Holonym: Private Proofs on Identity for Blockchains and Beyond

Nanak Nihal Khalsa
nanak@holonym.id

Caleb Tuttle
caleb@holonym.id

Kushal Kahar
kushal@holonym.id

Shady El Damaty
shady@opscientia.com

November 19, 2022

Motivation

Holonym addresses the problem of private identity on public blockchains. Sensitive data is often needed by web applications but cannot be provided to decentralized applications (dApps) whose underlying databases are publicly visible ledgers. Trusted custodians are not an ideal solution because they are privvy to a user's historical activity and proxy measures that can be used to determine identity or learn their wallet address. Holonym uses ZK-SNARKS for private identity on blockchains. This is done to enable novel uses such as private KYC-analogs, strong Sybil resistance, on-chain democracies, and noncustodial wallet recovery.

Introduction

The Web Needs Default Privacy Settings Internet users today have a choice between trusting either centralized servers or decentralized protocols with their data. Centralized servers are private but these custodians collect data that can be used to identify and track nonconsenting users. Centralized servers also have a track record of security breaches, and this trend is expected to continue as new exploits are found. Decentralized systems are more secure to tampering and possess remarkable transparency for how data is stored and used. However, the benefit of transparency comes at the cost of user privacy. Holonym was developed to address the problem of private identity on public blockchains, allowing users to reap the benefits of "trust-less" decentralized web architecture while retaining control over their sensitive data. Holonym provides a framework for default privacy settings on the internet by placing user consent, transparency, and decentralization as first principles. Holonym uses Zero Knowledge Succinct Noninteractive Arguments of Knowledge (ZK-SNARKs) for private identity on public blockchains. This synergy of privacy and public, composable ecosystems allows novel uses of portable cloaked identities: e.g., private credential proofs, strong Sybil resistance, on-chain democracies, and noncustodial wallet recovery.

Zero Knowledge Proofs Zero knowledge proofs (ZKPs) are innovations in cryptography from the 1980s which have become practical only recently [13]. ZKPs allow the proving of a statement without revealing any information other than the statement. ZKPs also enable the compression of an arbitrary computations into a small, often constant-sized proof of computational integrity. These properties of privacy and succinctness have led to uses in privacy and scalability. Holonym utilizes the privacy-preserving features of ZKPs, particularly ZK-SNARKs, in order to prove facts of an identity, such as being a US resident or a unique person, without revealing anything else about ones identity. This allows identity verification in contexts where you cannot trust the verifier to keep your data secure.

Definitions

Holonym Holonym is defined as a whole which is comprised of parts. We refer to a holistic private identity as a Holo, which is comprised of individual identifiers called nyms. The idea of a Holo to privately link a set of nyms was inspired by Nick Szabo, the first to introduce smart contracts and virtual personae in the late 90s [10]. We define a Holo as an individual's set of commitments that reference a variety of nyms, such as a name, social accounts, or social security number. Nyms may reveal various degrees of identifying information about a user. These are bundled together with the user's consent to produce proofs that reveal objective facts about a user, such as residency, whether they are an adult, or have voted in a digital election. Together, a full set of commitments produced from valid credentials provide a holistic picture of an individual that can be used to interact with web applications utilizing privacy as a default setting.

Rigorous Sybil Resistance Sybil resistance is a general term frequently applied to any method of preventing abuse of digital systems by automated programs (e.g., bots). The key to achieving Sybil resistance is the ability to distinguish activity that belong to authentic users from automated programs impersonating real users. There are many methods for establishing Sybil resistance, each with their strengths and weaknesses. For example, CAPTCHAs are useful for preventing abusive behavior on web platforms but they are useless for protecting blockchain consensus mechanisms from adversarial bot attacks. We define a strong form of Sybil resistance in relation to a reward function, $R_A(n)$, and cost function, $C_A(n)$, of an action, A, done $n \in \mathbb{Z}^*$ times (note: $\mathbb{Z}^*$ denotes non-negative integers). An action A has Sybil resistance if and only if:

$$sgn(R_A(n) - C_A(n)) = \begin{cases} 1, & \text{if } n = 1 \\ -1, & \text{otherwise} \end{cases}$$

