



City Research Online

City St George's, University of London

Citation: Dziembowski, S., Ebrahimi, S., Kędzior, P., Król, M. & Lizurej, T. (2026). PEACE: Privacy-Enhanced Authentication for Cryptocurrency Environments. Proceedings of the 23rd International Conference on Security and Cryptography, 1, pp. 317-324. doi: 10.5220/0015085400004103 ISSN 2184-7711 doi: 10.5220/0015085400004103

This is the published version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: <https://openaccess.city.ac.uk/id/eprint/38401/>

Link to published version: <https://doi.org/10.5220/0015085400004103>

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to [City Research Online policy](#).

PEACE: Privacy-Enhanced Authentication for Cryptocurrency Environments

Stefan Dziembowski¹^a, Shahriar Ebrahimi²^b, Paweł Kędzior^{1,5}^c, Michał Król³^d
and Tomasz Lizurej⁴^e

¹University of Warsaw, Poland

²Alan Turing Institute, U.K.

³City, University of London, U.K.

⁴NASK, Poland

⁵IDEAS Research Institute, Poland

Keywords: Authentication, Privacy, Blockchains, Zero-Knowledge Proofs, OpenID Connect.

Abstract: As blockchains gain traction in managing financial assets, robust authentication mechanisms become essential. Traditional public-private key authentication poses risks of key mismanagement, leading to significant financial losses. At the same time, in the non-blockchain context, systems like OpenID Connect (OIDC) provide user-friendly authentication with identity recovery in the traditional Web. However, they rely on a single trusted party, raising privacy concerns when applied to blockchains. In this paper, we present PEACE, an authentication mechanism for blockchains that combines the practicality of traditional Web systems with decentralization and privacy-preservation, building on a recent groundbreaking *zkLogin* protocol of Baldimtsi et al. (ACM CCS 2024). The protocol frees users from managing cryptographic material without giving away fund custody or relying on a single party. We detail PEACE's design, prove its properties in the Universal Composability framework, and present a real-world implementation compatible with existing OIDC deployments and the Ethereum blockchain.

1 INTRODUCTION

Blockchains have emerged as pivotal technologies in the management of significant financial assets. As of early 2025, the total market capitalization of the largest blockchains, such as Bitcoin and Ethereum, surpasses \$1 trillion (YCharts, 2025; CoinGecko, 2025). The secure access to digital wallets, enabling users to store and transfer their funds, hinges on robust authentication mechanisms. Traditionally, blockchain authentication relies on public-private key pairs. The public key is embedded within the user's address, while the private key is used to sign transactions. Thus, the private key is extremely sensitive, and its security is paramount. However, managing cryptographic keys is notoriously challenging for users. Numerous instances of lost, leaked, or stolen keys have led to significant financial losses—it is estimated

that about 20% of all Bitcoin (worth roughly \$140 billion) is inaccessible due to forgotten or mismanaged keys (Markets Insider, 2021). For example, one user (Stefan Thomas) lost the password to an IronKey holding 7,002 BTC (\$235M) after exhausting his limited login attempts (Times, 2021), illustrating how a single mistake can permanently lock away a fortune.

OpenID Connect (OIDC). In contrast, the traditional Web relies on user-friendly authentication mechanisms such as OpenID Connect (OIDC) (Sakimura et al., 2014; Sharif et al., 2022). Major providers (Google, Facebook, etc.) act as Identity Providers (IdPs), issuing signed JSON Web Token (JWT) tokens that allow single sign-on across multiple services. OIDC improves usability, reduces password fatigue, and allows recovery of lost credentials. Yet, directly reusing OIDC for blockchains is problematic: IdPs observe every login, can impersonate their users, and may be unavailable or compromised, threatening liveness and sovereignty (Brands, 2007; Zhang et al., 2021; Li and Mitchell, 2020).

