IEEE ICBC'24 1570977915

VELLET: Verifiable Embedded Wallet for Securing Authenticity and Integrity

***Abstract*—The blockchain ecosystem, particularly with the rise of Web3 and Non-Fungible Tokens (NFTs), has experienced a significant increase in users and applications. However, this expansion is challenged by the need to connect early adopters with a wider user base. A notable difficulty in this process is the complex interfaces of blockchain wallets, which can be daunting for those familiar with traditional payment methods. To address this issue, the category of "embedded wallets" has emerged as a promising solution. These wallets are seamlessly integrated into the front-end of decentralized applications (Dapps), simplifying the onboarding process for users and making access more widely available. However, our insights indicate that this simplification introduces a trade-off between ease of use and security. Embedded wallets lack transparency and auditability, leading to obscured transactions by the front end and a pronounced risk of fraud and phishing attacks. This paper proposes a new protocol to enhance the security of embedded wallets. Our VELLET protocol introduces a wallet verifier that can match the audit trail of embedded wallets on smart contracts, incorporating a process to verify authenticity and integrity. In the implementation architecture of the VELLET protocol, we suggest using the Text Record feature of the Ethereum Name Service (ENS), known as a decentralized domain name service, to serve as a repository for managing the audit trails of smart contracts. This approach has been demonstrated to reduce the necessity for new smart contract development and operational costs, proving cost-effective through a proof-of-concept. This protocol is a vital step in reducing security risks associated with embedded wallets, ensuring their convenience does not undermine user security and trust.**

***Index Terms*—blockchain, embedded wallet, decentralized applications, security audit**

I. Introduction

The blockchain-based ecosystem continues to attract a vast array of new participants, following the rise of Web3 and the Non-Fungible Token (NFT) market. Current estimates suggest that over 420 million users possess cryptographic assets [1]. As of December 2023, it is observed that there are approximately 15,000 decentralized applications (Dapps) and in excess of 431,000 smart contracts identified across upwards of 62 blockchains [2]. However, in order to achieve further growth, it is necessary to overcome the gap between the early adopters and the early majority, known as the "chasm".

Crossing this chasm faces several technical challenges. A prominent barrier is the complex user interfaces related to blockchain-specific services, such as wallets. For instance, Metamask [3], a popular non-custodial wallet among cryptocurrency enthusiasts, can be challenging for users accustomed to conventional payment systems. Recent extensive user interaction studies have revealed difficulties with wallets, including specific issues in the crypto-currency domain, such as transaction complexities, fees, address, and key management [4]. Many of these problems could be addressed by emulating the design of existing online banking and payment systems, which users are already familiar with [5].

To address these challenges, the emerging category of embedded wallets has attracted significant attention [6], with multiple companies now entering the distribution phase of these products [7]–[9]. These are seamless, non-custodial wallets instantly generated for blockchain applications. Unlike traditional non-custodial wallets, which exist independently outside a specific app but can connect to it, embedded wallets create a user's wallet in the background using conventional login credentials (such as email or social logins) when creating an account in the app. This eliminates the need to first create a wallet and then connect it to a Dapp account, thus favoring user onboarding to Dapp services.

However, there is generally a trade-off between usability and security. Our insights suggest that the simple and user-friendly appearance of embedded wallets could induce scam and phishing in wallet services. The most significant flaw is that embedded wallets do not offer transparency and auditability, making it impossible to prevent fraudulent activities. Users have no choice but to trust the representation of the embedded wallet displayed on the web application. Cryptocurrencies or NFTs they believe they have purchased may only appear on the embedded wallet and do not actually exist on the blockchain. Additionally, users may not immediately realize if the purchase amount has been manipulated to differ significantly from the order book.

In this paper, we propose a new wallet verification protocol that incorporates verifiability into embedded wallets. The protocol ensures that the wallet has been audited by an audit organization, such as a security audit company or a community of security experts and enthusiasts. Additionally, by linking the integrity of the embedded wallet with the domain of the Dapp front-end provided, it prevents tampering with the embedded wallet by external attackers. A distinctive implementation idea is to use the Text Records [10] of the Ethereum Name Service (ENS) [11], a decentralized name service for Ethereum, as a repository that links the domain to the integrity of the

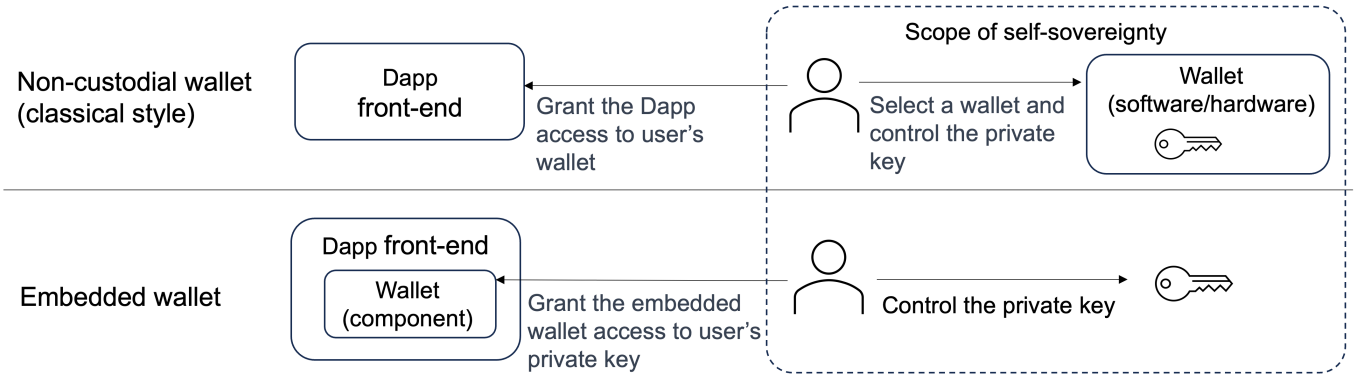

Fig. 1: Comparison of Embedded Wallets and Conventional Non-Custodial Wallets

resources of the wallet. This method is cost-effective as it ensures transparency while reducing the expenses associated with introducing and operating new contracts.