I.e., an action is Sybil resistant if and only if its reward outweighs its cost for the first time only; doing the action any higher number of times, the cumulative cost will outweigh the cumulative reward. Bot attacks are characterized by a high number of repeated requests to a web service or application. These repeated requests are often intended to artificially inflate content engagement with fake votes, likes, comments, or other common types of web activity. Here we define adequate Sybil resistance as a high cost to satisfying any requests beyond the first. Using this definition as a benchmark, it becomes clear that many types of bot prevention mechanisms which advertise Sybil resistance do not actually meet this definition because they are no more difficult to do multiple times; they rather simply add user friction to an action, a friction that does not increase with n. A common example are mechanisms that require users to link multiple web accounts. The thinking behind this strategy is that bots are less likely to create multiple accounts to use the platform. However, this is also gameable with composable Sybil attacks by bots on each of the web accounts required to be linked. The cost for creating the first account does not scale with repetition. A real-world example of a highly resistant mechanism, is one that is difficult to repeat more than once such as a passport registration. Currently, there are no existing digital solutions that approach the level of Sybil resistance afforded by traditional forms of identity verification. One could argue that this definition is exclusionary, which is correct. However, this definition may be found helpful to delineate a robust level of security against bot attacks.

Use-Cases

Private Proofs on Identity Verification of identity and credential validation is often performed using more information than is required to satisfy the needs of the verifier (PII). For example, verifiers may only need to know whether an individual has a legitimate proof of residency, is above a certain age, has a registered legal entity, etc. Current verification flows utilize a credential rich with sensitive information that reveal more information than is required, such as home address, personal finances, or social security number. Sensitive information is often leaked in data breaches, and cannot even legally be collected by certain entities, despite its utility to the entity. Therefore, privacy-preserving proofs about identity can not only prevent PII leakage and bolster user safety, but also enable use-cases of PII that does not reveal the actual PII.

Sybil resistance Voting protocols (such as blockchain consensus, DAO voting, & quadratic funding) require Sybil resistance [12]. Web3 Sybil attacks can only be resisted currently – they cannot be completely prevented [12]. Holonym changes the mechanisms of Sybil resistance to make Sybil attacks nearly impossible: it enables the same Sybil resistance mechanisms that the governments use for national elections and for multi-trillion-dollar social security dispersal. This would be a substantial change to web3 Sybil resistance, which can often be attacked for as only $10 [7]. Furthermore, current Sybil-resistance methods require significant effort on the user's part, which can be mitigated by using automated zero-knowledge proofs instead of social attestations and verification parties that are currently employed. With easier and more rigorous Sybil resistance, on-chain democracies, mechanisms for universal basic income, and a wide array of secure "trust-less" business transactions become possible. The uses cases can go beyond on-chain proofs to reduce spam on email, games, and social networks.

Holonym produces proofs on data that need verification, without revealing sensitive information. This may include proofs of age or proofs of residency, such as "User 0x1234 is an adult resident of the US and has voted in three elections". Proof of residency is accomplished by a novel, simple cryptographic accumulator scheme that represents each country as a prime number for simple set membership in country accumulators.

Anonymous Reputation and History Nullifier schemes enable recording actions across anonymous users. An immutable ledger like a blockchain may store commitments to actions that can be used to prove those actions were or were not performed. For example, without revealing ones identity, one can prove that one has returned 3 loans or has not yet submitted $3,000 in transactions.

Such a scheme can be useful in cases such as the Bank Secrecy Act[4, 5] (BSA). The BSA does not require collecting customer identity information for transactions under a certain threshold. However, one person can easily break a large transaction into small pieces to avoid requirements for "Know-Your-Customer" (KYC). Governments keen to curtail money laundering and financing of illicit activities have made this practice illegal but practical enforcement of this law remains to be demonstrated. ZK identity can prevent such structured transactions by representing unspent transaction limits in a UTXO-like model within two Merkle trees, one sparse. The Merkle tree architecture is described in further detail in the structured transactions architecture section. This design pattern may enable non-web3 financial institutions to detect structured transactions without in-depth customer identification, combating crime which is currently expensive and impossible to prevent completely.