Prior Work on OIDC for Blockchain Authentication. A growing body of work has sought to adapt

^a <https://orcid.org/0000-0002-6914-6425>

^b <https://orcid.org/0000-0003-0344-921X>

^c <https://orcid.org/0000-0003-2270-8694>

^d <https://orcid.org/0000-0002-3437-8621>

^e <https://orcid.org/0000-0001-8563-4325>

OIDC for blockchains (Heilman et al., 2023; Palladino, 2019; Park et al., 2023; Baldimtsi et al., 2024). Early systems explored account recovery using JWTs but exposed sensitive information on-chain and trusted the IdP too strongly (Heilman et al., 2023; Palladino, 2019). Subsequent work incorporated zero-knowledge proofs to improve privacy (Park et al., 2023), but remained impractical due to expensive proof production, non-standard signature schemes, and the reliance on a single trusted IdP. The first truly practical protocol in this line of work is *zkLogin*, proposed in the groundbreaking work (Baldimtsi et al., 2024), which introduced ephemeral on-the-fly key rotation authenticated through OIDC. ZkLogin improves practicality by avoiding long-term key storage, hiding sensitive OIDC attributes via zero-knowledge proofs, and supporting direct deployment in Sui (Labs, 2024). ZkLogin thus marks a significant milestone: it shows that OIDC can serve as the foundation for blockchain authentication without requiring changes to existing IdPs.

Open Challenges. Despite its numerous important advantages, (Baldimtsi et al., 2024) still leaves some open challenges, which we list below.

- (C1). Users must still manage cryptographic material (called *salt* in (Baldimtsi et al., 2024)), which must remain confidential and synchronized across devices.
- (C2). Authentication information (salt and IdP choice) cannot be updated once set.
- (C3). The system lacks account recovery if the salt is lost.
- (C4). zkLogin relies on external servers to generate zero-knowledge proofs, leaking user credentials.
- (C5). A single IdP signs the OIDC token, creating a single point of failure and exposing the chosen IdP to targeted attacks.
- (C6). The security analysis is game-based only; it does not provide guarantees under arbitrary composition as required in real-world deployments.

Our Contribution. In this paper, we introduce PEACE, which extends the zkLogin line of research by addressing these challenges. Unlike naive OIDC reuse (e.g., “Sign in with Google”), which centralizes trust in a single identity provider, PEACE distributes trust across a threshold of multiple IdPs so that no single party can observe or impersonate users (Brands, 2007; Auth0, 2024). PEACE is the first blockchain authentication protocol to combine threshold-based OpenID Connect login with password-protected secret sharing (PPSS) (Bagherzandi et al., 2011; Dziem-

bowski et al., 2024), eliminating the need for long-term private key management while enabling secure account recovery. This design uniquely delivers Web-style single sign-on and recovery *and* on-chain non-custodial security—properties not jointly offered by prior schemes.

It eliminates the need for long-term cryptographic material such as salts, supports recovery, allows users to update their authentication configuration, and provides a rigorous security proof in the *Universal Composability* (UC) (Canetti, 2020) framework. Our work can be seen as a generalization of zkLogin: while zkLogin demonstrated the practicality of OIDC-based blockchain authentication, PEACE extends this paradigm to support multiple IdPs, recovery via PPSS, and UC-grade security. In more detail, our contributions are as follows.

- **Protocol Design.** We design PEACE, the first blockchain authentication protocol that combines threshold OIDC login with password-protected secret sharing (PPSS) (Bagherzandi et al., 2011), eliminating long-term cryptographic key management. Table 1 compares our protocol to previous OIDC-based blockchain authentication systems.
- **Formal Guarantees.** We provide a full UC proof that PEACE achieves *unforgeability*, *unlinkability*, and *liveness*, extending prior work that offered only game-based proofs. This UC-security guarantee is a significant advantage over previous blockchain login proposals: it ensures composable, adaptive security even if some components are compromised or protocols are run in parallel.
- **Implementation and Evaluation.** We present an efficient open-source implementation of PEACE compatible with existing IdPs and Ethereum-like blockchains. Our client generates proofs within 4s on commodity hardware and verifies them on-chain with negligible gas overhead. As part of our implementation, we create the first practical deployment of augmented PPSS (Dziembowski et al., 2024).

2 BACKGROUND

In this section, we provide background on the main cryptographic tools used in our paper.