II. Background

A. Non-Custodial and Embedded Wallets

According to the taxonomy of wallets proposed by Erinle et al. [12], wallets are primarily categorized based on two factors: ”control over assets” and ”internet connectivity”. The term ”control over assets” refers to whether the wallet's private key is custodial, stored by a third party, or non-custodial, directly controlled by the user. Conversely, ”internet connectivity” distinguishes between hot wallets, which are always connected to the internet, and cold wallets, which are operated on physical hardware isolated from the internet.

Given the importance of sovereignty in crypto assets, non-custodial wallets are often preferred by users. These wallets are accessible through various means, including browser extensions or standalone applications. Notable examples include MetaMask [3], Argent [13], and Trust Wallet [14]. From the service provider's perspective, there is significant interest in hot wallets, as they facilitate user onboarding to their services without barriers.

Consequently, the emergence of cloud-based Wallet as a Service (WaaS) provided by cryptocurrency companies such as Coinbase [15] and Circle [16] is noteworthy. WaaS is a cloud-based service that delivers essential wallet functionalities, designed for seamless integration into the applications and services of businesses and developers. It provides APIs that enable the incorporation of wallet capabilities into business applications, primarily in non-custodial and hot wallet configurations. In terms of key management, WaaS frequently employs technologies such as Multi-Party Computation (MPC) to ensure that private keys remain under the control of the user, contrasting sharply with custodial solutions where key management is handled by third-party entities.

Embedded wallets constitute a specific category of WaaS. Targeted primarily at web application developers, these wallets are available as integrated components, including a JavaScript SDK and a user interface. This configuration allows developers to effortlessly embed them into their web service front-end, thus facilitating easy access to blockchain services for users. Figure 1 illustrates a rudimentary comparison between the design of a conventional non-custodial wallet and an embedded wallet. Embedded wallets contribute to improved user onboarding by simplifying the use of Dapps, reducing the extent of user control from the entire wallet to just the private key. Notable companies active in this field include Thirdweb [7], Privy [8], and Dynamic [9]. Since there is no clear definition of embedded wallets in the existing literature, we refer to their products and define them as follows in this paper:

- Offered as a non-custodial, internet-connected web wallet.
- A single wallet is provided for a single web application (different wallet spaces are provided for different web apps).
- Can be easily integrated as part of one's own app in the form of reusable components.
- The web app and wallet are tightly coupled, and exporting the private key is necessary to transfer it to an external wallet.
- The implementation method for key storage is not limited; various methods such as MPC, key sharding, and contract wallets are seen in practice. Some services allow the developers of the web application themselves to choose.

B. Challenges of Embedded Wallets

In 2022, the cryptocurrency trading sector experienced substantial losses due to fraudulent activities, which amount to approximately $5.9 billion [17]. A significant threat within this domain is phishing, wherein cybercriminals use deceptive emails or websites to illicitly acquire users' cryptocurrencies [18] or NFTs [19]. Although phishing strategies range from social engineering tactics like clone attacks to technical ap-

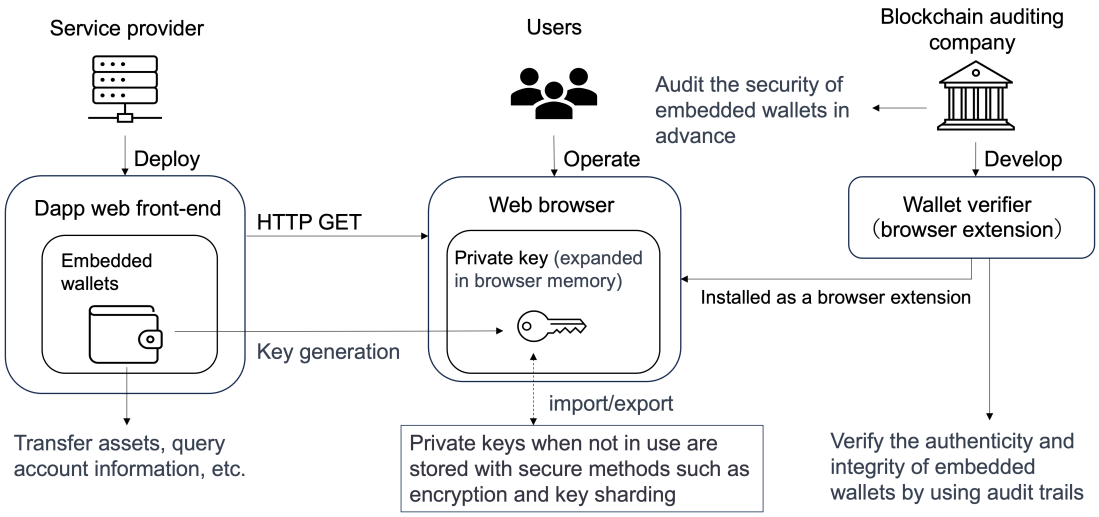

Fig. 2: Interactions between Service Providers, Users, and Audit Organizations

proaches such as DNS hijacking [18], these methods typically involve mimicking legitimate transactions to deceive users.

A proactive measure against such phishing in crypto assets involves meticulous verification of wallet displays [20]. These displays provide essential information, including transaction details, recipient addresses, and the amount being transferred. Vigilant examination of this information can alert users to potential irregularities. Additionally, some wallets are equipped with features to identify and warn against fraudulent accounts, enhancing security [21], [22]. This is particularly effective as conventional non-custodial wallets, which operate independently from Dapp front-ends, remain under user control.

