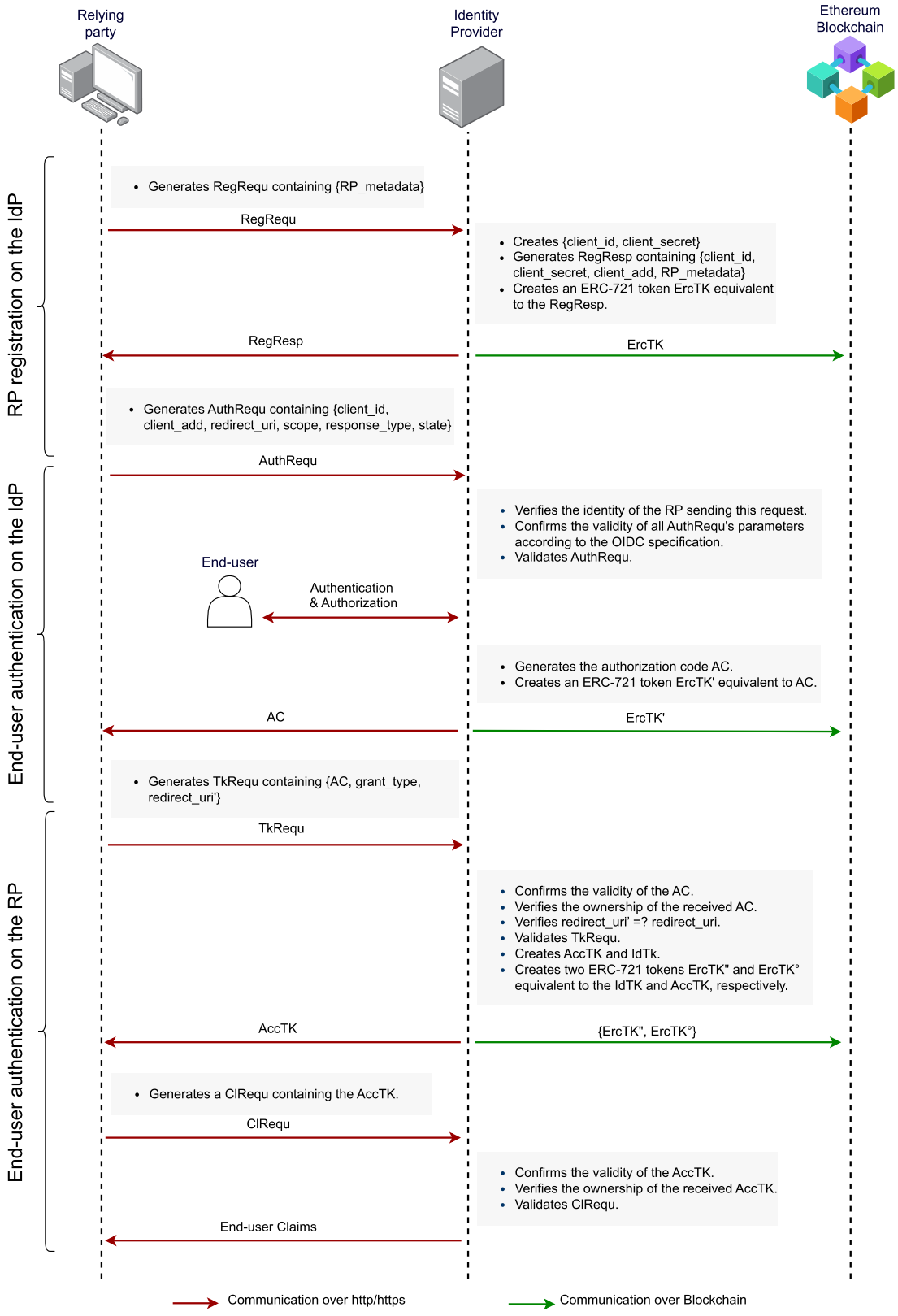
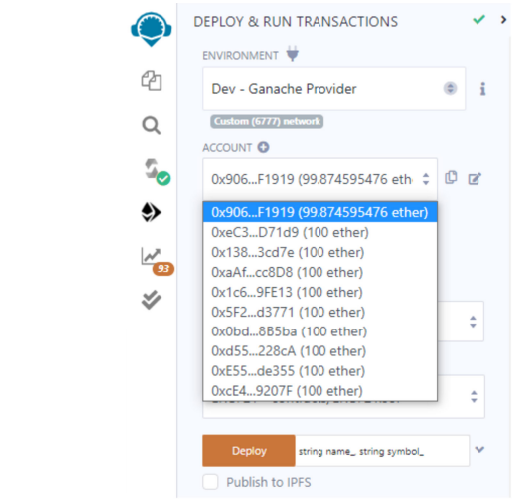
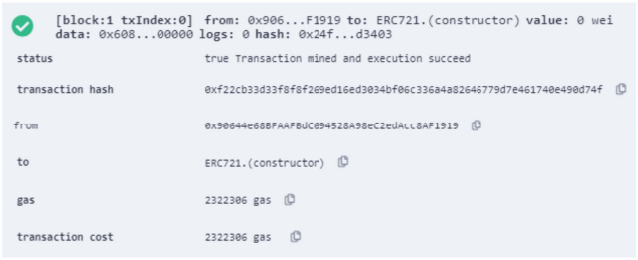


FIGURE 4. The proposed blockchain-based OpenID Connect system model.



(a) IdP account balance after deploying the ERC-721 smart contract



(b) Transaction log

FIGURE 5. Remix IDE screen of our deployed ERC-721 smart contract.

TABLE 3. Cost of the operations used on our proposed blockchain-based OIDC system.