Universal Composability. As highlighted above, we model the security of PEACE in the Universal Composability (UC) framework (Canetti, 2020). UC is the gold standard for cryptographic protocol analysis: it guarantees that security is preserved even when the protocol is composed with arbitrary others. In UC,

Table 1: Properties of OIDC-based blockchain authentication systems. *Unlinkability*, represents inability to correlate user’s OIDC and blockchain identities. ● indicates full unlinkability, ◐ indicates that some entities (e.g. zkLogin proof or salt servers) can deanonymize the user. ○ indicates public linkability. *IdP Compatible* represents using RSA signatures rather than other schemes (e.g. EDDSA) that, while easier for generating Non-Interactive Zero-Knowledge Proofs (NIZK), are not used by major IdPs. *Updatable* indicates whether the user can update their authentication information (e.g. the list of IdPs).

System	Unlinkable	No SPoF	IdP Compatible	Updatable	ID Recovery	Security
OIDC Recovery (Heilman et al., 2023; Palladino, 2019)	○	✗	✓	✓	✓	informal
Addr abstraction (Park et al., 2023)	◐	✗	✗	✗	✓	game-based
zkLogin (Baldimtsi et al., 2024)	◐	✗	✓	✗	✗	game-based
PEACE	●	✓	✓	✓	✓	UC

security is defined by comparing a real protocol execution with an ideal functionality that captures the desired behavior. We refer readers to standard surveys (e.g., (Canetti, 2004; Bitansky et al., 2017)) for a full treatment.

OpenID Connect. OpenID Connect (OIDC) (Sakimura et al., 2014) is a widely deployed Web authentication protocol. An IdP authenticates a user and issues a signed JWT containing user attributes. OIDC supports a *nonce* parameter, intended to prevent replay attacks, that allows clients to embed arbitrary data into the signed token. This mechanism is the hook exploited by prior work (Heilman et al., 2023; Park et al., 2023; Baldimtsi et al., 2024) and by our system.

Password-Protected Secret Sharing. As mentioned above, our solution uses password-protected secret sharing (PPSS) (Bagherzandi et al., 2011), which is a protocol that enables a user to reconstruct a high-entropy secret from a threshold of servers using only a short password. Even if some servers are compromised, they cannot recover the secret without the password. We use PPSS to protect against IdP compromise and to enable identity recovery. Full technical details of the scheme and our augmented implementation (Dziembowski et al., 2024) are deferred to the appendix.

Non-Interactive Zero-Knowledge Proofs. Non-Interactive Zero-Knowledge Proofs (NIZKs) (Blum et al., 1988) allow non-interactively proving knowledge of a secret without revealing it. They are a key tool in blockchain authentication, as they enable users to prove possession of valid JWTs and PPSS secrets without exposing them on-chain. We instantiate our system with a zk-SNARK (Bitansky et al., 2017), but any UC-secure NIZK suffices. We defer formal definitions and proof system details to the appendix.

3 OVERVIEW

We now provide an overview of PEACE before presenting the formal model and protocol details. The goal of PEACE is to allow users to authenticate blockchain transactions using familiar OIDC logins, without having to manage long-term cryptographic material. At a high level, the system combines threshold OIDC authentication with password-protected secret sharing (PPSS) and zero-knowledge proofs to achieve non-custodial, privacy-preserving, and UC-secure authentication.

Initialization. The user registers with a set of n_{IdP} OIDC providers and chooses a threshold t_{IdP} required to authorize transactions. To mitigate the risk of IdP compromise or collusion, PEACE augments OIDC with PPSS (Bagherzandi et al., 2011). The user memorizes a password and uses it to reconstruct a high-entropy secret from at least t_{PPSS} PPSS servers. This secret need not be stored by the user and can be updated at any time, providing both recoverability and flexibility.

Certificate Generation. When submitting a transaction, the user generates an ephemeral key pair (epk, esk) and requests threshold-signed JWTs from t_{IdP} IdPs that include epk in the *nonce* field. Simultaneously, the user derives the PPSS secret using their password. Rather than exposing JWTs or secrets on-chain, the client locally proves in zero-knowledge that it possesses valid IdP tokens and the PPSS secret authorizing epk. The resulting certificate binds epk to the user’s OIDC and PPSS credentials without revealing them.