Conversely, embedded wallets, which provide user interfaces akin to those in Web 2.0, can potentially enable more sophisticated phishing attacks. Compromised or counterfeit embedded wallets can alter or misrepresent crucial transaction information, thus mimicking the appearance of legitimate wallets. They might also falsely claim to have passed security audits to gain user trust. The core issues can be summarized as follows:

- Difficulty in distinguishing between legitimate and malicious embedded wallets.
- Inability of users to detect when an embedded wallet is compromised due to tampering with a legitimate Dapp's front-end.

These issues highlight challenges in ensuring both authenticity and integrity, which have long been central in digital security. To effectively address these challenges, various mechanisms and technologies have been developed historically. For instance, DNSSEC (Domain Name System Security Extensions) [23] offers a suite of extensions to DNS, providing clients with origin authentication, authenticated denial of existence, and data integrity. It employs public-key cryptography to confirm that DNS records remain unaltered, thereby preserving their authenticity and integrity. In DNSSEC, the root zone's public key is a common trust anchor, validating the authenticity of the entire root zone. Similarly, Code Signing [24] allows software distributors to certify the authenticity and integrity of their code. By digitally signing executables and scripts, developers can affirm that their code has not been modified or compromised. When combined with security audits, such as malware scans, it's feasible to distribute applications verified for trustworthiness and compatibility (e.g., Windows Driver Signing [25], macOS Code Siging [26], Android App Signing [27]). Here, the trust anchor is typically the platform operator managing the application distribution infrastructure, with client devices relying on the operator to verify code signatures.

Historically, operations ensuring authenticity and integrity have involved a third-party trust point, in addition to the data provider and recipient. This party validates the provider's data through mechanisms like digital signatures. However, in emerging digital frameworks employing blockchain technology, such as Web3, the move towards decentralization often leads to a dearth of mechanisms for ensuring authenticity and integrity. Embedded wallets in this context exemplify this issue, which remains largely unaddressed in existing literature.

III. VELLET Overview

A. Objective

This paper proposes the VELLET protocol, aimed at addressing the issues of authenticity and integrity within embedded wallets. VELLET seeks to fulfill the following requirements:

- Users can receive security warnings when using embedded wallets that have not been audited, addressing authenticity.
- Users can receive security warnings if the embedded wallet they are about to use has been compromised or tampered with, addressing integrity.
- The system is decentralized, transparent, and can be easily implemented.

B. Entities

The implementation of the VELLET protocol is realized through the cooperation of three principal entities. Figure 2 illustrates the relationship between the entities utilizing the VELLET protocol and the software they control. The assumptions in this paper are as follows:

1) **Service Providers.** Service providers provide decentralized applications (Dapps) linked to the blockchain. Their front-end employs embedded wallets, providing users with a means to connect to the blockchain. While embedded wallets may be offered via Wallet as a Service (WaaS), the sovereignty of the system rests with the service providers. Therefore, they hold the rights to control both the Dapp front-end and the embedded wallet.
2) **Users.** Users access the service provider's Dapp front-end through a Web browser. They manage the private keys used within the embedded wallet. It is important to note that while service providers supply the embedded wallet, they do not possess the private keys; hence, they are non-custodial. Users import their keys onto the embedded wallet and deploy them in the browser's memory only when using the wallet. Thus, only the user possesses and controls the rights to their private keys.
3) **Audit Organizations(e.g., security audit companies).** Audit organizations conduct security audits on the trustworthiness of embedded wallets and grant their approval. In the blockchain service industry, it's common for numerous companies to perform third-party audits to assess the security and soundness of smart contract code. However, the records or documentation of such audit trails are not widely available for public scrutiny. In VELLET, audit organizations submit an audit trail to the blockchain beforehand, and this trail is used to carry out the verification process for embedded wallets. VELLET introduces a new software tool called Wallet Verifier. Installed as a browser extension for the user, the Wallet Verifier utilizes audit trails to confirm the authenticity and integrity of embedded wallets. This paper posits that audit organizations are responsible for developing and maintaining the Wallet Verifier, thus preventing the centralization of authority with the service providers. An example of an auditing organization could be a blockchain-specialized security audit company [28], [29], or a community of security experts.

C. Architecture

We describe the overall architecture of the software modules necessary for the VELLET verification process. An overview of this architecture is shown in Figure 3.

Embedded Wallet. The embedded wallet software w is integrated within the Dapp front-end D, meaning $w \in D$. This web-based wallet w is optimally adjusted for the use of Dapp D and may have restricted functions compared to common non-custodial wallets such as Metamask. For instance, it might only generate private keys or receive tokens, omitting the functionality to sign payment transactions. (Specific use cases for this restricted embedded wallet are explained in detail in Section V-A.) w consists of JavaScript modules, and during the verification process, some fragmented scripts within this module are subject to inspection.

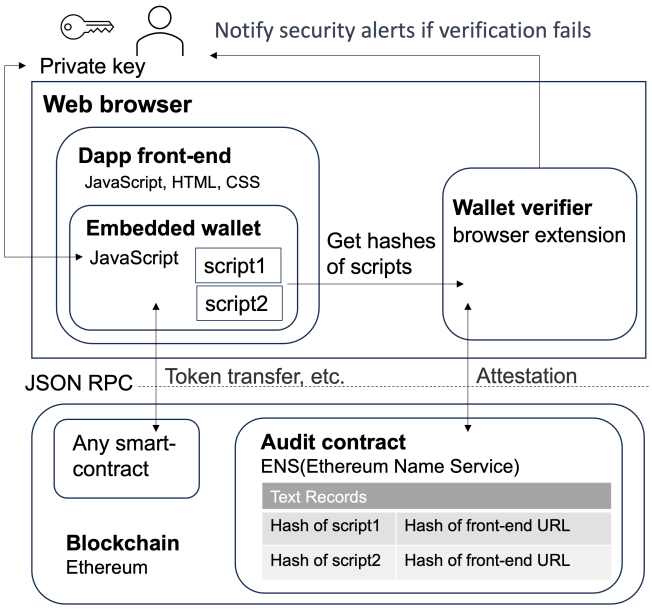

Fig. 3: Overview of the software architecture executing the proposed protocol.

