League of Identity
&
Bank3

A cryptographic threshold network implementing distributed identity-based encryption and authentication from Google and other social identity providers, with applications to secure and anonymous identity-based web3 payments via ZK proofs

Vincenzo Iovino, Alex Kampa, and Nil Soroush

AZKR Research

April 2025 - *draft*

Abstract

Identity-based encryption and signatures (IBE/IBS) were introduced by Adi Shamir in 1984 as a method to simplify certificate management in email systems. Despite extensive research on the subject, IBE has never been adopted in real-world applications. The main challenges lie in the difficulty of establishing concrete IBE infrastructures for identity management and the absence of unified identity standards.

In this work, we address this gap by proposing and implementing the **League of Identity** (LoID), an IBE/IBS system and tooling that leverages existing real-world identity providers.

With LoID, users can obtain a cryptographic token linked to their Gmail or Facebook account, phone number (linked to their Google account), social security number (via their digital identity card), Ethereum account, and more. These tokens can be acquired through a practical Single Sign-On (SSO) process for Google, Facebook, and other social providers, or by signing a document with a digital ID card or an Ethereum or other web3 wallet.

The token can be used to sign messages or transactions on blockchains, or to decrypt secret messages.

As an application of LoI, we propose the first **anonymous identity-based payment system for web3**, which we call **Bank3**, allowing crypto users to send crypto assets to non-crypto users, an advancement of independent interest.

1 Introduction

Background. Identity-based encryption and signatures (IBE/IBS) were proposed by Adi Shamir in 1984 [1] as a way to simplify certificate management in email systems. Despite extensive research on this topic, IBE has been never been used in the "real world". This is largely due to the difficulty of creating infrastructures for identity management and the absence of unified identity standards. To make IBE/IBS practical we propose the following problem statement for which we show practical solutions along with a proof of concept.

Problem statement. We wish to implement a system that satisfies the following properties. A user should be able to use his existing email or social media account (Google, Facebook, Twitter, etc.) to engage with in other distributed systems, e.g. a blockchain or a Decentralized Autonomous Organization (DAO).

In addition, we would also like to have identity-based encryption functionalities: a user Alice should be able to encrypt a message for the email address *bob@gmail.com* without interacting with Bob. Bob in turn should be able to decrypt a ciphertext associated with his Gmail address just by using his ability to log into Google.

Why do we need cryptography? Consider the following scenario. An organization wishes to set up a Decentralized Autonomous Organization (DAO) exclusively for individuals who have an email address with the organization's domain name (e.g., @azkr.org). A trivial solution is have a "gateway" verify that users are indeed able to log into their email account. The gateway can then authorize user to sign transactions on a blockchain. For example, the gateway could provide a valid public/secret key pair to users, and add the public keys to the DAO.

The downside of this solution is that the gateway would have to be fully trusted. Moreover, it is not clear how identity-based encryption could be efficiently implemented.

Our model To solve the issue we envision the following model and we will later show how to implement it.

We assume that there is a set of n nodes, whom we which we call *League of Identity (LoID)*, and who share a master secret key MSK without actually knowing MSK. Each node $i \in [n]$ keeps a share MSK_i of MSK. If some threshold number t of nodes (e.g., the majority) were to collude, they could reconstruct MSK. The assumption, however, is that most nodes act correctly and do not collude. Corresponding to MSK there is a master public key MPK that is a public parameter of the system.

A user Bob with email bob@gmail.com can log into Google and obtain an OAuth 2.0 access token. This is a well-defined string that is associated with the address bob@gmail.com and which can be verified using public Google APIs. The user Bob then sends his access token to a subset S of the nodes of cardinality t. Each node $i \in S$ replies with a share $T_{Bob}(i)$ and from these t shares Bob can

reconstruct a token T_{Bob}. This token can be used in two ways.

- For encryption/decryption. A user Alice can encrypt a message m using just the master public key MPK and the email address bob@gmail.com to produce a ciphertext CT. Only Bob, having T_{Bob}, can decrypt CT and recover m.
- Authentication and signatures. The token can be used to sign a message m in the following sense. The signature algorithm takes as input the token T_{Bob} associated to the email bob@gmail.com, the master public key MPK and a message m (e.g., a transaction to sign) and produces a signature σ. This signature is a well-defined string that can be transmitted together with a transaction.