Operation	Transaction gas	Gas cost in Ether
Contract deployment	2322306	0.125288409
Create a token	73282	$3.95535639 \times 10^{-}3$
Transfer a token	50987	$2.75074865 \times 10^{-}3$
Burn a token	22843	$1.23237985 \times 10^{-}3$
Approve	46154	$2.4900083 \times 10^{-}3$

an Ethereum transaction successfully. Since each transaction requires computational resources to execute, transactions consume an amount of Gas. The ether (ETH), the native currency of Ethereum, is used to pay the gas fees. Gas costs are expressed in (gwei), which is a denomination of (ETH), and one (gwei) equals 10-9 (ETH). The Ether consumption can be calculated as Ether_cost = (Gas_cost × Gas_price per unit) × $10^{-}9$. According to [37], the Ethereum average gas price on May 19, 2023, is 53.95 (gwei). Table 3 shows the gas cost of each operation used in our system and the corresponding Ether cost, starting from the contract deployment in the Ganache Ethereum network to the destruction of a token. Furthermore, the functions tokenURI and ownerOf are declared as view functions, which only read the state of the blockchain without doing any modifications. Therefore, they afford no cost or computational effort.

C. PERFORMANCE EVALUATION

In this subsection, we evaluate the performance of our proposed system in terms of time and Bytes needed to authenticate a user. However, the transaction speed, denoted as the number of transactions per second within a blockchain network, depends on various factors such as block time, gas limit, and transaction gas, which refers respectively to the number of seconds to wait between mining new blocks and transactions, the maximum amount of gas available to each transaction and block and the gas fees required to successfully conduct a transaction or execute a contract on the blockchain. The transaction speed is calculated using the following formula [22]: *Transaction speed = Block gas limit ÷ (Transaction gas × Block time)*. To ensure reliable outcomes, we set the block gas limit and the block time at their current values (May 19, 2022). According to [38] and [39], the Ethereum block time is 12.15 seconds, and the block gas limit is 29,999,954 gas. Table 4 presents the number of transactions processed per second, the time required to process a single transaction, and the transaction TX data size for the various functions employed in our system.

TABLE 4. Transaction speed and process time.

Operation	Transaction speed	Process time (ms)	TX data size (Bytes)
Contract deployment	1	943.4	24076
Create a token	33	29.68	778
Transfer a token	48	20.66	202
Burn a token	108	9.25	74
Approve	53	18.69	138

When authenticating a user using the OIDC authorization code flow, typically, the RP sends two requests. The first is the authentication request to get the authorization code, and the second is the token request to get the Id and access tokens. To calculate the approximate time between the RP request and the IdP Response in these two phases, we have used the Google API console to establish our test in the same infrastructure as Google. First, we created a web application (i.e. RP) in the Google API console that can interact with Google's APIs [40]. Next, we configured our application to make requests to Google Gmail API using Postman. This means that we set up the callback URI (i.e. redirec_uri) of our application to the Postman domain (https://oauth.pstmn.io/v1/browser-callback) from which we can send API requests to Gmail API. Upon a valid HTTPS GET request to the Google authorization endpoint (https://accounts.google.com/o/oauth2/v2/auth), Google responds with a one-time authorization code in 1158 ms, taking into consideration that the average round-trip time to and from Google Cloud is 41 ms. Next, we used the received code to send an HTTPS POST request to the Google token endpoint (https://oauth2.googleapis.com/token). Upon a successful response to this request, Google sends back

an Id token and access token in 329 ms. In our proposed blockchain-based OIDC system, Table 4 shows that time needed to create and send an ERC-721 token over the blockchain network is approximately 50 ms. This proves that in all OIDC authentication phases, our system will not affect users' experience and the performance of the existing OIDC-based systems.

VII. SECURITY ANALYSIS OF THE PROPOSED SYSTEM

In this section, we analyse the security of our proposed system formally using AVISPA and scyther tools and informally by discussing its different security aspects.

A. FORMAL SECURITY ANALYSIS

To ascertain the security of our scheme, we present in this subsection the simulation of the proposed model using the most widely used and accepted tools for formal security verification and analysis of protocols in recent years, AVISPA [41] and Scyther [42], which are both based on the adversary model Dolev-Yao. The Automated Validation of Internet Security Protocols and Applications (AVISPA) tool is a modern push-button tool used to specify protocols and their security properties by using a modular and expressive, role-based, formal language called the High-Level Protocol Specification Language (HLPSL) [43]. The robustness and flexibility of AVISPA are due to its modular approach where the back-end tools are independent of the specification language, giving third parties the possibility to develop their back-ends as long as they follow the intermediate format [44]. The HLPSL2IF tool translates HLPSL specifications into an intermediate format (IF), a lower-level language at a lower abstraction level. AVISPA uses the IF specification format as input to other verification tools (back-ends) that analyse the same protocol in different ways [45]. Presently, AVISPA supports four back-ends: On-the-fly Model-Checker (OFMC) [46], Constraint-Logic-based Attack Searcher (CL-AtSe) [47], SAT-based Model-Checker (SATMC) [48], and Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (TA4SP) [49].