Wallet Verifier. While embedded wallets help in user onboarding and provide excellent user experience, there is a risk that it could evolve into a centralized structure. The service provider that offers Dapp and hosts the embedded wallet could conceal most of the internal operations. Additionally, it is vulnerable to recognized attacks such as website changes due to Dapp hacking or phishing attempts that redirect users to fake sites. To counter these drawbacks, we introduce a verification process for the embedded wallet. Each time a user accesses the embedded wallet, the Wallet Verifier scrutinizes the wallet code and certifies its functionality. In particular, the embedded wallet is only executed if the code has not been tampered with and if code execution conditions (such as in encrypted communication) are met. This wallet verifier is designed as a browser extension that can analyze JavaScript modules on the Dapp front-end and the embedded wallet. Furthermore, the wallet verifier communicates with the audit contract, a smart contract on the blockchain, to attest to the integrity and authenticity of the embedded wallet. Attestation is performed by matching the script's hash against the audit trail previously submitted to the audit contract by the audit organization. (Details are discussed in Section IV.)

Audit Contract with ENS Text Records. Audit contract C is a type of smart contract executed on the blockchain. Its main purpose is to manage the audit trails of embedded wallets and to ensure their authenticity and integrity. Ideally, the

registration of audit trails in the contract should be conducted by a third-party audit organization, following the principles of decentralization, excluding the service provider. The audit trail is composed of essential audit-related data. This includes the URL (Uniform Resource Locator) of the Dapp front-end, which hosts the embedded wallet, and the hashes of the scripts used within the wallet. The audit contract publishes the audit trails on the blockchain for reference by other participants. This enables users and blockchain-connected wallet verifiers to inspect the audit results of the embedded wallet and evaluate its trustworthiness.

The point to be emphasized in our approach is the proposal to adopt the Ethereum Name Service (ENS), a decentralized naming service running on the Ethereum blockchain, as the foundational structure of the audit contract, instead of constructing and operating an audit contract from scratch. ENS, which consists of smart contracts, is a decentralized domain name system operational on the Ethereum blockchain, capable of holding text records of specific metadata and general information, not just domain names [10]. Registering information with ENS operating on the blockchain requires a signed transaction, ensuring the authenticity of the registrar as the domain owner. In this manner, an auditing organization can fulfill the same requirements as constructing an audit contract by registering audit trails as text records linked to domains. Utilizing ENS allows auditing organizations to reduce the costs of independently operating an audit contract and lowers the threshold for adopting the VELLET protocol.

IV. Protocol in detail

In this section, we explain the detailed workings of the VELLET protocol, which operates on the aforementioned architecture. The VELLET protocol is made up of two fundamental phases:

A. Audit Phase

For the sake of simplicity, this document describes the audit contract C using the most basic implementation. C consists of a key-value data store $C.store$ for storing audit trails, a stateless operation $C.regit()$ for registering audit trails A_t with $C.store$, and an operation $C.getAudit()$ for referencing A_t. The audit contract C is deployed on any blockchain that supports the execution of smart contracts. Only an audit organization with appropriate permissions can perform write operations that involve state changes to the audit contract C. The audit organization reviews the program code of the embedded wallet w and inspects its behavior during the execution of the code. A code audit contains the selection of appropriate cryptographic primitives, the use of secure libraries, and the absence of inappropriate external connections. The audit trail set, denoted as A_t, is assured to encompass at least the hash H_u of a Uniform Resource Locator (URL) string u, which identifies the location of the Dapp front-end; in other words, $H_u \in A_t$. Both the hash value H_u and H_w, representing the hash value of the embedded wallet w's code, are stored in the contract C as key-value pairs. Thus, the data store $C.store$ is updated as follows:

$$C.store \leftarrow TX_b(C.regit(H_w, A_t)) \tag{1}$$

Here, the execution of $C.regit(H_w, A_t)$ is performed through a blockchain transaction. That is, a valid execution transaction $TX_b(C.regit(H_w, A_t))$ on the blockchain network is broadcasted by an audit organization with registration permissions for the contract C. Through the execution process of the smart contract, the transaction is processed, and the pair (H_w, A_t) is stored in the data store $C.store$ defined in contract C. Using H_w as a query, the audit trails A_t can be retrieved as follows:

$$C.getAudit(H_w) \rightarrow A_t \quad \text{where} \quad H_u \in A_t \tag{2}$$

The *getAudit()* operation retrieves the audit trail set A_t from the store without changing the state of the smart contract. It does not require issuing a transaction to read the audit trails, and the value is publicly accessible. It should be noted that A_t contains the hash H_u of the URL u indicating the location of the Dapp front-end. The preimage of the pair (H_w, H_u) registered with C is (w, u), which serves as the trust anchor between the embedded wallet and the location where the Dapp front-end is hosted. Furthermore, A_t can include additional audit-related information, such as the audit completion date or the name of an audit company, but this document does not provide specific definitions for these to ensure clarity.