Non-custodial Wallet Recovery A key feature of the decentralized web is the self-custody of cryptographic keys by individual users. Instead of centralized web services that host the keys used to identify individuals and authorize transactions associated with identities, users keep their keys local on their own hardware. Cryptoassets are often permanently lost due to lost private keys. Recovery solutions utilizing technology such Apple Keychain or AWS servers have been proposed but at the expense of nominating a centralized authority to safe-guard user identities, effectively making them an owner of stored user data. Social recovery has been proposed [14] as a more decentralized solution, however this still requires trust and delays lasting a week for a single transaction [14]. Holonym enables "antisocial recovery", where keys can be recovered without trusting friends & without delaying regular transactions with a long wait period. It accomplishes this via proofs that one is the rightful owner of a wallet, which don't require a private key but rather linked identities, or nyms. A smart contract wallet can use this as a recovery mechanism, or a decentralized threshold encryption network such as the Lit protocol [9] can store the private key, recovering it after a private proof of identity. This form of wallet recovery uses an architecture entirely different than the Merkle tree-based mechanism used elsewhere within Holonym, and more information on this architecture is in the wallet recovery architecture section.

Existing Approaches to Identity

Multiple approaches for identity and trust on the decentralized web, or web3, exist with unique capabilities and shortcomings. Some with radically different approaches include:

- Verifiable Credentials, e.g. Verite and Disco
- Social attestation networks, e.g. Proof of Humanity & BrightID
- On-chain resume / badge services, e.g. Rabbit Hole & Project Galaxy
- Centralized federated identity, e.g. Magic Link & traditional SSO
- Enterprise DID and SSI services, e.g. Evernym, BloomID, & Mattr
- Traditional KYC services, e.g. Blockpass & Stripe

Verifiable Credentials Verifiable Credentials (VCs) are a digital schema for user-owned identities. A standard for VCs has emerged through the

World Wide Web Consortium (W3C), which seeks to bring interoperability to self-sovereign identity systems. [8]. VCs are given in JSON format which is practically incompatible with R1CS used in ZK-SNARKs and with EVM blockchains; the variable length data and unserialized format of raw JSON do not lend themselves to such environments. Nevertheless, VCs have been a significant help in bringing interoperability to non-ZK, off-chain, self-sovereign identity protocols.

Social attestation networks such as Proof of Humanity and BrightID provide a handful benefits such as a Universal Basic Income token and some level of Sybil resistance, albeit nonconforming to the definition we provided of robust Sybil resistance. This level of Sybil resistance is suitable for lower-reward use cases, but game-theoretic concerns prevent higher-reward use cases (an attack can cost \$10 [7]).

On-chain resume / badge services provide pseudonymous credentials. They effectively show accomplishments which usually to skill- and experience-related uses, rather than security- and identity-related uses. There is work by Sismo to add anonymity to these badges, enabling security- and identity use.

Centralized Federated Identity achieves a unified identity across apps, but it's owned by identity providers instead of users. Google SSOs are centralized. These have been well-adopted due to making verification for users and developers much easier, by routing requests efficiently through a centralized provider. They often involve JWTs, which Holonym verifies on-chain with ZK circuits to enable proof of Web2 accounts.

Enterprise DID and SSI services such as Evernym, BloomID, & Mattr provide some benefits of DIDs for organizations. They are often focused on credential storage and verification, reducing expenses and increasing efficiency of organizations through user-owned identities.

Traditional KYC services provide deep checks into not just identity but also legal status. On-chain KYC is a more significant privacy risk than Web2 KYC; after verification in Web2, a user's data on one website is no longer private, yet after verification in web3, a user's activity everywhere on public blockchains is no longer private. Therefore, we introduce steps so that no identity verifier can learn a user's wallet address.

Holonym's Architecture

1 Identity Proofs & Sybil Resistance

The primary mechanism of Holonym's ID verification that enables Sybil resistance is an architecture that could be described as "ZCash for IDs". KYC and Sybil resistance require relatively trivial engineering solutions if you compromise privacy, by having some centralized registry that maps government IDs to wallet addresses. To add privacy on top of this, a global Merkle tree is created with commitments to everyone's credentials. Proofs of properties of leaves serve as proofs of facts about identities. Such proofs use a nullifier paradigm similar to ZCash [15] and Tornado Cash [11], adapted for IDs instead of money.