To verify the security of a protocol using AVISPA needs to be described in the HLPSL language. This means defining the roles and their composition for representing the protocol. Then, specify the execution environment and the security properties that should be satisfied by the protocol. AVISPA verifies protocols' safety based on three types of security goals: confidentiality (secrecy_of), strong authentication (authentication_on), and weak authentication (weak_authentication_on). In the HLPSL specification of our proposed system, we defined two basic roles: role_IdP and role_RP. Then, we specified two goals for authentication and two goals for tokens' secrecy in the goal section. The HLPSL description of our system is presented in Appendix A. In general, if the analysis result of the two models OFMC and CL-AtSe is **SAFE**, the protocol can be considered secure against replay and MITM attacks. The simulation summaries of OFMC and CL-AtSe are presented in Fig. 6 and 7 and show that our proposed protocol is **SAFE**. The translation time in CL-AtSe is 0.01 seconds, and the search time in OFMC is 0.02 seconds for visiting 16 nodes with a depth of 4 piles.

```
% OFMC
% Version of 2006/02/13
SUMMARY
 SAFE
DETAILS
 BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
 /home/span/span/testsuite/results/BlockchainOIDC.if
GOAL
 as_specified
BACKEND
 OFMC
COMMENTS
STATISTICS
 parseTime: 0.00s
 searchTime: 0.02s
 visitedNodes: 16 nodes
 depth: 4 plies
```

FIGURE 6. AVISPA verification result with OFMC backend.

```
SUMMARY
 SAFE

DETAILS
 BOUNDED_NUMBER_OF_SESSIONS
 TYPED_MODEL

PROTOCOL
 /home/span/span/testsuite/results/BlockchainOIDC.if

GOAL
 As Specified

BACKEND
 CL-AtSe

STATISTICS

 Analysed   : 4 states
 Reachable  : 0 states
 Translation: 0.01 seconds
 Computation: 0.00 seconds
```

FIGURE 7. AVISPA verification result using CL-AtSe backend.

For a more in-depth analysis, we have re-analyse the security of our proposed system using the scyther tool since it provides additional features not offered by AVISPA [50]. Scyther guarantees protocols' security verification for a bounded/ unbounded number of sessions, provides multi-protocol analysis by verifying the concatenation of their description files and generates attack graphs when attacks are found [42]. Scyther takes as input a formal

description of the security protocol to be analyzed, which is specified in the form of a security protocol description language (SPDL) file. The SPDL language describes the protocol's roles which are defined by a sequence of declarations, sending and receiving events, claim events and security properties that the protocol is expected to satisfy. In the SPDL file of our blockchain based-OIDC system, we have defined two roles: the IdP and the RP, in which we have defined claims to verify confidentiality (Secrecy) and other authentication properties such as aliveness (Alive), non-injective synchronisation (Nisynch), non-injective agreement (Niagree) and weak agreement (Weakagree). Appendix B presents the SPDL description of our proposed system. As illustrated in Fig. 8, the Scyther tool finds no attacks for our proposed system.

Scyther results : verify

Claim				Status		Comments
blockchainOIDC	RP	blockchainOIDC,RP1	Secret codeTk	Ok		No attacks within bounds.
		blockchainOIDC,RP2	Secret ERCtk	Ok		No attacks within bounds.
		blockchainOIDC,RP3	Alive	Ok	Verified	No attacks.
		blockchainOIDC,RP4	Weakagree	Ok	Verified	No attacks.
		blockchainOIDC,RP5	Niagree	Ok		No attacks within bounds.
		blockchainOIDC,RP6	Nisynch	Ok		No attacks within bounds.
	IdP	blockchainOIDC,IdP1	Secret codeTk	Ok	Verified	No attacks.
		blockchainOIDC,IdP2	Secret ERCtk	Ok	Verified	No attacks.
		blockchainOIDC,IdP3	Alive	Ok	Verified	No attacks.
		blockchainOIDC,IdP4	Weakagree	Ok	Verified	No attacks.
		blockchainOIDC,IdP5	Niagree	Ok	Verified	No attacks.
		blockchainOIDC,IdP6	Nisynch	Ok	Verified	No attacks.

Done.

FIGURE 8. Results of Scyther's evaluation of the proposed system.

B. INFORMAL SECURITY ANALYSIS

1) RESISTANCE TO REPLAY ATTACKS

A replay attack allows an attacker to intercept and record a legitimate data transmission, such as tokens, and then replay that data to gain unauthorized access to restricted resources. Let's assume that an attacker obtains one of the OIDC security parameters (i.e. access token, authorization code, redirect_uri, and client_id). Our proposed blockchain-based OIDC system ensures authenticity by verifying not only the parameters contained in a request but also the identity of the sender of this request using the smart contract's OwnerOf() function. For example, if an attacker gets a legitimate access token and attempts to re-transmit it to obtain the user authentication claim or an id_token, the system would not approve this request because the ownerOf() output address of this token would not match the address of the attacker.

2) RESISTANCE TO IMPERSONATION ATTACKS

An attacker can impersonate legitimate entities and access protected resources if he obtains one of the OIDC-sensitive parameters transmitted over the authentication flows. Our system prevents this attack due to our robust verification process. If the attacker attempts to impersonate the user and sends a request to access restricted resources using a stolen freshly generated token, his request will be rejected because the attacker's Ethereum address will not match the address contained in the ''sub'' attribute of the JWT nor the output address of the ownerOf() function.

3) RESISTANCE TO FAKE TOKEN MANUFACTURER ATTACKS

A fake token manufacturer attack allows an attacker to create counterfeit tokens that look like legitimate tokens and distribute them to gain unauthorized access to RP resources and the user's sensitive information. Our system prevents this attack since we create for each token generated by the IdP an associated ERC-721 token transmitted over the blockchain network, which enables the RP to verify if this token is generated from the legitimate IdP address by a simple check on the public ledger and if the returned string of the tokenURI() function is equal to the base64url representation of the received token.