B. Verification Phase

The previous subsection explained how the trust anchor of the embedded wallet w is established within the audit contract C during the audit phase. In this section, we propose a protocol that utilizes this trust anchor to enable users to safely use the wallet w. As mentioned in Section III-C, the user's client terminal requires software for the wallet verification. A wallet verifier *Ins* is delivered to the client from a repository by an entity different from the Dapp service provider and is installed on the client. *Ins* incorporates the operations *verify()* for wallet verification and *execute()* for code execution. The *verify()* operation returns the result of wallet verification as follows:

$$Ins.verify(w, u, ca, cond) \in \{\text{true}, \text{false}\} \tag{3}$$

Here, ca indicates the access information for the audit contract, such as the contract's address or Application Binary Interface (ABI). *cond* represents additional verification conditions. For example, it can be used to verify the name of the audit agency or the execution environment. The verification uses the audit contract as a trust anchor to verify that the wallet w and the URL u indicating the provider have been audited. Furthermore, this verification process ensures that the wallet w has not been tampered with, demonstrating resistance to phishing attacks. Additionally, the execution permission for wallet w resides in the wallet verifier *Ins*. In other words, *Ins* implements the following function:

$$Ins.execute(w) \rightarrow \text{result} \tag{4}$$

The *execute()* function executes the functionalities provided by the embedded wallet w. Importantly, the execution permission is granted to *Ins*, not to the Dapp front-end where the wallet is embedded. By decentralizing permissions among entities, this approach eliminates a single point of failure and distributes security risks, enhancing overall security of the system. In contrast to conventional feature-rich wallets, wallets in this context are designed with simplicity in mind and only possess the necessary functionalities. Furthermore, by applying the principles of modular design, components such as key generation can be provided as reusable separate modules. Therefore, w can be embedded in the Dapp as several modules represented by $w_i, i = 0, 1, 2....$ In such cases, Equation (3) can be extended as follows:

$$\text{Verification} = \begin{cases} \text{true} & \text{if } \forall i, \textit{Ins.verify}(w_i, u, ca, cond) = \text{true} \\ \text{false} & \text{otherwise} \end{cases} \tag{5}$$

In our implementation, discussed in Section V, the wallet verifier *Ins* alerts the user with a security warning when the verification result is false. This mechanism provides resistance against fraudulent Dapp websites by informing users of potential risks such as wallet tampering w_i or inadequate URL auditing u.

V. Implementation

To verify the feasibility of the proposed protocol, we worked on its implementation. In this section, we explain the details of the implementation to facilitate a comprehensive understanding of the protocol.

A. Applications and Scenarios

Embedded wallets are designed to make using Dapps as easy as traditional Web 2.0 apps, aiming to boost their widespread adoption. A key application in this context is the distribution of utility-focused NFTs (Non-Fungible Tokens), moving away from speculative trading. In our VELLET proof-of-concept, we explore using utility-based NFTs for digital identity. As described by Weyl et al. [30], Soulbound Tokens (SBTs) are non-transferable NFTs employed to articulate the personal attributes such as identity, skills, experiences, and qualifications of the individual to whom they are assigned. This allows for the potential use of SBTs as a means to bridge digital identities, social bonds, and trust verification between the blockchain economy and the societal context. Accordingly, we have constructed a scenario that leverages embedded wallets for the deployment of SBTs via a Dapp.

Use Case: The user is a member of a specific community (e.g., an art club, sports club, or an open-source developer community). The community site issues SBTs to its members as proof of membership. The user can present SBT to third-party institutions outside the community site. Unlike credit scoring systems provided by centralized entities, SBTs demonstrates bottom-up social creditworthiness. Third-party institutions can evaluate the user's activities within the community and provide certain benefits (such as unsecured loans, voting rights, job referrals, etc.) based on trust relationships.

Scenario I: Distribution of SBTs

1) A user logs in to the website of a community.
2) After logging in, the user requests the issuance of a membership SBT from their profile page.
3) The site generates an encrypted private key through the embedded wallet.
4) The SBT is issued to the address associated with the private key.
5) The user exports the private key.

Scenario II: Presentation of SBTs

1) The user accesses a website dedicated to presenting or submitting SBTs.
2) The user imports the encrypted private key into the embedded wallet of the website.
3) The embedded wallet generates an electronic signature and sends it to the website.
4) The website verifies and authenticates the signature.

B. Implementation of Proof-of-Concept

We have developed an implementation for a proof-of-concept of VELLET based on the scenario we devised. For the web front-end of the Dapp, we have chosen the Next.js framework, and for the issureing of SBTs, we are utilizing Ethereum's Goerli testnet. The SBT standard adopts ERC-5192 [31] (Minimal Soulbound NFTs) and is backward compatible with the well-known NFT standard, ERC-721 [32]. We have adopted ethers.js for the construction of the embedded wallet. Ethers.js is a JavaScript library that allows developers to interact with the Ethereum blockchain and its ecosystem, providing wallet functionality (such as private key generation and transaction signing) that operates with JavaScript on a browser. The wallet verifier has been developed to adhere to the standards of Chrome Extension Manifest V3. The audit contract with which the wallet verifier interacts, as mentioned in Section III, utilizes the ENS (Ethereum Name Service) Text Records [10]. In this implementation, we used the ENS deployed on the Goerli testnet.

Let us clarify the parts related to the VELLET protocol among the scenarios. As mentioned above, the VELLET protocol is realized through the collaboration of the embedded wallet, audit contract, and wallet verifier. The embedded wallet is used in Scenarios I-3 to I-5 and Scenarios II-2 to II-3 steps. In these scenarios, the embedded wallet operates only when cryptographic operations involving the user's private key are required. The wallet is seamlessly integrated into the application, naturally embedded within the user interface interaction flow of the application. In the specific implementation, the embedded wallet is described as a client-side JavaScript module executable within the browser and called within the Dapp's website. The critical aspect to emphasize is the cryptographic operations are exclusively executed within the user's browser, never on the server side, ensuring that the embedded wallet is inherently non-custodial.