  The verification algorithm takes as input the master public key MPK, the email bob@gmail.com and the signature σ and outputs 1 or 0 to denote acceptance or rejection of the signature. The signature should satisfy the usual properties of digital signatures adapted to this context, namely completeness and unforgeability (e.g., hardness of computing signatures of messages for which no signature was observed before). The token can be also used to authenticate interactively in a way that a user Alice is able to verify the identity of the user Bob but Alice is unable to reuse the transcript of the protocol to impersonate Bob (that is, Bob can convince Alice to be the actual Bob but after this interaction Alice is unable to convince Charlie of the false claim "I am Bob"). Furthermore, we envision that the verification of the signature should be efficient for web3 applications. For instance, it is efficient to verify signatures of digital identity cards off-chain but verification on Ethereum of such signatures would consume too much gas to be practical.

The previous properties enable very powerful applications. It can allow users with access tokens from Google and similar services to participate in a blockchain or DAO. Moreover, one can extract useful information from access tokens, for example pictures of the users. Such pictures and additional info can be profitably used in the blockchain. As an example the access token for Instagram could be used to extract information about the number of followers of users and create a DAO of influencers. As another example profile pictures could be exploited in the following way: a group of people can add the same banner to their own profile pictures (e.g., a banner about some social cause) to create a DAO of people united by the some cause.

Token validity For some applications a token should not be valid forever (accounts can be deleted or blocked). One can associate e.g. a month+year to the token so that its period of validity is publicly visible. An alternative solution would be to resort to more advanced functional encryption systems like wildcards IBE or Hidden Vector Encryption that would allow to decrypt past ciphertexts with a single token.

Verifiability and privacy In the context of our system there are different levels of verifiability. The best property would be to guarantee that even if

all nodes in the League of Identity network collude together, they cannot forge valid signatures under identities they do not control. This property cannot, however, be achieved. Instead, for encryption and privacy, the security must rely on threshold assumptions: a sufficiently large set of nodes can always break the privacy.

Having verifiability based on threshold assumptions is more general and offers more flexibility. Some providers can only offer online deniable verification for their access tokens or for particular features (e.g. linking the identity to a phone number). Therefore, in this work we assume that the providers's access tokens are verified online by the LoID nodes and thus both the verifiability and privacy are under a threshold assumption. In our system the threshold can be completely arbitrary (not necessarily majority). Choosing a higher threshold offers more security, but is less robust in case of faults and adds more communication overhead for users.

Threshold security against providers. In the basic setting providers like Google must be trusted. It is possible to consider the following variant. Let us assume for simplicity that Bob has a Facebook account with email address *bob@gmail.com*. Bob logs into both his Google and Facebook account to request the respective access tokens and sends both access tokens to the League of Identity nodes which grant to Bob a token for a joint $google + facebook$ identity. Abstractly, the provider can be seen as the intersection of Google and Facebook. That way, the security of Bob can be broken only if Google and Facebook collude, or if both accounts are compromised simultaneously.

Applications In Section 3 we propose our anonymous payment system that enables crypto users to send crypto assets to non crypto users.

Other applications include the creation of DAOs based on real-world identities or organizations or e.g. the DAO of influencers having $>$ 500k followers, or the DAO of all members of an organization sharing the same email domainetc. These DAOs can have at same time verifiability for the members and *encrypted proposals*.

Related work IBE/IBS were proposed by Shamir in '84 [1] but concretely implemented only in 2001 by the breakthrough work of Boneh and Franklin [2]. Recently, there has been interest in deploying identity-based solutions for the web3 [3, 4, 5]. We remark that none of them implements IBE and as such cannot implement the applications that we propose in Section 3. Moreover, the latter solutions require expensive SNARK proofs for circuits relative to JSON Web Token computation and as such are not easy to adapt to providers' APIs which do not provide verifiable tokens.

2 Our implementation

2.1 The Setting

We are in a bilinear group (G_1, G_2, G_T, e) of type 3, of prime order p, with generators g_1, g_2, and $g_T = e(g_1, g_2)$.

H is an hash to point function that maps arbitrary length bit strings to G_1, and is indistinguishable from a random oracle.

H_T is a function that maps G_T into bit strings of fixed output length and is indistinguishable from a random oracle. A message m is said to be compatible with H_T if its bit length is less than or equal to the output length of H_T.

2.2 The Master Secret Key and its Shares

Of critical importance to the The League of Identity is its master secret key $MSK \in \mathbb{Z}_p$, which is related to the master public key MPK as follows:

$$MPK = g_2^{MSK}$$

The master secret key is not known to any of the nodes, but each node $i \in [n]$ has a share $MSK_i \in \mathbb{Z}_p$ of MSK. Any subset of $t < n$ nodes can reconstruct MSK. The main security assumption of LoID is that honest nodes will not participate in reconstructing MSK, and that at least $n - t + 1$ nodes are honest.