4) INTEGRITY AND AVAILABILITY

Our proposed system is based on the Ethereum blockchain, which maintains integrity and availability through cryptographic hash functions, digital signatures, consensus mechanisms, and a distributed network of nodes. In the Ethereum network, each transaction is hashed using the keccak-256 hash function, which is considered highly secure, and then signed with the private key using a digital signature algorithm called Elliptic Curve Digital Signature Algorithm (ECDSA). Finally, the transaction is broadcast to the entire network for miners to verify and approve. Before adding transactions to the network, miner nodes have to agree on the validity and the correctness of those transactions using a consensus protocol, such as Proof of Work (PoW), Proof of Stake (PoS), Proof of Authority (PoA), etc. This makes it challenging for attackers to gain network control and tamper with the transactions. Moreover, our system ensures availability where authorized users can access the network and interact with it at any time without interruption. Due to the blockchain's decentralized and distributed architecture, nodes have a simultaneous copy of the ledger and work together to validate transactions, create new blocks, and maintain the integrity of the network. If some nodes go offline, the transaction processing and the network can continue ensuring system availability.

5) REVOCATION AND DELEGATION

Our proposed system allows access revocation from an RP by using the ERC-721 contract transfertFrom() function and returning the token to the IdP. If the RP has already used the JWT and this token has been associated with a session

TABLE 5. Comparison of the proposed system with other existing systems.

	OIDC	EOIDC [31]	PRIMA [30]	EL PASSO [8]	PESTO [34]	DAMFA [35]	Our work
Integrity	✓	✓	✓	✓	✓	✓	✓
System availability	✓	✓	✓	✓	≈	✓	✓
Users Anonymity	×	×	×	✓	×	✓	✓
Unlinkability	×	×	≈	✓	×	✓	✓
Real-time assessment	×	×	×	×	×	✓	✓
Resistance to fake token manufacturer attacks	×	×	✓	✓	✓	✓	✓
Resistance to replay token attacks	×	✓	×	×	✓	✓	✓
Resistance to impersonation attacks	×	✓	×	×	✓	✓	✓
Revocation and delegation	×	×	✓	×	×	×	✓

identifier, the verification process will fail if the RP tries to use it after the revocation because the output of the ownerOf() function will not match the RP Ethereum address EthRP given in the JWT "sub" attribute. Delegation property is also guaranteed in our system, enabling the user to log in to his RP account using another device that does not have access to the EthRP address and delegate tokens to another Ethereum address using the approve() function.

6) ANONYMITY AND UNLINKABILITY

In the traditional OIDC protocol, IdP and RPs agents can track users' activities and collect data about their behaviour and interests. The IdP can associate data about which RPs the user accesses using the "sub" attribute included in the id token. Our proposed system effectively addresses this concern by substituting the user's real identity in the "sub" attribute of a generated id token with the public address EthRP. Moreover, an Ethereum account consists of a pair of public and private cryptographic keys. The public key is generated from the private key using the ECDS algorithm, and the public address of an Ethereum account is the last 20 bytes of the Keccak-256 hash of the public key. Therefore, users can generate multiple addresses for different RPs, allowing a single user to possess distinct EthRP addresses for each RP he attempts to sign-up with. This property also prevents colluding RPs from linking users' accounts with a specific individual since the Id token does not contain any identifying information about the user's real identity. Thus, our system ensures the unlinkability of the IdP and colluding RPs, thereby safeguarding user privacy.

VIII. COMPARATIVE ANALYSIS

In this section, we compare our proposed Blockchain-based OIDC system with similar authentication schemes in terms of security and privacy-preserving in order to understand the strengths and weaknesses of each one. Table 5 presents the results of our evaluation of the seven schemes, including our work, based on their effectiveness in ensuring integrity, system availability, users anonymity, RP and IdP unlikability, Real-time assessment, resistance to fake manufacturer attacks, replay token attacks and impersonation attacks, access revocation and delegation. Symbolic notations are employed in the table, where × indicates that the scheme does not provide the corresponding property, ✓ denotes the opposite, and ≈ indicates that the property is partly ensured by this scheme or under certain conditions.

The comparison shows that all schemes ensure data integrity using a secure communication channel over TLS protocol or relying on cryptographic techniques such as one-way hash functions. In addition, all schemes guarantee system availability except PESTO, which is vulnerable to Denial of Service (DoS) attacks since a single offline server can lead to a shutdown of the entire system. Our work and DAMFA are the only schemes ensuring real-time assessment due to the public decentralized and distributed architecture of the Ethereum blockchain. Moreover, all schemes are safe against fake manufacturer attacks, except OIDC and EOIDC, where tokens have no inherent protection against modification. As for impersonation and token replay attacks, the schemes EOIDC, PESTO, DAMFA and our work demonstrate resilience against these threats. In terms of anonymity and unlinkability, the schemes EL PASSO, DAMFA and our work are the only ones that enable users to interact anonymously with the system while preventing IdPs from tracking their activities and RPs from colluding. Although the PRIMA system also provides IdP unlinkability, it still shares personal information with RPs, which could enable colluding RPs to link several accounts to a single user. Finally, our proposed scheme and PRIMA are the only systems that provide access revocation and delegation. The comparison results showed that our proposed Blockchain-based OIDC system provides all nine security and privacy-preserving properties, demonstrating its high security and efficiency.

IX. CONCLUSION

In this paper, we proposed an efficient new model for OpenID connect protocol based on blockchain technology to ensure data security, integrity and availability and protect users' privacy. Our proposed system used the Ethereum blockchain and non-fungible token NFT standard to securely record OIDC critical parameters, namely the access token, id token, authorization code, and client credentials in the blockchain

network, verify the authenticity of entities asking for legitimate data and deal with validation errors. Furthermore, our system prevents IdPs from profiling users and protects them from colluding RPs by using the pseudonymity property of blockchain and modifying the attributes of the exchanged JWT tokens. We also provided a proof-of-concept, a cost and performance evaluation of the proposed system and analyzed its security formally using AVISPA and scyther tools and informally against different possible attacks. Moreover, we compared our proposed blockchain-based OIDC system against other existing similar schemes. Our system has proved that it's highly resilient against several attacks, such as replay, impersonation, fake token manufacturer, Man-In-The-Middle, etc. providing strong privacy guarantees while maintaining a fast and efficient authentication process, but it still has some limitations. Firstly, the system is still vulnerable to compromised IdP accounts threat, but it provides users with an access revocation mechanism in case they suspect that their accounts have been taken over. Secondly, IdPs must be online and available to generate the required OIDC tokens to authenticate users on RPs. To address these challenges in future work, we believe that a decentralized anonymous multifactor authentication mechanism with indirect communication between the IdP and RP would significantly enhance the security and privacy of the field.