Algorithm 1 Script Verification in Wallet Verifier

```
procedure EXTRACT AND VERIFY SCRIPTS
    tabId ← getActiveTabId()
    tabUrl ← getActiveTabUrl(tabId)
    pageUrlHash ← getHash(tabUrl)
    html ← getInnerHTMLFromTab(tabId)
    pageDoc ← parseHTML(html)
    Extract scripts from pageDoc and save in scripts
    for each script in scripts do
        Compute hash of script and append to scriptDigests
    end for
    provider ← connectToWeb3Provider()
▷ The following example uses ENS as an audit contract.
    ens ← configureENS(provider)
    domain ←′ vellet.eth′
    for each digest in scriptDigests do
        Get text record for digest from ENS and save result in scriptChecks
        textRecord ← ens.name(domain).getText(digest)
        isValid ← (textRecord == pageUrlHash)
        Append isValid to scriptChecks
    end for
    if scriptChecks contains any false value then
        setIsNotSecure(true)
    else
        setIsNotSecure(false)
    end if
end procedure
```

The audit contract, which employs ENS's Text Records [10], serves as a searchable data store. It facilitates the retrieval of audit trails for embedded wallets by using the hash value of the Dapp front-end site URL as a query. The audit trails are audit results in which an audit company has pre-inspected the module code of each embedded wallet, as described in the audit phase in Section IV-A. Examples of audit points at this stage include the following:

- Selection of appropriate cryptographic primitives.
- Setting appropriate seed values for key generation.
- Prevention of content being sent to external servers (e.g., not sending the private key).

The main function of the wallet verifier, implemented as a browser extension, is to attest to the legitimacy of the embedded wallet and then initiate wallet operations. This process of attestation is crucial to prevent execution in potentially fraudulent embedded wallets or on counterfeit Dapp websites. Although several implementation approaches are possible, in our design, the wallet verifier, when deployed as a browser extension, processes the HTML document to extract the wallet code encapsulated within script tags. This code, as previously indicated, is modularized as a JavaScript component and is processed as a string in our algorithm. Algorithm 1 presents the pseudocode for this wallet verifier. Given its design as a browser extension, the verifier has the capability to read the HTML Document of the currently active browser tab. It fetches the audit outcomes for the vetted Dapp by using the hash of the embedded wallet code within the HTML as the reference key. The audit data incorporates the hash of the Dapp's URL, facilitating the cross-checking of the Dapp website URL being accessed by the user against its audited status.

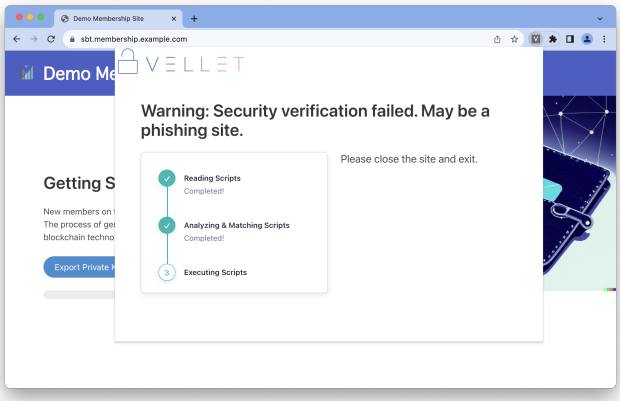

Fig. 4: The user interface of the wallet verifier, which pops up on the website, displays a warning when the verification of an embedded wallet fails.

If the audit trails of Dapps cannot be obtained, that is, if the hash value of the Dapp URL is not registered on the ENS Text Record, this means the authenticity of the embedded wallet is not secured. Furthermore, if the audit trails are obtained and they do not include the code hash of the embedded wallet, it indicates a loss of the wallet's integrity. Therefore, ENS records are necessary to act as a trust anchor. The resistance to data tampering and the authenticity provided by blockchain infrastructure further strengthen this assertion. In our proof-of-concept, as mentioned in Section III, if either authenticity or integrity is negated, a warning is displayed to the user as shown in Figure 4.

C. Evaluation of Introduction Cost

In order to address the security shortcomings of embedded wallets, our introduced protocol's fundamental idea involves incorporating audit processes from an audit organization into the wallet system. The audit organization, which in this implementation is assumed to be a security audit company, is required to provide an audit contract built using smart contracts. Generally, developing and maintaining smart contracts requires high technical expertise, including scalable contract design and robust security measures. These requirements can be challenging for a typical organization to meet. In our approach, we utilize the Ethereum Name Service (ENS) to facilitate the audit contract. ENS, built on smart contract bases and managed by a Decentralized Autonomous Organization (DAO), is a distributed naming service. Using the TextRecord feature [10] in ENS, a mechanism for custom records, eliminates the need for developing audit contracts from scratch, thereby saving time and costs involved in implementation.

The costs necessary for the deployment of an audit contract using ENS are shown in Table 1. As all operations are performed via smart contracts, the costs include transaction gas fees and ENS fees. The initial setup cost, based on the ETH token price as of November 12, 2023, is below $30.

Action	Gas Used	USD
ENS registration fee (1 year)	-	$5.00
Registration - Start timer	44,206	$2.27
Registration - Register name	275,933	$14.16
Update text records	136,234	$7.01
Total Cost	**-**	**$28.44**

TABLE I: Costs for ENS and Related Operations (25 Gwei Gas Price, 1 ETH = $2,057 on November 12, 2023)

Regarding operational costs, while gas fees for transactions are not required for blockchain reference operations, maintaining blockchain nodes incurs costs. Depending on the setup, whether on-premises or using cloud services, costs may vary. A common cost-effective method is to use Ethereum JSON-RPC endpoints such as Infura [33]. According to Infura's pricing, up to 100,000 transactions per day are free, and 1 million transactions per day cost $225 per month. This can be adjusted based on the scale, such as the number of users utilizing the wallet verifier.

Based on these calculations, it is demonstrated that audit organizations can feasibly construct audit contracts using ENS with realistic cost implications. Moreover, not only is the mechanism cost-effective, but since all registration operations on the audit contract are conducted via blockchain transactions, it is resistant to tampering of audit trails and offers high transparency, allowing anyone to access the audit trails.