The MSK will typically be the constant term of a polynomial, and the shares MSK_i will be evaluations of that polynomial at predetermined points. Given a set S of t shares, the Lagrange coefficients $\lambda_i(S)$ for that set can be computed and we have:

$$MSK = \sum_{i \in S} \lambda_i(S) \cdot MSK_i$$

2.3 Distributed Key Generation

We assume that the nodes have performed a Distributed Key Generation (DKG) procedure to calculate the shares MSK_i, without any of the nodes learning the value of MSK. Several such protocols exist, one could for example use the DKG protocol used by drand [6].

When new nodes enter or leave the system, a new DKG procedure can be run to re-share the same master secret key. That way the master public key remains unchanged. This keeps things simple but potentially weakens security. To see why, suppose that we start with 10 nodes and $t = 5$. If we then increase the number of nodes to 20, with $t' = 10$, then it would take 10 of these 20 nodes to reconstruct the MSK. However, 5 of the 10 origial nodes would also still be able to reconstruct MSK.

2.4 LoID Token Generation

In the following, we will consider the user Bob with a given email or social media identifier string *uid*. This *uid* could for example be his email address bob@gmail.com.

Bob starts by obtaining an OAuth token from his email or social media provider, corresponding to his *uid*.

Bob then selects a random set S of t nodes, and requests a LoID token share from each of them, attaching his OAuth token to each request. In response to a valid OAuth token, each node $i \in S$ returns a token share of the form:

$$T_{Bob}(i) = H(uid)^{MSK_i}$$

From t values received, Bob can reconstruct his LoID token T_{Bob}:

$$T_{Bob} = \prod_{i \in S} T_{Bob}(i)^{\lambda_i} = H(uid)^{\sum \lambda_i MSK_i} = H(uid)^{MSK}$$

This LoID token can then be used for encryption and authentication/signatures.

Token validity? Restrictions? TODO add section here

2.5 Encryption and Decryption of Messages

For encryption/decryption, we can use the Boneh and Franklin AnonIBE [2] scheme.

Encryption. Alice wishes to encrypt a message m for Bob whose identifier string is *uid*. We assume that the length of m is compatible with H_T. Alice chooses a random value $s \in \mathbb{Z}_p$ and computes:

$$A = g_2^s \qquad \begin{aligned} b &= e(H(uid), MPK^s) \\ B &= m \oplus H_T(b) \end{aligned}$$

She then sends the cyphertext $CT = (A, B)$ to Bob.

Decryption. To decrypt the cyphertext, Bob first recovers b as follows:

$$\begin{aligned} e(T_{Bob}, A) &= e(H(uid)^{MSK}, g_2^s) \\ &= e(H(uid), g_2^{s \cdot MSK}) \\ &= e(H(uid), MPK^s) \\ &= b \end{aligned}$$

Bob can now recover the message m:

$$\begin{aligned} H_T(b) \oplus B &= H_T(b) \oplus m \oplus H_T(b) \\ &= m \end{aligned}$$

This system is IND-CPA secure and can be made IND-CCA secure using the Fujisaki-Okamoto's transform as explained in [2].

2.6 Signatures and Authentication

We first focus on signatures. Bob wishes to sign a message m or arbitrary length. He chooses a random value $r \in \mathbb{Z}_p$ and computes the following:

$$\begin{aligned} C &= MPK^r \\ E &= g_1^r \\ F &= T_{Bob}^r \end{aligned}$$

In addition, the user computes a proof π of knowledge of the exponent r in E, binding the message m to that proof. This is done via a non-interactive Schnorr proof in which the challenge is set to $H(t||E||m)$, where t is the first message of the Schnorr protocol. Details are provided in Appendix A.

The signature of the message m consists of the tuple $\sigma = (C, E, F, \pi)$.

The verifier needs to verify the following:

$$\begin{aligned} e(E, MPK) &\stackrel{?}{=} e(g_1, C) \\ e(H(uid), C) &\stackrel{?}{=} e(F, g_2) \\ \pi &\stackrel{?}{=} \text{valid proof} \end{aligned}$$

If the signature is correct, we do indeed have:

$$\begin{aligned} e(E, MPK) &= e(g_1^r, MPK) \\ &= e(g_1, MPK^r) \\ &= e(g_1, C) \end{aligned}$$

$$\begin{aligned} e(H(uid), C) &= e(H(uid), MPK^r) \\ &= e(H(uid), g_2^{r \cdot MSK}) \\ &= e(H(uid)^{r \cdot MSK}, g_2) \\ &= e(T_{Bob}^r, g_2) \\ &= e(F, g_2) \end{aligned}$$

The first verification $e(E, MPK) \stackrel{?}{=} e(g_1, C)$ convinces the verifier that

$$E = g_1^r \quad \text{and} \quad C = g_2^{MSK \cdot r}$$

The second verification $e(H(uid), C) \stackrel{?}{=} e(F, g_2)$ convinces the verifier that

$$F = H(uid)^{MSK \cdot r}$$

The verifier then verifies that π is a Schnorr proof computed including E and m in the challenge. Since a valid Schnorr proof can only be computed with knowledge of r, this convinces the verifier that the user knows a valid LoID token T_{Bob} without disclosing it. Since only Bob can get the token from the LoID nodes, this is a proof that the signature was from Bob.