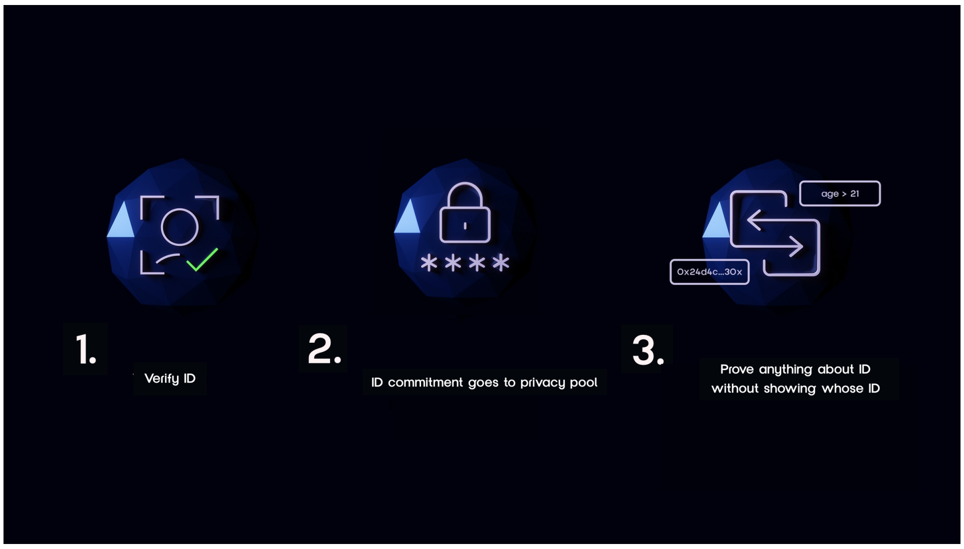
age > 21
0x24d4c...30x
1.
Verify ID
2.
ID commitment goes to privacy pool
3.
Prove anything about ID without showing whose ID

Figure 1: The steps of identity verification and anonymization. The anonymity set of Merkle leaves is referred to in this graphic as the privacy pool. First, identity verification is done by an issuer, who then gives a signed credential to the user. Next, to prevent the issuer from learning anything about the user beyond the fact they exist and just registered, the user makes a commitment of their signed credentials, which Holonym then submits to the Merkle tree. Finally, the user is free to prove any statement about their Merkle tree element.

ZK-friendly Credential Format Credentials must be properly formatted to enable decentralized, private attestation and proving. JSON formatting such as in W3C DIDs is (practically) incompatible with R1CS which requires fixed-length inputs. For optimal performance with common implementations of the Poseidon hash, the credential format we propose consists solely of field elements less than:

```
21888242871839275222246405745257275088548364400416034343698204186575808495617
```

There are two mandatory fields and any number of optional fields. The mandatory fields are: *issuer*, the credential issuer's Ethereum address; & *secret*, a value that can be used as a nullifier and pepper. One example of a credential format is:

index	name	description
0	issuer	Ethereum address of the credential issuer
1	country	prime representation of country
2	subdivision	numeric representation of state/province
3	birthdate	birthdate, as days since 1900
4	iat	credential issuance date, as days since 1900
5	secret	secret that acts as both a nullifier and pepper

Table 1: The credential format used in the Holonym beta version. This is one possible format of credentials, containing both mandatory fields of issuer and secret.

Credentials are self-custodied in a browser extension or app for privacy. A hash of the credentials (with a pepper) is put on-chain as a commitment, inserted within a global Merkle tree of credential commitments.

Checking and modifying commitments before submission Credential commitments must be determined valid before being added to the global Merkle tree. The issuer is an entity that signs valid credentials. Valid credentials can come from any identity authority or attestation service. Holonym prevents the issuer from learning anything about the user beyond the received credential. To accomplish this, the commitment has a secret that the issuer does not know. The checking and modifying of the commitment is an 8-step process:

1. Issuer returns verified credentials, generates a commitment (the hash of the credentials), and signs the commitment. For Sybil resistance, issuer can also store the credentials to check that the same user doesn't try to register twice.