APPENDIX A
THE PROPOSED SYSTEM DESCRIPTION IN HLPSL

```
role role_RP (A,B:agent, Ka,Kb:public_key, SND,RCV:channel(dy))
played_by A
def=
    local
        State:nat,
        AuthReq:text,
        Na,Nb,Nt:text,
        Ts:text,
        H:hash_func,
        CodeTk,ERCTk:text
   init
      State:= 0
        transition
          0. State=0 /\ RCV(start) =|>
             State'=2 /\ Na'=new() /\ SND({A.Na'}_Kb)

    2. State=2 /\ RCV({B.Na.Nb'}_Ka) =|>
             State'=4 /\ AuthReq'=new() /\ SND({Nb'.AuthReq'}_Kb)
                      /\ request(A,B,auth_1,Na)
                     /\ witness(A,B,auth_2,Nb')
         4. State=4 /\ RCV({Na.CodeTk'.ERCTk'.Ts'.Nt'.
            H(ERCTk'.Ts'.Nt').{H(ERCTk'.Ts'.Nt')}_inv(Kb)}_Ka) =|>
            State'=6
end role

role role_IdP (A,B:agent, Ka,Kb:public_key, H:hash_func,
SND,RCV:channel(dy)) played_by B
def=
     local
        State:nat,
        AuthReq:text,
        Ts:text,
        Na,Nb,Nt:text,
        CodeTk,ERCTk:text
    init
        State:= 1
   transition
     1. State=1 /\ RCV({A.Na'}_Kb) =|>
        State'=3 /\ Nb'=new() /\ SND({B.Na'.Nb'}_Ka)
                 /\ witness(B,A,auth_1,Na')

     3. State=3 /\ RCV({Nb.AuthReq'}_Kb) =|>
        State'=5 /\ CodeTk'=new() /\ ERCTk'=new() /\ Nt'=new()
       /\ Ts'=new() /\ SND({Na.CodeTk'.ERCTk'.Ts'.Nt'.
          H(ERCTk'.Ts'.Nt').{H(ERCTk'.Ts'.Nt')}_inv(Kb)}_Ka)
            /\ request(B,A,auth_2,Nb)
            /\ secret(CodeTk',sec_1,{A,B})
            /\ secret(ERCTk',sec_2,{A,B})
end role

role session(A,B:agent,Ka,Kb:public_key,H:hash_func)
def=
    local
 SND2,RCV2,SND1,RCV1:channel(dy)
    composition
 role_RP(A,B,Ka,Kb,SND1,RCV1) /\ role_IdP(B,A,Ka,Kb,H,SND2,RCV2)
end role

role environment()
    def=
      const
     ka,kb:public_key,
           authReq,ts:text,
           h:hash_func,
           code,ercTK:text,
           sec_1,sec_2,auth_1,auth_2:protocol_id,
           id_provider,rp:agent

 intruder_knowledge = {rp,id_provider,ka,kb}
 composition
  session(rp,id_provider,ka,kb,h)
           /\ session(rp,id_provider,ka,kb,h)
end role

goal
        secrecy_of sec_1
        secrecy_of sec_2
        authentication_on auth_1
        authentication_on auth_2

end goal

environment()
```

APPENDIX B
THE PROPOSED SYSTEM DESCRIPTION IN SPDL

```
usertype string;
usertype JWT;
usertype ERCToken;
usertype timestamp;
hashfunction H;


protocol blockchainOIDC(RP,IdP){

role RP {

     fresh AuthReq: string;
     fresh Na: Nonce;
     var Nb: Nonce;
     var Nt: Nonce;
     var T: timestamp;
     var codeTk: JWT;
     var ERCtk: ERCToken;

send_1(RP,IdP,{Na,RP}pk(IdP));
recv_2(IdP,RP,{IdP,Na,Nb}pk(RP));
send_3(RP,IdP,{Nb, AuthReq}pk(IdP));
recv_4(IdP,RP,{Na,codeTk,ERCtk,T,Nt,
 H(ERCtk,T,Nt),{H(ERCtk,T,Nt)}sk(IdP)}pk(RP));

claim_RP1(RP,Secret,codeTk);
claim_RP2(RP,Secret,ERCtk);
claim_RP3(RP,Alive);
claim_RP4(RP,Weakagree);
claim_RP5(RP,Niagree);
claim_RP6(RP,Nisynch);
}

role IdP{

    var AuthReq: string;
    var Na: Nonce;
    fresh Nb: Nonce;
    fresh Nt: Nonce;
    fresh T: timestamp;
    fresh codeTk: JWT;
    fresh ERCtk: ERCToken;

recv_1(RP,IdP,{Na,RP}pk(IdP));
send_2(IdP,RP,{IdP,Na,Nb}pk(RP) );
recv_3(RP,IdP,{Nb,AuthReq}pk(IdP));
send_4(IdP,RP,{Na,codeTk,ERCtk,T,Nt,
 H(ERCtk,T,Nt),{H(ERCtk,T,Nt)}sk(IdP)}pk(RP));

claim_IdP1(IdP,Secret,codeTk);
claim_IdP2(IdP,Secret,ERCtk);
claim_IdP3(IdP,Alive);
claim_IdP4(IdP,Weakagree);
claim_IdP5(IdP,Niagree);
claim_IdP6(IdP,Nisynch);
}
}
```

ACKNOWLEDGMENT

The authors express their gratitude to the anonymous reviewers and the associate editor for their valuable feedback on the article, which greatly contributed to enhancing its quality and presentation.