Transaction Submission. The user signs the transaction with esk and submits it along with epk and the zero-knowledge certificate. Once epk is authorized, subsequent transactions can be signed directly until

the user rotates to a fresh ephemeral key. Multiple ephemeral keys can be active in parallel, improving usability across devices.

Security Intuition. Due to the integration with PPSS servers, our construction eliminates the need to store salts or long-term keys (C1), our smart contract design allows an authenticated user updating IdPs and PPSS servers without changing the blockchain address (C2), and enables recovery via PPSS (C3), we ensure that client-side generation of ZK proofs is efficient (C4), and at the step of certificate generation we require communication with threshold of IdP servers, and thus remove reliance on a single IdP (C5). Finally, our signing solution is proven secure in the Universal Composability framework (C6).

3.1 PEACE Protocol

Due to space constraints, sections on the full protocol description and security analysis are deferred to the full version of the paper, available as an eprint on arXiv. We refer the reader to the full version for details on the protocol and its security analysis.

4 DESIGN AND IMPLEMENTATION CHOICES

We justify PEACE design and implementation choices that enabled the creation of a practical system that represents the protocol described in the Section 3.1.

Account Updates and Identity Recovery. In PEACE, an adversary requires both compromising user's password-protected secret sharing (PPSS) password and breaching t_{oidc} IdPs making our scheme more secure. Beyond day-to-day authentication, PEACE also supports changing an account's authentication configuration over time (e.g. rotating the PPSS password, updating the set of IdPs). Since $auth_{str}$ commits to both the IdP-side configuration and the PPSS-derived secret, any configuration change naturally corresponds to updating $auth_{str}$ (and its on-chain commitment) to a newly computed value.

Authenticated Updates (Normal Case). When the user still possesses the current authentication material, updates can be authorized under the *current* configuration. Concretely, the user authenticates as usual to obtain threshold JWTs from t_{oidc} IdPs and produces a Zero-Knowledge Proofs (ZKP) attesting to correct authentication under the currently active $auth_{str}$. The resulting transaction can then carry a new commitment (corresponding to the desired new configura-

tion) and update the account state. Intuitively, this is "just another authorized operation" under the same interface used for ordinary transactions, except that its effect is to rotate the committed authentication state.

Recovery Updates (Lost PPSS Password). Integration with OIDC additionally enables a practical recovery path when the user loses the PPSS password. In this case, the user cannot authenticate under the current $auth_{str}$, but can initiate an update that is authorized by t_{oidc} IdPs and the user. At a high level, the user chooses a fresh PPSS password, recomputes $auth_{str}$ accordingly, and obtains JWT tokens from at least t_{oidc} IdPs where the authentication transcript is bound to the *new* $auth_{str}$. The user then submits a transaction that proves possession of these threshold JWTs and correctness of the new commitment, causing the blockchain account to accept the updated $auth_{str}$.

To mitigate the risk of malicious or compromised IdPs triggering unauthorized recovery, we propose delayed activation for updates that are not authorized under the currently active configuration. The account accepts the new $auth_{str}$ only after a delay (e.g. one week). During this window, the current account holder can veto the pending update by submitting a dedicated ZKP proving knowledge of the PPSS secret under the currently active configuration, which cancels the pending change. If the user truly lost the PPSS password, the update will eventually activate, while if an attacker attempts to abuse IdP-side recovery, the legitimate user can prevent takeover as long as they monitor activity on their account during the delay period.