2. User generates a random value that the issuer doesn't know and replaces secret with it

3. User commits to the credentials with the new secret

4. User generates ZK-SNARK, arguing that the new commitment and old commitment both come from the same preimage but with a different secret. This ZK-SNARK will have the issuer address (a part of the preimage) as a public input.

5. Holonym smart contract verifies ZK-SNARK

6. Holonym smart contract checks old commitment's signature against the issuer address provided as public input in the ZK-SNARK, to assert credentials came from the provided issuer

7. Holonym smart contract checks that old commitment hasn't been used before, to prevent double-spend attacks where an attacker can make multiple new commitments from one signed commitment. This is necessary (but not sufficient) to prevent Sybil attacks.

8. Holonym smart contract adds new commitment (which is now known to be valid) to the Merkle tree.

Proofs of attributes Once the credential commits are added to the on-chain Merkle tree, the user can prove arbitrary facts about them from any wallet address, as long as they know the secret. This way, users can spin up ephemeral addresses and prove identity from them, **so they can act from many verified addresses with no linkable trace between them**. This abstracts personhood away from an address, breaking the continuity of ones actions. Ones actions are instead linked by ones secret which nobody else knows. A user can reveal the continuity of their actions only if they know their secret, if they act from different wallet addresses. However, this is optional; nothing prevents a user from doing all actions from one wallet address and this will likely be the predominant way users create proofs as it is currently the most convenient.

Proofs of attributes happen via ZK-SNARK that includes a set of credentials. The credential hash is shown to be included in the Merkle tree, and the individual credentials are shown to have certain properties such as "country == 2", where 2 represents USA.

Country accumulators Countries are represented as prime numbers for efficient whitelisting. To set an "allow-list" of countries, an acculumator is used as public input. The accumulator is the mathematical product of all allowed countries' code, represented as a prime number for each country. Then, "user is from a valid country" simply becomes *country % accumulator == 0.*

Nullifiers *Secrets* can act as nullifiers, indicating a user has spent a credential. This is useful to prevent Sybil and double-spend attacks. Since credential issuance step is done by a "boring" centralized entity which can log who has registered, it is trivial to prevent somebody from registering twice. Since this issuer can choose to only give each user one set of credentials with one secret, there is a sort of centralized Sybil resistance here preventing users from registering twice. This privacy level is not ideal, which is why the process detailed in the **Checking and modifying commitments before submission** section allows the user to change the secret and use the new secret for Sybil resistance. This new secret is unknown to the issuer, but required the issuer's permission to create. In summary, the centralized issuer can check it hasn't verified a user before allowing them to make a secret. But the new secret won't be known to the issuer.

The secret can then be used to create proofs such as unique personhood. For example, a hash of the secret can be published along with a proof the resulting digest is correct. Thus, if a user tries to do an action twice from different addresses, it will reveal they are using the same secret! This allows Sybil resistance by tracking which secret hashes are spent. This construction becomes powerful when working with "salted" secret hashes. Any action has a unique salt and to do the action, the user must publish the hash of their secret with the action's salt. This makes a particular action Sybil resistant.

Prevention of Tax Evasion, Money Laundering, Drug & Terrorism Financing, and Miscellaneous Crime Having one secret per person enables what was not possible before in the prevention of unstructured transactions, to our knowledge. Most mechanisms of fighting money laundering are invasive to privacy and are usually only employed for transactions above \$3000[4, 5]. Criminals often split large transactions into smaller transactions to bypass identity checks. However, using secrets as nullifiers can prevent such splitting while preserving user privacy.

Doing so can be accomplished with a version of the UTXO model. Instead of unspent transaction outputs, this model employs Merkle tree representing unspent remaining balances (URBs). When money is first spent, a hash of *(secret, date, new balance)* would be added to a sparse Merkle tree of spent remaining balances (SRBs). *Note that this is an incomplete outline of how such a system can be built on Holonym; a full description of it may necessitate another whitepaper and can be left to the reader's imagination (or implementation!).*

A sparse Merkle tree also could be used for a "deny-lists", which can prevent criminals or sanctioned entities from interacting with smart contracts based on their government IDs. However, this is a form of censorship and should be avoided unless strictly necessary.