VI. Related Work

Classification of Embedded Wallets. Erinle et al. [12], through an extensive literature survey, have established a comprehensive classification of cryptocurrency wallets, analyzing the unique characteristics and related security considerations for each wallet category. To the best of our knowledge, their work appears to provide the most comprehensive classification of wallets available to date. However, recent developments in wallet forms, namely embedded wallets, have not been adequately addressed in the literature. Although the term "embedded wallet" appears in recent work [34] where the burgeoning concept of Web3 is explored from the perspective of blockchain architecture, this term describes wallets operating within a browser, akin to browser extensions such as Metamask, but does not delineate a distinct category. Conversely, in the industrial realm, embedded wallets are increasingly being recognized as a distinct form of cryptocurrency wallet offerings [6]. Therefore, in an academic context, our study appears to be the first to elucidate the structural features of embedded wallets and systematically categorize security concerns.

Combatting Phishing and Scams. Research to prevent phishing scams related to blockchain projects has been increasing recently. In particular, scam account detection has been widely performed, with known methods including those based on traditional machine learning techniques specializing in feature extraction [35], and those using graph network embedding methods [36], [37]. However, in account-based detection, it is difficult to prevent phishing where the entire front-end behaves fraudulently. Therefore, approaches like those of Roy et al. [38], which distinguish NFT phishing websites by features of the website and analyze phishing URLs to include them in browser protection tool blocklists, would be effective.

However, these existing approaches, which mainly rely on identifying fraudulent activities through external observations, have limitations. Our research represents a first step forward in incorporating mechanisms into wallets that ensure authenticity and integrity, classic challenges in security. This approach is not intended to replace existing methods but to complement them.

Auditing Blockchain Projects. Our approach leverages auditing practices used in Web3 and blockchain projects. As the idiom "Don't trust, verify" is commonly used in the blockchain community, audits of blockchain projects are conducted extensively across various aspects. For example, Proof of Reserves in the audit of cryptocurrency exchange reserves is well known, and beyond this, various methods have been developed to provide auditability to individuals and regulatory authorities [39]. Undergoing security audits at the launch of blockchain projects has become common for gaining user trust. In particular, academia has proposed many tools for smart contract security audits [40], and numerous blockchain-specialized security audit companies have been established in the industry, such as CertiK [28] and Hacken [29]. In particular in the domain that our paper addresses, reports on the security audits of wallets have been published [41]. On the other hand, the exploration of mechanisms for utilizing audit reports, as proposed by us, to verify audit subjects and alert users about potential authenticity and completeness deficiencies, remains an uncharted territory.

VII. Conclusion and Future Work

This paper has clarified the security characteristics of recently emerging embedded wallets and discussed the potential to induce fraudulent transactions. As a solution, we proposed the VELLET protocol, a verifiable embedded wallet that incorporates the activities of audit organizations. Our proposal has been demonstrated to be implementable at a feasible cost through proof-of-concept.

Similar to embedded wallets, many blockchain projects prioritize user convenience and corporate profits, often at the expense of security. We believe that providing users with defenses against fraudulent Dapps is of utmost importance. In this study, we focused on securing wallet authenticity and integrity, utilizing by audits and their trails as a key to user security. We consider this approach to be extendable not only to embedded wallets but also to various aspects of Dapps, for instance, to entire websites and smart contracts, and we have positioned it as a future challenge.

REFERENCES

[1] Triple-A, "Cryptocurrency ownership data," accessed: 2023-11-23. [Online]. Available: https://triple-a.io/crypto-ownership-data/

[2] DappRadar, "Dapps Industry Overview," accessed: 2023-12-03. [Online]. Available: https://dappradar.com/industry-overview

[3] MetaMask, "The crypto wallet for defi, web3 dapps and nfts," accessed: 2023-11-23. [Online]. Available: https://metamask.io/

[4] M. Fröhlich, F. Waltenberger, L. Trotter, F. Alt, and A. Schmidt, "Blockchain and cryptocurrency in human computer interaction: A systematic literature review and research agenda," in *Proceedings of the 2022 ACM Designing Interactive Systems Conference (DIS '22)*, 2022, pp. 155–177.

[5] A. Voskobojnikov, O. Wiese, M. M. Koushki, V. Roth, and K. Beznosov, "The U in crypto stands for usable: An empirical study of user experience with mobile cryptocurrency wallets," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*, 2021, pp. 1–14.

[6] O. Ohayon, "Personal wallets vs. embedded wallets: Who wins in crypto?" March 2023, accessed: 2023-11-23. [Online]. Available: https://zengo.com/personal-wallets-vs-embedded-wallets-who-wins/

[7] Thirdweb, "Embedded wallets - overview," accessed: 2023-11-23. [Online]. Available: https://portal.thirdweb.com/embedded-wallet

[8] Privy, "Embedded wallets documentation," accessed: 2023-11-23. [Online]. Available: https://docs.privy.io/guide/frontend/embedded/overview

[9] Dynamic, "Overview of embedded wallets," 2023, accessed: 2023-11-23. [Online]. Available: https://docs.dynamic.xyz/embedded-wallets/overview

[10] R. Moore, "ENSIP-5: Text Records," May 2017, ENS Improvement Proposals, no. 5. [Online]. Available: https://docs.ens.domains/ens-improvement-proposals/ensip-5-text-records

[11] ENS, "Ethereum name service: Decentralised naming for wallets, websites, & more," accessed: 2023-11-23. [Online]. Available: https://ens.domains

[12] Y. Erinle, Y. Kethepalli, Y. Feng, and J. Xu, "Sok: Design, vulnerabilities, and security measures of cryptocurrency wallets," 2023, arXiv:2307.12874.