Decoupling the Account Address from Authentication Information. Previous works (Baldimtsi et al., 2024; Park et al., 2023) derive blockchain account addresses directly from information encoded in the OIDC JWT. This is partially to enable discoverability - the ability to intentionally make an address findable or publicly attributable to a real-world identity. In PEACE, we deliberately make the authentication string (i.e., $auth_{str}$) independent from the user address. This enables seamless migration of authentication information (e.g. rotating the PPSS password, updating the IdP set/threshold) while keeping a stable address. At the same time, discoverability remains possible when the user opts in. For example, a user may disclose their OIDC identifiers (or selected attributes) to enable third parties to verify ownership. Directly disclosing the current PPSS password or PPSS output is undesirable, as it lowers authentication security; however, a user may disclose an *old* PPSS output (or deliberately rotate to a chosen value) to prove account control without weaken-

ing the current configuration. Finally, users can generate a dedicated ZKP that proves ownership of an address (equivalently, knowledge of the corresponding auth_{str}) or proves possession of an OIDC identifier satisfying a predicate (e.g. having an email from a given domain such as `nytimes.com`), without revealing the identifier itself. We consider these opt-in mechanisms more secure and more flexible than permanently linking the address to OIDC identities.

Combining OIDC with PPSS. PEACE is built on top of both OIDC and PPSS. One can build an authentication system based on only one of those components. In the PPSS-only-based protocols (e.g. aptSIG (Dziembowski et al., 2024)), the security relies only on the password strength chosen by the user. If an adversary successfully guesses the password, they gain the ability to generate signatures on arbitrary messages. Further, aptSIG does not provide any account recovery option if the password is lost. OIDC-only-based protocols offer identity recover, as IdPs already implement this feature (e.g. based on identity documents). However, malicious IdPs can take control of the user account.

UC-Compatible NIZKs. NIZKs are not Universal Composability (UC)-compliant by default. To make sure that our implementation inherits the UC security proven for the protocol, PEACE Succinct Non-Interactive Argument of Knowledge (SNARK)s must be compiled into a universally-composable succinct non-interactive argument of knowledge (UC-SNARK) (Kosba et al., 2015; Ganesh et al., 2023). This is done by preserving a random public key pk in a common reference string, and then attaching to the UC-SNARK public-key encryption of a witness (further denoted ct) to the statement in relation $\mathcal{R}$. We then prove with the SNARK not only that the witness and the statement are in relation but that the ciphertext correctly encrypts the witness as well. This construction assures straight-line extractability of the witness.

NIZK-Friendly Key Management. Verifying certificates submitted by the users requires access to the recent public key of each IdP selected by the users. Blockchain validators must thus agree on the keys to use. At the same time, storing a large number of keys on-chain would be inefficient.

PEACE submits a commitment to all the IdPs public keys in the form of a Merkle tree root without storing on-chain the actual keys. This limits the amount of data submitted to the blockchain. The Merkle tree root is shared across all PEACE users. Such a solution enables sharing the key update costs across all the users using a specific IdP. A production deployment should use an oracle that confirms this information at

IdP-exposed endpoints. Additionally, this form of key storage also simplifies the design of our ZKP circuit and enables a significant speedup of proof generation.

PPSS Reconstruction. In the basic version of the PPSS protocol (Dziembowski et al., 2024), the user contracts t_{PPSS} out of n_{PPSS} servers and attempts to reconstruct the secret. However, if one of the servers returns an incorrect value, the user needs to select a new disjoint set of t_{PPSS} servers and try again. To speed up the process, in PEACE, the client contacts all the n_{PPSS} servers in parallel and locally reconstructs the secret by iterating through all the possible sets of t_{PPSS} servers. This method is guaranteed to succeed as long as there are at least t_{PPSS} non-byzantine PPSS servers.

OIDC Compatibility. PEACE is compatible with the standard OIDC protocol flow. All the major IdPs that implement OIDC at the same time implement the server routines in the Initialization and Get Certificate phases. The only difference is that in OIDC the JWT *nonce* is randomly chosen by the user, while in PEACE the nonce binds the username to the auth_{str} . However, this change is opaque for the IdP and does not require any modification at the server side. We also use RS256 (RSA using SHA-256) to remain compatible with all the mainstream IdPs.

5 IMPLEMENTATION

We provide PEACE implementation in a public GitHub repository¹.

Identity Provider. Existing OIDC implementations have been evaluated before (Mainka et al., 2017; Blazquez et al., 2015), we thus simulate token issuance by writing an HTTP server and issuing RSA256-signed JWT tokens with `golang-jwt` library (golang jwt, 2024).