2 Proving ownership of Web2 accounts

Please note that the features mentioned in this section are not production-ready at the time of writing and may change slightly in the process of becoming production-ready.

Holonym can also allow the proof of Web2 credentials. This happens via verification of JSON Web Tokens (JWTs) within a SNARK.

Holonym works by forwarding web tokens to smart contracts which verify their data is truly signed by the OpenID provider, e.g. Google for a gmail account. Single sign-on (SSO) services (login with Google et al. buttons) return JSON web tokens (JWTs) with RSA or HMAC signatures. RSA signatures used by Google, Facebook, ORCID, and others have the benefit of being verifiable by anyone. ECDSA signatures could also work and be far easier to implement on EVM chains but unfortunately, few servers sign JWTs with elliptic curve signatures.

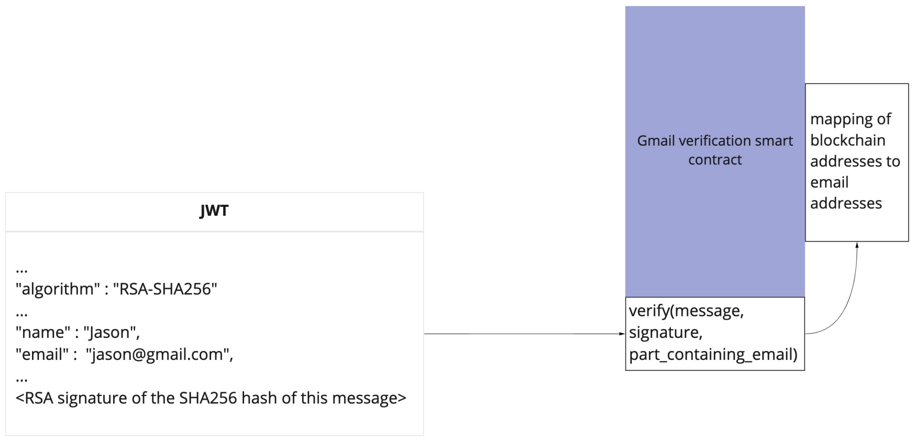
JWT
...
"algorithm" : "RSA-SHA256"
...
"name" : "Jason",
"email" : "jason@gmail.com",
...
<RSA signature of the SHA256 hash of this message>
Gmail verification smart contract
mapping of blockchain addresses to email addresses
verify(message, signature, part_containing_email)

Figure 2: JWT verification process. The smart contract verifies the JWT's signature and therefore verifies its contents. The smart contract retrieves the certain data from the JWT's proof.

Asymmetric RSA Signatures

RSA Signatures are verified by sending the JWTs to the Verifier smart contracts, which use the 0x05 modular exponentiation precompile to verify the signature. Note that the actual JWT is not signed. Rather, the digest of its header and payload, separated by a '.' after being converted to base64, is signed. Practically speaking, the digest of a JWT is signed; consequently, signatures can be verified on-chain *only knowing the digest of the JWT, which does not reveal the actual JWT*. The digest is checked via a ZK-SNARK.

Verification of RSA signature

Below is the verification function for signature s, public key *(e,n)*, keccak256 hash function H, and padding function P, where m is the message, decoded from base64 format.

$$s^e(mod\ n) == H(P((m)))$$

Signature verification is similar to how all blockchain transactions work; transactions must be signed by the party transacting and verified by the blockchain. Here, instead of solely checking that a transaction is signed by the sender, the contract also checks the credentials are signed by Google, Facebook, and etc.

Symmetric HMAC Signatures

HMAC signatures, such as those currently returned by Twitter, are not signed by an asymmetric key. They are not as straightforward to verify on-chain, since there is no public key anybody can use to verify a signature. Instead, the verifying key is identical to the signing key – anyone who has the key can forge signatures! This is incompatible with public blockchains as bad actors can forge signatures with publicly known signing keys. Rather, symmetric keys are meant to be known only by the app (e.g., website with Twitter sign-in) and the provider (e.g. Twitter). As a result, there is an added layer of centralization where Holo must reissue these tokens from a central server with an asymmetric signature. We mitigate this centralization risk by using a trusted execution environment.