REFERENCES

[1] (2020). *World Internet Users Statistics and 2020 World Population Stats*. [Online]. Available: https://www.internetworldstats.com/stats.htm

[2] S. Gaw and E. W. Felten, "Password management strategies for online accounts," in *Proc. 2nd Symp. Usable Privacy Secur. (SOUPS)*, 2006, pp. 44–55, doi: 10.1145/1143120.1143127.

[3] S. Hammann, R. Sasse, and D. Basin, "Privacy-preserving OpenID connect," in *Proc. 15th ACM Asia Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 277–289, doi: 10.1145/3320269.3384724.

[4] D. Fett, R. Kuesters, and G. Schmitz, "The web SSO standard OpenID connect: In-depth formal security analysis and security guidelines," 2017, *arXiv:1704.08539*.

[5] (2023). *OpenID Market Share and Web Usage Statistics*. SimilarTech. [Online]. Available: https://www.similartech.com/technologies/openid

[6] M. Ghasemisharif, A. Ramesh, S. Checkoway, C. Kanich, and J. Polakis, "O single sign-off, where art thou? An empirical analysis of single sign-on account hijacking and session management on the web," in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)* 2018, pp. 1475–1492. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/ presentation/ghasemisharif

[7] D. Fett, R. Kuesters, and G. Schmitz, "A comprehensive formal security analysis of OAuth 2.0," 2016, *arXiv:1601.01229*.

[8] Z. Zhang, M. Król, A. Sonnino, L. Zhang, and E. Rivière, "EL PASSO: Privacy-preserving, asynchronous single sign-on," 2020, *arXiv:2002.10289*.

[9] C. Mainka, V. Mladenov, J. Schwenk, and T. Wich, "SoK: Single sign-on security—An evaluation of openID connect, in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Jan. 2017, pp. 251–266.

[10] J. Navas and M. Beltrán, "Understanding and mitigating OpenID connect threats," *Comput. Secur.*, vol. 84, pp. 1–16, Jul. 2019, doi: 10.1016/j.cose.2019.03.003.

[11] R. Weingärtner and C. M. Westphall, "A design towards personally identifiable information control and awareness in OpenID connect identity providers," in *Proc. IEEE Int. Conf. Comput. Inf. Technol. (CIT)*, Aug. 2017, pp. 37–46.

[12] N. Fotiou, I. Pittaras, V. A. Siris, S. Voulgaris, and G. C. Polyzos, "OAuth 2.0 authorization using blockchain-based tokens," 2020, *arXiv:2001.10461*.

[13] M. Schwartz and M. Machulak, *Securing the Perimeter: Deploying Identity and Access Management With Free Open Source Software*. Berkeley, CA, USA: Apress, 2018, pp. 151–203, doi: 10.1007/978-1-4842-2601-8.

[14] *Final: OpenID Connect Dynamic Client Registration 1.0 Incorporating Errata Set 1*. Accessed: Mar. 20, 2023. [Online]. Available: https://openid.net/specs/openid-connect-registration-1_0.html

[15] J. Bradley, N. Sakimura, and M. B. Jones. *JSON Web Token (JWT)*. Accessed: Mar. 10, 2023. [Online]. Available: https://tools.ietf.org/ html/rfc7519

[16] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H. Lee, "Systematic review of security vulnerabilities in Ethereum blockchain smart contract," *IEEE Access*, vol. 10, pp. 6605–6621, 2022, doi: 10.1109/ACCESS.2021.3140091.

[17] H. Guo and X. Yu, "A survey on blockchain technology and its security," *Blockchain, Res. Appl.*, vol. 3, no. 2, Jun. 2022, Art. no. 100067, doi: 10.1016/j.bcra.2022.100067.

[18] A. S. Rajasekaran, M. Azees, and F. Al-Turjman, "A comprehensive survey on blockchain technology," *Sustain. Energy Technol. Assessments*, vol. 52, Aug. 2022, Art. no. 102039, doi: 10.1016/j.seta.2022.102039.

[19] S. Paavolainen and C. Carr, "Security properties of light clients on the Ethereum blockchain," *IEEE Access*, vol. 8, pp. 124339–124358, 2020, doi: 10.1109/ACCESS.2020.3006113.

[20] S. Hakak, W. Z. Khan, G. A. Gilkar, B. Assiri, M. Alazab, S. Bhattacharya, and G. T. Reddy, "Recent advances in blockchain technology: A survey on applications and challenges," 2020, *arXiv:2009.05718*.

[21] V. Buterin. (2014). *Ethereum Whitepaper*. [Online]. Available: https://ethereum.org/en/whitepaper/

[22] *Introduction to Dapps*. Accessed: Feb. 10, 2023. [Online]. Available: https://ethereum.org/en/developers/docs/dapps/

[23] D. Mohan, L. Alwin, P. Neeraja, K. D. Lawrence, and V. Pathari, "A private Ethereum blockchain implementation for secure data handling in Internet of Medical Things," *J. Reliable Intell. Environments*, vol. 8, no. 4, pp. 379–396, Dec. 2022, doi: 10.1007/s40860-021-00153-2.