Password-Protected Secret Sharing. We are the first to implement password-protected secret sharing (Dziembowski et al., 2024). Each server is implemented in Golang as an HTTP server that stores user values (ω in (Dziembowski et al., 2024)) in LevelDB (Google, 2024) indexed by custom user identifiers.

Blockchain Verifier. We implement Verify Transaction functionality as an Ethereum smart contract. This enables the blockchain to verify PEACE transactions. Ethereum had two types of accounts *Externally Owned Accounts (EOA)* and *Contract Accounts (CA)*. EOAs are user accounts used

¹<https://link-to-github-repo---anonymized-for-review--->

for sending Ether or invoking smart contracts. The address of an EOA is derived from the owner’s public key, and every transaction must be signed by the corresponding private key. CA are smart contracts, which have associated code that can be executed based on predefined logic. With the introduction of Ethereum Account Abstraction (EAA) (Ethereum, 2024), EOAs can now initiate transactions on their own.

Ethereum Account Abstraction (EAA). EAA allows smart contracts to (i) initiate transactions and (ii) use any custom logic to validate them. It enables smart contract wallets that function like EOAs but with flexible control mechanisms. EIP-4337 implements EAA as a smart contract without requiring changes to the core Ethereum consensus layer. Its key mechanism is an alternative mempool for `UserOperations` which encapsulate the transaction data and authentication material. EIP-4337 uses bundlers to gather these `UserOperations` from the alternative mempool, validate them, and batch them into standard Ethereum transactions. This validation involves executing the smart contract logic associated with each operation (e.g. authentication). Once validated, the bundler includes the batch in an Ethereum block, turning the operations into executable Ethereum transactions.

Verifier. We implement PEACE EAA verifier using ZeroDev (ZeroDev, 2024). We replace `validateUserOp` function, responsible for authentication, with the Solidity code generated by the client component that implements `Verify Transaction`. ZeroDev provides bundlers and infrastructure to submit `UserOperations`. We store `authstr` and the most recently authenticated ephemeral public key `epk` as a contract variable that can be updated by the user, enabling updating the PPSS password or the IdP set. This allows using the same private key for multiple transactions without generating a new zero-knowledge proof every time.

User Component. We build the PEACE ZKP generation engine on top of circuits from the zkEmail (zkEmail, 2024), which offers standard JWT verification using Circom (Bellés-Muñoz et al., 2022). We implement a code generator that creates circuit code for the specified total number of IdPs n_{IdP} and the required signatures threshold t_{IdP} .

We provide implementations using Rapidsnark (Rapidsnark, 2024) and SnarkJS (SnarkJS, 2024). Rapidsnark is a zkSNARK proof generation written in C++ and Intel/ARM assembly, while SnarkJS is JavaScript and Pure WebAssembly implementation. The former offers higher performance but

requires software installation by the user. The latter enables compilation to WebAssembly and on-the-fly execution in all the major Web browsers but suffers from lower performance.

6 EVALUATION

We implement the PEACE and evaluate its performance. We make the following claims: 1. PEACE incurs low blockchain monetary costs for both deployment and usage operations; 2. PEACE user-side operations can be completed within a reasonable time on mid-range devices. We do not compare against zkLogin (Baldimtsi et al., 2024), as their implementation is not fully open-source.

Setup. We deploy the client on a midrange laptop (12th Gen. Core i5 CPU). We host all the servers on dedicated AWS `t2.xlarge` instances located in the region closest to the client. To calculate blockchain USD prices, we use the gas cost of 0.11 GWEI and ETH price of \$2,948 from January 25th, 2026.

Blockchain Cost. We report the cost of deploying and running `validateUserOp` rather than the cost of the entire ZeroDev pipeline. This is to remain compatible with other EIP-4337 implementations and isolate ZeroDev costs that change frequently due to intense development.

Setting a PEACE account requires deploying two smart contracts consuming 542k (\$0.33) and 392k (\$0.42) gas respectively. This results in a total deployment cost of 935k (\$0.57). This cost must be paid only once during the account lifetime, we thus consider it affordable for the end-users. Modifying a Merkle root holding IdP public keys requires spending 68k gas (\$0.42). This is done only when a new IdP is added to the system or one of the existing ones rotates its keys. This cost can be shared across all the PEACE users or covered by the system operator.