Prevention of Front-running

Once a blockchain-compatible signature for credentials has been obtained, further steps are needed to proceed. The steps above by themselves would allow impersonation via front-running. Verifying a JWT on-chain leaves it in the mempool for anyone to steal and verify first by paying more gas. Holonym initially resolved this security concern by verifying the JWT in two transactions. The two transactions were a commit-reveal pattern:

1. XOR digest of the JWT with owner's address, and submit the hash of that result
2. Wait for the current block to be finalized
3. Submit the plain-text JWT for the smart contract to check that it was committed in a previous block.

When 3. happens, the smart contract checks for a previous commitment; it XORs the hashed JWT with the sender's public key, then hashes the result. If a result exists and was submitted in an earlier block, it may check the result was not only known but also linked with the user's public key. In this block, the JWT remained unknown to all but the user because it was hashed and subject to XOR before it was shared. After these steps confirm a JWT belongs to the submitter of the transaction, its signature can then be verified as mentioned previously.

Now, Holonym uses a ZK-SNARK to do the above in a single transaction comprised of:

- signature of the JWT, which is recovered on-chain to find the digest of the JWT
- proof that the preimage of the digest of the JWT does indeed represent a valid JWT
    - the address of the sender as part of the proof

This way, the proof is not malleable regarding the address; nobody can claim it as their own because the address is part of the proof.

Trust-less Wallet Recovery

There are two key barriers to wallet recovery: trust and ease of setup. A trusted custodian is against the ethos of web3. And "social recovery" doesn't fully solve this problem – it delegates trust to people rather than institutions, and requires time to setup or make transactions.

Rather, wallets can be access-gated by a zero knowledge proofs of identity. Government ID, login with Google or Apple, or MacBook fingerprint reader should enable recovery of a wallet. For example, a 2/4 threshold for these authentication methods could be used where a login with Google and biometric scan can recover your account. Thus, there is no one custodian that can single-handedly steal funds. Setup via clicking "sign-in with Google/Apple/Microsoft" or "enter your legal name" , is arguably a simple UX for wallet-creation, enabling both ease of use and non-custodiality.

For the setup process, an encrypted version of the UUID, such as the JWT *sub* claim, is stored on-chain. For privacy, the encryption process must be such that an Google, Apple, and other identity providers cannot tell which UUID produced the encrypted UUID - doing so would enable them to know who owns which address. As long as the user can prove a verified identity with the same UUID, they can recover the wallet. ZK-SNARKs are used for private proofs of JWTs. For non-smart-contract wallets, the Lit protocol can share the secret after identity has been proven.

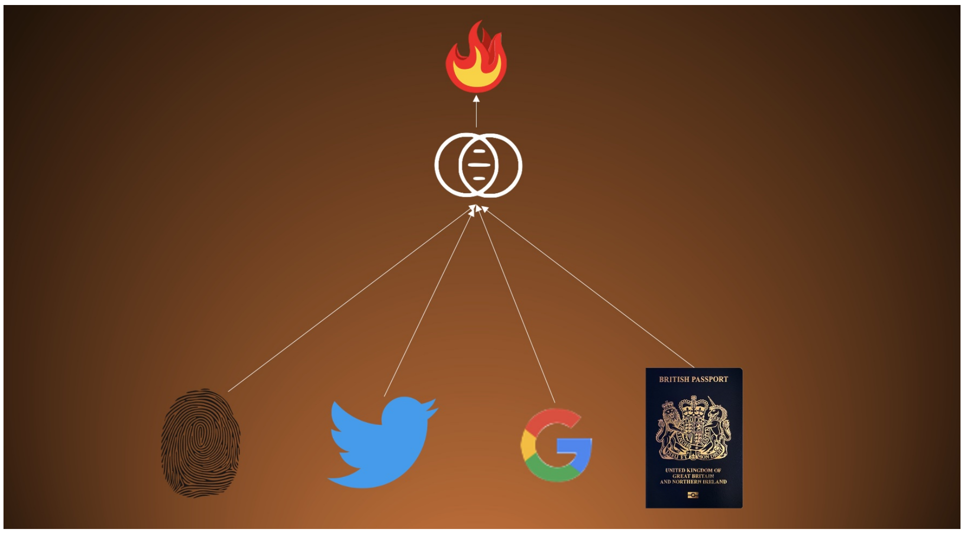

Figure 3: Wallet recovery using a MacBook fingerprint scanner, Google account, Twitter account, and passport. Access to keys granted by a proof via Holo (middle), and keys stored on Lit protocol (top).