[24] *Introduction to Smart Contracts*. Accessed: Feb. 5, 2023. [Online]. Available: https://ethereum.org/en/developers/docs/smart-contracts/

[25] R. Zhang, R. Xue, and L. Liu, "Security and privacy on blockchain," 2019, *arXiv:1903.07602*.

[26] L. Chen, D. Moody, A. Regenscheid, A. Robinson, and K. Randall. (Feb. 3, 2023). *Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters*. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-186/final

[27] W. Entriken, D. Shirley, J. Evans, and N. Sachs. (Jan. 24, 2018). *EIP-721: ERC-721 Non-Fungible Token Standard*. Ethereum Improvement Proposals. [Online]. Available: https://eips.ethereum.org/EIPS/eip-721

[28] D. Fett, R. Küsters, and G. Schmitz, "SPRESSO," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 1358–1369, doi: 10.1145/2810103.2813726.

[29] R. Weingärtner and C. M. Westphall, "Enhancing privacy on identity providers," in *Proc. 8th Int. Conf. Emerg. Secur. Inf., Syst. Technol. (SECURWARE)*, 2014, pp. 1–7.

[30] M. R. Asghar, M. Backes, and M. Simeonovski, "PRIMA: Privacy-preserving identity and access management at internet-scale," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6, doi: 10.1109/ICC.2018.8422732.

[31] R. Deeptha and R. Mukesh, "Extending OpenID connect towards mission-critical applications," *Cybern. Inf. Technol.*, vol. 18, no. 3, pp. 93–110, Sep. 2018, doi: 10.2478/cait-2018-0041.

[32] W. Li, C. J. Mitchell, and T. Chen, "OAuthGuard," in *Proc. 5th ACM Workshop Secur. Standardisation Res. Workshop*, 2019, pp. 35–44, doi: 10.1145/3338500.3360331.

[33] W. Li and C. J. Mitchell, "User access privacy in OAuth 2.0 and OpenID connect," *IEEE Eur. Symp. Secur. Privacy Workshops (EuroS&PW)* Sep. 2020, doi: 10.1109/EUROSPW51379.2020.00095. [Online]. Available: https://ieeexplore.ieee.org/document/9229747

[34] C. Baum, T. Frederiksen, J. Hesse, A. Lehmann, and A. Yanai, "PESTO: Proactively secure distributed single sign-on, or how to trust a hacked server," *IEEE Eur. Symp. Secur. Privacy (EuroS&P)* Sep. 2020, pp. 587–606, doi: 10.1109/eurosp48549.2020.00044. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9230400

[35] O. Mir, M. Roland, and R. Mayrhofer, "Decentralized, privacy-preserving, single sign-on," *Secur. Commun. Netw.*, vol. 2022, pp. 1–18, Jan. 2022, doi: 10.1155/2022/9983995.

[36] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 2, pp. 198–208, Mar. 1983.

[37] *Ethereum Average Gas Price Chart|Etherscan*. Ethereum (ETH) Blockchain Explorer. Accessed: Feb. 11, 2023. [Online]. Available: https://etherscan.io/chart/gasprice

[38] *Ethereum Average Block Time Chart|Etherscan*. Ethereum (ETH) Blockchain Explorer. Accessed: Feb. 11, 2023. [Online]. Available: https://etherscan.io/chart/blocktime

[39] *Ethereum Average Gas Limit Chart|Etherscan*. Ethereum (ETH) Blockchain Explorer. Accessed: Feb. 11, 2023. [Online]. Available: https://etherscan.io/chart/gaslimit

[40] *OpenID Connect|Authentication*. Google Developers. Accessed: Feb. 10, 2023. [Online]. Available: https://developers.google.com/identity/openid-connect/openid-connect

[41] Y. Berguig, J. Laassiri, and S. Hanaoui, "Anonymous and lightweight secure authentication protocol for mobile agent system," *J. Inf. Secur. Appl.*, vol. 63, Dec. 2021, Art. no. 103007, doi: 10.1016/j.jisa.2021.103007.

[42] C. J. F. Cremers, "The Scyther tool: Verification, falsification, and analysis of security protocols," in *Computer Aided Verification*, vol. 5123. Jul. 2008, pp. 414–418, doi: 10.1007/978-3-540-70545-1_38.

[43] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P. C. Héam, O. Kouchnarenko, J. Mantovani, and S. Mödersheim, "The AVISPA tool for the automated validation of internet security protocols and applications," in *Computer Aided Verification*, vol. 3576. 2005, pp. 281–285, doi: 10.1007/11513988_27.

[44] S. Ziauddin and B. Martin, "Formal analysis of ISO/IEC 9798–2 authentication standard using AVISPA," in *Proc. 8th Asia Joint Conf. Inf. Secur.*, Seoul, South Korea, Jul. 2013, pp. 108–114, doi: 10.1109/asiajcis.2013.25.

[45] A. R. R. Shaikh and S. Devane, "Formal verification of payment protocol using AVISPA," *Int. J. Infonomics*, vol. 3, no. 3, pp. 326–337, Sep. 2010, doi: 10.20533/iji.1742.4712.2010.0035.

[46] D. Basin, S. Mödersheim, and L. Vigano, "OFMC: A symbolic model checker for security protocols," *Int. J. Inf. Secur.*, vol. 4, no. 3, pp. 181–208, Jun. 2005.

[47] M. Turuani, "The CL-Atse protocol analyser," in *Term Rewriting and Applications* (Lecture Notes in Computer Science), vol. 4098, F. Pfenning, Ed. Berlin, Germany: Springer, doi: 10.1007/11805618_21.