The on-chain proof verification requires 235k gas, translating into \$0.14. Importantly, this cost is constant and is not influenced by the number of IdPs used by the user. Users need to authorize a new ephemeral key only when they add a new device or want to rotate their old key (e.g. for security reasons). We thus expect the increased per-transaction cost to be negligible, especially for accounts sending large volumes of transactions.

User-Side Cost. Figure 1 presents PEACE performance for the client-side operations. We compare Rapidsnark (solid line) and SnarkJS (dashed line) implementations and regular (yellow line) and UC-compatible (blue line) versions of NIZK. Increasing

the total number of IdPs n_{IdP} has a negligible impact on the proof generation (*i.e.* $< 1\%$). For clarity, we thus set $n_{IdP} = 4$ and investigate the impact of increasing the threshold of required IdPs t_{IdP} up to 4. Higher IdP numbers are rarely used together in real-world OIDC deployments (Dimova et al., 2023).

For all the implementations, we observe an increase in the proof generation time, and memory usage linear to t_{IdP} . For the generation time, we report the mean value and the standard deviation from 20 runs. The Rapidsnark version enables proof generation within a few seconds for both scenarios. We conclude that such proof generation times are acceptable for the end-users. Authorizing a new ephemeral public key is a procedure that is performed sporadically. Furthermore, the values are comparable to the average transaction confirmation for the Ethereum blockchain (*i.e.* 12s to 14s).

The SnarkJS version requires up to 2 minutes to generate the witness and proof. This is an order of magnitude slower than the Rapidsnark implementation. However, we suspect that many users might be willing to pay this cost to avoid the installation of additional software components. Furthermore, the execution time will be lower on higher-end laptops. The UC-compliant version of NIZK increases the number of constraints, computation time, and memory usage by $\approx 10\%$.

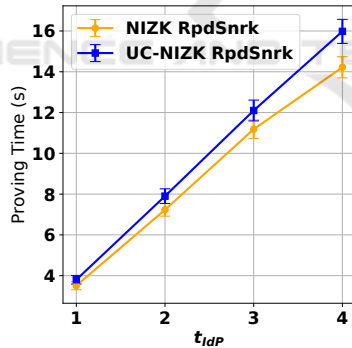


Figure 1: Comparison of different ways of generating user-side NIZK.

7 RELATED WORKS

Previous work has proposed embedding arbitrary data into the OpenID Connect (OIDC) nonce (Heilman et al., 2023; Palladino, 2019), but these approaches lack a zero-knowledge component, allowing linkage between blockchain activity and user IdP logins. As explained in Section 1, the closest related system is *zkLogin* (Baldimtsi et al., 2024). While this important work implements zero-knowledge proofs (ZKPs),

it imposes constraints: users must manage cryptographic material (e.g., the so-called *salt*), are restricted in modifying their initial set of IdPs, and rely on custom support from the underlying blockchain. We compared PEACE with *zkLogin* in detail in the introduction.

ZK Address Abstraction (Park et al., 2023) employs general-purpose ZKPs over JWTs for transaction authentication. However, it tightly couples ZKPs with transaction generation, requiring the user to generate a new proof for each transaction, reducing its practicality. *CanDID* (Maram et al., 2021) offers another method for migrating web credentials to blockchains without altering existing identity providers. However, CanDID introduces an MPC committee, adding a layer of complexity. While CanDID supports various credential types (e.g., SAML, username-password), PEACE is restricted to signatures, such as those in OIDC-based systems.

ACKNOWLEDGMENTS

Stefan Dziembowski, Paweł Kędzior: this work has been partially funded by the European Research Council under the European Union’s Horizon 2020 innovation program (grant PROCONTRA-885666).