Conclusion

Holonym is an identity bridge and mixer that privately cloaks off-chain credentials, putting them on-chain for portable use by any web application. An array of new use cases are unlocked by combining the composability and reliability of public ledgers with the privacy of zero-knowledge. These unlocks include on-chain identity verification, rigorous Sybil resistance, digital crime prevention, and noncustodial wallet recovery. While they rely on composable ledgers, they do not just benefit the web3 ecosystem. A key outcome of the architecture employed by Holonym is the elucidation of novel, practical utility of blockchains to benefit the internet at large, outside of web3 bubbles, by reducing crime, Sybil attacks, and data leaks.

References

[1] Chainalysis. (2022, February). *The 2022 Crypto Crime Report*. Chainalysis. Retrieved March 27, 2022, from `https://go.chainalysis.com/rs/503-FAP-074/images/Crypto-Crime-Report-2022.pdf`

[2] Elizabeth Licorish. (2021, November). *Chainlink Announces Its Total Value Secured (TVS) Is Now Over $75 Billion* Chainlink Today. Retrieved May 13, 2022, from `https://chainlinktoday.com/chainlink-announces-its-total-value-secured-tvs-is-now-over-75-billion/`

[3] @rchen8. (2022, March 27). Dune Analytics. *OpenSea.* Retrieved March 27, 2022, from `https://dune.xyz/rchen8/opensea`

[4] Bank Secrecy Act. *31 USC § 5311.* Retrieved Oct 19, 2022, from `https://www.govinfo.gov/content/pkg/USCODE-2012-title31/pdf/USCODE-2012-title31-subtitleIV-chap53-subchapII-sec5311.pdf`

[5] Customer identification program requirements for banks. *31 CFR § 1020.220.* Retrieved Oct 19, 2022, from `https://www.law.cornell.edu/cfr/text/31/1020.220`

[6] OpenSea. (2022, January 27). *OpenSea Tweet.* Twitter. Retrieved March 27, 2022, from `https://twitter.com/opensea/status/1486843201352716289`

[7] @RoboTeddy. (2021, June). *Proof of humanity: The cost of attack.* HackMD. Retrieved March 27, 2022, from `https://hackmd.io/@RoboTeddy/SkFEYwptd`

[8] W3C. (2022, March). *Verifiable Credentials Data Model v1.1* . W3C. Retrieved May 13, 2022, from `https://www.w3.org/TR/vc-data-model/`

[9] Lit Protocol. (2021). *Automate & Free the Web.* Lit Protocol. Retrieved October 19, 2022, from `https://litprotocol.com`

[10] Nick Szabo. (1995). *Smart Contracts Glossary.* Nakamoto Institute. Retrieved June 29, 2022, from `https://nakamotoinstitute.org/smart-contracts-glossary`

[11] John R. Douceur. (2019). *Tornado Cash Privacy Solution Version 1.4.* Tornado. Retrieved June 29, 2022, from `https://tornado.cash/Tornado.cash_whitepaper_v1.4.pdf`

[12] John R. Douceur. (2002). *The Sybil Attack.* Nakamoto Institute. In: Peer-to-Peer Systems. Ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260

[13] Goldwasser, S., Micali, S., & Rackoff, C. (1985). *The knowledge complexity of interactive proof-systems.* In Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali (pp. 203-225).

[14] Vitalik Buterin. (2021). *Why we need wide adoption of social recovery wallets.* Vitalik Buterin's Website. Retrieved June 29, 2022, from `https://vitalik.ca/general/2021/01/11/recovery.html`

[15] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, Madars Virza. (2014). *Zerocash: Decentralized Anonymous Payments from Bitcoin.* Proceedings of the IEEE Symposium on Security & Privacy (Oakland) 2014, 459-474, IEEE, 2014. Retrieved September 21, 2022, from `http://zerocash-project.org/media/pdf/zerocash-oakland2014.pdf`