[13] Argent, "Argent – the best ethereum wallet for defi and nfts," 2023, accessed: 2023-11-23. [Online]. Available: https://www.argent.xyz/

[14] Trust Wallet, "Best crypto wallet for web3, nfts and defi," accessed: 2023-11-23. [Online]. Available: https://trustwallet.com/

[15] Coinbase, "Waas - coinbase cloud," accessed: 2023-11-23. [Online]. Available: https://www.coinbase.com/cloud/products/waas

[16] Circle, "Programmable wallets — wallet as a service," accessed: 2023-11-23. [Online]. Available: https://www.circle.com/en/programmable-wallets

[17] Chainalysis, "The Chainalysis 2023 Crypto Crime Report," 2023, accessed: 2023-11-23. [Online]. Available: https://go.chainalysis.com/2023-crypto-crime-report.html

[18] A. A. Andryukhin, "Phishing attacks and preventions in blockchain based projects," in *2019 International Conference on Engineering Technologies and Computer Science (EnT)*, Moscow, Russia, 2019, pp. 15–19.

[19] J. Yang, J. Liu, and J. Wu, "With trail to follow: Measurements of real-world non-fungible token phishing attacks on ethereum," 2023, arXiv:2307.01579.

[20] Trust Wallet Community, "How to spot a phishing attack & protect your crypto," 2023, accessed: 2023-11-23. [Online]. Available: https://community.trustwallet.com/t/how-to-spot-a-phishing-attack-protect-your-crypto/753663

[21] MetaMask Support, "How to turn on blockaid security alerts," accessed: 2023-11-23. [Online]. Available: https://support.metamask.io/hc/en-us/articles/19878220833947-How-to-turn-on-Blockaid-security-alerts

[22] Trust Wallet Community, "Introducing the trust wallet security scanner: Making crypto & web3 safer for everyone," 2022, accessed: 2023-11-23. [Online]. Available: https://community.trustwallet.com/t/introducing-the-trust-wallet-security-scanner-making-crypto-web3-safer-for-everyone/643056

[23] D. E. Eastlake 3rd, "Domain Name System Security Extensions," RFC 2535, Mar. 1999. [Online]. Available: https://www.rfc-editor.org/info/rfc2535

[24] D. Cooper, A. Regenscheid, M. Souppaya, C. Bean, M. Boyle, D. Cooley, and M. Jenkins, "Security considerations for code signing," *NIST Cybersecurity White Paper*, 2018. [Online]. Available: https://doi.org/10.6028/NIST.CSWP.01262018

[25] Microsoft, "Driver signing - windows drivers," https://learn.microsoft.com/en-us/windows-hardware/drivers/install/driver-signing, May 2023, accessed: 2023-11-23.

[26] Apple, "About code signing," September 2016, accessed: 2023-11-23. [Online]. Available: https://developer.apple.com/library/archive/documentation/Security/Conceptual/CodeSigningGuide

[27] Google for Developers, "Sign your app - android studio," https://developer.android.com/studio/publish/app-signing, 2023, accessed: 2023-11-23.

[28] CertiK, "Web3 security leaderboard," accessed: 2023-11-27. [Online]. Available: https://www.certik.com/

[29] Hacken, "Blockchain security services company - web3, crypto, defi," accessed: 2023-11-27. [Online]. Available: https://hacken.io/

[30] E. G. Weyl, P. Ohlhaver, and V. Buterin, "Decentralized society: Finding web3's soul," 2022, available at SSRN. [Online]. Available: https://ssrn.com/abstract=4105763

[31] T. Daubenschütz and Anders, "ERC-5192: Minimal Soulbound NFTs," July 2022, Ethereum Improvement Proposals, no. 5192. [Online]. Available: https://eips.ethereum.org/EIPS/eip-5192

[32] J. E. William Entriken, Dieter Shirley and N. Sachs, "ERC-721: Non-Fungible Token Standard," Jan 2018, Ethereum Improvement Proposals, no. 721. [Online]. Available: https://eips.ethereum.org/EIPS/eip-721

[33] Infura, "Web3 development platform — ipfs api & gateway — blockchain node service," 2023, accessed: 2023-11-23. [Online]. Available: https://www.infura.io/

[34] Q. Wang, R. Li, Q. Wang, S. Chen, M. Ryan, and T. Hardjono, "Exploring web3 from the view of blockchain," 2022, arXiv:2206.08821.

[35] W. Chen, X. Guo, Z. Chen, Z. Zheng, and Y. Lu, "Phishing scam detection on ethereum: Towards financial security for blockchain ecosystem." in *IJCAI*, vol. 7, 2020, pp. 4456–4462.

[36] J. Wu, Q. Yuan, D. Lin, W. You, W. Chen, C. Chen, and Z. Zheng, "Who are the phishers? phishing scam detection on ethereum via network embedding," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 2, pp. 1156–1166, 2020.

[37] S. Li, R. Wang, H. Wu, S. Zhong, and F. Xu, "Siege: Self-supervised incremental deep graph learning for ethereum phishing scam detection," in *Proceedings of the 31st ACM International Conference on Multimedia*, 2023, pp. 8881–8890.

[38] S. S. Roy, D. Das, P. Bose, C. Kruegel, G. Vigna, and S. Nilizadeh, "Unveiling the risks of nft promotion scams," 2023, arXiv:2301.09806.

[39] P. Chatzigiannis, F. Baldimtsi, and K. Chalkias, "Sok: Auditability and accountability in distributed payment systems," in *International Conference on Applied Cryptography and Network Security*. Springer, 2021, pp. 311–337.

[40] S. Chaliasos, M. A. Charalambous, L. Zhou, R. Galanopoulou, A. Gervais, D. Mitropoulos, and B. Livshits, "Smart contract and defi security: Insights from tool evaluations and practitioner surveys," 2023, arXiv:2304.02981.

[41] MetaMask, "Security bug bounties," accessed: 2023-11-27. [Online]. Available: https://metamask.io/security/