REFERENCES

- Auth0 (2024). Introduction to openid connect.
- Bagherzandi, A., Jarecki, S., Saxena, N., and Lu, Y. (2011). Password-protected secret sharing.
- Baldimtsi, F., Chalkias, K. K., Ji, Y., Lindström, J., Maram, D., Riva, B., Roy, A., Sedaghat, M., and Wang, J. (2024). *zklogin*: Privacy-preserving blockchain authentication with existing credentials.
- Bellés-Muñoz, M., Isabel, M., Muñoz-Tapia, J. L., Rubio, A., and Baylina, J. (2022). Circom: A circuit description language for building zero-knowledge applications.
- Bitansky, N., Canetti, R., Chiesa, A., Goldwasser, S., Lin, H., Rubinfeld, A., and Tromer, E. (2017). The hunting of the SNARK.
- Blazquez, A., Tsiatsis, V., and Vandikas, K. (2015). Performance evaluation of openid connect for an iot information marketplace.
- Blum, M., Feldman, P., and Micali, S. (1988). Non-interactive zero-knowledge and its applications.
- Brands, S. (2007). The problem (s) with openid.
- Canetti, R. (2004). Universally composable signature, certification, and authentication.
- Canetti, R. (2020). Universally composable security.
- CoinGecko (2025). Global Cryptocurrency Market Cap Charts.

- Dimova, Y., Van Goethem, T., and Joosen, W. (2023). Everybody's looking for something: A large-scale evaluation on the privacy of oAuth authentication on the web.
- Dziembowski, S., Jarecki, S., Kedzior, P., Krawczyk, H., Ngo, C. N., and Xu, J. (2024). Password-protected threshold signatures.
- Ethereum (2024). Erc-4337: Account abstraction using alt mempool.
- Ganesh, C., Kondi, Y., Orlandi, C., Pancholi, M., Takahashi, A., and Tschudi, D. (2023). Witness-succinct universally-composable snarks.
- golang jwt (2024). A go implementation of json web tokens.
- Google (2024). Leveldb.
- Heilman, E., Mugnier, L., Filippidis, A., Goldberg, S., Lipman, S., Marcus, Y., Milano, M., Premkumar, S., Unrein, C., and Merfeld, J. (2023). OpenPubkey: Augmenting OpenID connect with user held signing keys.
- Kosba, A., Zhao, Z., Miller, A., Qian, Y., Chan, H., Papamanthou, C., Pass, R., abhi shelat, and Shi, E. (2015). C0c0: A framework for building composable zero-knowledge proofs.
- Labs, M. (2024). Sui.
- Li, W. and Mitchell, C. J. (2020). User access privacy in oAuth 2.0 and openid connect.
- Mainka, C., Mladenov, V., Schwenk, J., and Wich, T. (2017). Sok: single sign-on security—an evaluation of openid connect.
- Maram, D., Malvai, H., Zhang, F., Jean-Louis, N., Frolov, A., Kell, T., Lobban, T., Moy, C., Juels, A., and Miller, A. (2021). Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability.
- Markets Insider (2021). There's roughly \$140 billion of inaccessible bitcoin right now — or 20% of the world's limited supply. Here's what could happen to it.
- Palladino, S. (2019). Sign in with google to your identity contract.
- Park, S., Lee, J. H., Lee, S., Chun, J. H., Cho, H., Kim, M., Cho, H. K., and Moon, S.-M. (2023). Beyond the blockchain address: Zero-knowledge address abstraction.
- Rapidsnark (2024). RapidSnark: fast zksnark prover written in c++.
- Sakimura, N., Bradley, J., Jones, M., De Medeiros, B., and Mortimore, C. (2014). Openid connect core 1.0 incorporating errata set 1.
- Sharif, A., Carbone, R., Sciarretta, G., and Ranise, S. (2022). Best current practices for oAuth/oidc native apps: A study of their adoption in popular providers and top-ranked android clients.
- SnarkJS (2024). SnarkJS: Javascript and pure web assembly implementation of zksnark and plonk schemes.
- Times, T. N. Y. (2021). Lost passwords lock millionaires out of their bitcoin fortunes.
- YCharts (2025). Bitcoin Market Cap.
- ZeroDev (2024). Zerodev.
- Zhang, Z., Król, M., Sonnino, A., Zhang, L., and Rivière, E. (2021). El passo: efficient and lightweight privacy-preserving single sign on.
- zkEmail (2024). jwt-tx-builder.