[48] A. Armando and L. Compagna, "SATMC: A SAT-based model checker for security protocols," in *Logics in Artificial Intelligence* (Lecture Notes in Computer Science), vol. 3229, Berlin, Germany: Springer, J. J. Alferes and J. Leite, Eds. 2004, doi: 10.1007/978-3-540-30227-8_68.
[49] Y. Boichut, P.-C. Ham, O. Kouchnarenko, and F. Oehl, "Improvements on the Genet and klay technique to automatically verify security protocols," in *Proc. Automated Verification Infinite States Syst. (AVIS)*, 2004, p. 84.
[50] S. S. Ahamad and A.-S. K. Pathan, "Trusted service manager (TSM) based privacy-preserving and secure mobile commerce framework with formal verification," *Complex Adapt. Syst. Model.*, vol. 7, no. 1, pp. 1–18, Aug. 2019, doi: 10.1186/s40294-019-0064-z.

BELFAIK YOUSRA received the master's degree in computer science with a focus on telecommunications systems and computer networks from the Polydisciplinary Faculty, Sultan Moulay Slimane University, Beni-Mellal, Morocco, in 2019, where she is currently pursuing the Ph.D. degree with a research focus on identity and access management systems, networks security and privacy, and blockchain.

SADQI YASSINE (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science with a focus on computer security from the Faculty of Sciences, Ibn Zohr University, Agadir, Morocco, in 2012 and 2015, respectively. From 2017 to 2021, he was an Assistant Professor with Sultan Moulay Slimane University. As of February 2021, he became an Associate Professor. He is the African Research Center of Information Technology Cybersecurity Vice President. He has made contributions in the fields of Web security, authentication protocols, network security, and cybersecurity. He has published several peer-reviewed research articles in international journals, book chapters, and conferences/workshops. He is a member of the ACM Professional and OWASP. Furthermore, he has served and continues to serve on executive and technical program committees and as a reviewer for numerous international conferences and journals.

MALEH YASSINE (Senior Member, IEEE) received the dual Ph.D. degree in computer sciences management. He is currently an Associate Professor of cybersecurity and IT governance with Sultan Moulay Slimane University, Morocco. He is the Founding Chair of the IEEE Consultant Network Morocco and the Founding President of the African Research Center of Information Technology Cybersecurity. He has published over 100 papers (book chapters, international journals, and conferences/workshops), 23 edited books, and five authored books. He is a member of the International Association of Engineers IAENG and the Machine Intelligence Research Labs. He received the Publons Top 1% Reviewer Award for the years 2018 and 2019. He was the Publicity Chair of BCCA 2019 and the General Chair of the MLBDACP 2019 Symposium and ICI2C 2021 Conference. He is the Editor-in-Chief of the *International Journal of Information Security and Privacy* and the *International Journal of Smart Security Technologies* (IJSST). He serves as an Associate Editor for IEEE Access (2019 Impact Factor 4.098), the *International Journal of Digital Crime and Forensics* (IJDCF), and the *International Journal of Information Security and Privacy* (IJISP). He is a Series Editor of *Advances in Cybersecurity Management* (CRC Taylor & Francis). He has served and continues to serve on executive and technical program committees and as a Reviewer for numerous international conferences and journals, such as *Ad Hoc Networks* (Elsevier), *IEEE Network* magazine, IEEE Sensors Journal, *ICT Express*, and *Cluster Computing* (Springer).

SAFI SAID received the B.Sc. degree from Cadi Ayyad University, Marrakech, Morocco, in 1995, the M.Sc. degree from Chouaib Doukkali University, in 1997, and the Ph.D. degree from Cadi Ayyad University, in 2002. He was a Professor of information theory and telecommunication systems with the National School of Applied Sciences, Tangier, Morocco, from 2003 to 2005. Since 2006, he has been a Professor of applied mathematics and computer science with the Polydisciplinary Faculty, Sultan Moulay Slimane University, Beni-Mellal, Morocco. In 2015, Sultan Moulay Slimane University approved his promotion to the rank of a Full Professor. His general research interests include the areas of communications and signal processing, estimation, time-series analysis, and system identification subjects. His current research interests include transmitter and receiver diversity techniques for single and multiuser fading communication channels, and wideband wireless communication systems.

TAWALBEH LO'AI (Senior Member, IEEE) received the M.Sc. (Hons.) and Ph.D. degrees in electrical computer engineering from Oregon State University, in 2004 and 2002, respectively. He is currently an Associate Professor with the Department of Computing and Cyber Security, Texas A&M University at San Antonio. Before that, he was a Visiting Researcher with the University of California at Santa Barbra. Since 2005, he has been teaching/developing more than 25 courses in different disciplines of computer engineering and science with a focus on cyber security for the undergraduate/graduate programs with the New York Institute of Technology (NYIT), DePaul University, and the Jordan University of Science and Technology. He has won many research grants and awards with over U.S. $2 million. He has over 80 research publications in refereed international journals and conferences.

KHALED SALAH (Senior Member, IEEE) received the B.S. degree in computer engineering with a minor in computer science from Iowa State University, USA, in 1990, and the M.S. degree in computer systems engineering and the Ph.D. degree in computer science from the Illinois Institute of Technology, USA, in 1994 and 2000, respectively. He is currently a Full Professor with the Department of Electrical and Computer Engineering, Khalifa University, United Arab Emirates. He has over 220 publications and three U.S. patents, has been giving a number of international keynote speeches, invited talks, tutorials, and research seminars on the subjects of blockchain, the Internet of Things (IoT), fog and cloud computing, and cybersecurity. He is also leading a number of projects on how to leverage blockchain for healthcare, 5G networks, combating deep fake videos, supply chain management, and AI. He is a member of the IEEE Blockchain Education Committee. He served as the Chair of the Track Chair for IEEE GLOBECOM 2018 on Cloud Computing. He is an Associate Editor of *IEEE Blockchain Technical Briefs*.