Authentication can be done using the interacive Schnorr protocol, without binding the challenge to any message.

3 Anonymous Identity-Based Payments (AIBP)

3.1 Overview

Recently, some financial applications enable payments via phone numbers and other real-world identities. One example is the Polish BLIK payment system [7]. Similarly, we propose what we term an *Anonymous Identity-Based Payment* (AnonIBP, or AIBP) system.

Using an AIBP, Alice can deposit crypto assets in favor of Bob's email address, social identity, or any service that supports OAuth 2.0.

Bob, who is not necessarily a crypto user, can verify that there are crypto assets deposited for him. He does not need a web3 wallet to do this. All he needs is to log into Gmail, Facebook etc. account. The assets are securely and anonymously deposited into an on-chain program.

Bob in the future can decide to become a crypto user and to install a wallet and he will be able to withdraw in favor of any Eth address - his own or someone else's address. The system can also be extended to enable Bob to withdraw in favor of other Gmail, Facebook, etc. accounts without even having a wallet; the GAS fees will be paid by paymasters that can take a fee for this service. Both deposits and withdrawals are fully anonymous, that is they do not reveal the Gmail, Facebook, etc. account.

3.2 A Trivial AIBP System

We can implement a trivial AIBP using our IBE system described in section 2.5. We assume that this is done on a public blockchain that supports on-chain programs[1].

Suppose that Alice wants to send assets to Bob. She chooses a random nonce x and computes the ciphertext $CT = (A, B)$ that encrypts x for Bob's identity uid (cf. Sectionn 2.5). She also computes $y = Hash(x)$, with the hash function being one that can be efficiently computed on the chosen blockchain. Alice then sends the assets along with the pair (CT, y) to the on-chain program. Notice that, as our IBE scheme is anonymous, the ciphertext CT hides Bob's identity.

[1] Programs that are deployed and executed on-chain are called "smart contracts" on many blockchains, including most EVM-compatiblle chains, "runtime modules" in Polkadot, "chaincode" in Hyperledger Fabric, "XRPL Hooks" in Ripple etc. The expression "on-chain program", as used in Solana, seems to most accurately describe their fundamental nature.

Using the LoID token relative to his identity, Bob can verify which deposits (if any) are for him. For each pair (CT, y), Bob can run the decryption procedure on CT, obtaining a result x. Then, if $Hash(x) = y$, Bob will know that this deposit was made for him. To do this, Bob does not need to have a web3 wallet.

To claim the deposit, Bob simply sends x to the program. The program verifies that $Hash(x) = y$, and if the check is successful, transfers the assets to Bob's wallet.

One issue with this trivial AIBP system is that the sender can also withdraw the assets at any time. This can be avoided only in applications in which sender and receiver can communicate at deposit time: in that case Bob can compute (CT, y) himself and send it to Alice, who will use this data for the deposit. Another issue is that, when claiming a deposit, the value x could be intercepted.

3.3 Secure AIBP via ZK proof of correct decryption (Bank3)

We now describe a fully secure AIBP scheme which does not suffer from the drawbacks of the trivial system. Hereafter, this scheme, as well as any on-chain program implementing it, will be called *Bank3*.

Sending assets As before, Alice wishes to deposit crypto assets in favour of Bob. The process is as follows.

1. choose a random value $r \in \mathbb{Z}_p$
2. compute the IBE ciphertext $CT(r, uid)$
3. compute $D = H(uid)^r$
4. send assets to the Bank3 program, together with values CT and D.

Verification of deposits To verify that a deposit is for him, Bob proceeds as follows.

1. reads the values CT and D from the Bank3 smart contract
2. decrypts CT using his LoID token and obtains a value r
3. checks if $D \stackrel{?}{=} H(uid)^r$

If the equality check is successful, Bob knows that the deposit has been made in his favour (i.e. in favour of his uid).

Note that the values CT and D are public, and reading them does not require a web3 wallet. Steps 2 and 3 are carried out off-chain, for example in Bob's browser. In step 2, the decryption process always returns some value. If the CT was not created for Bob, that value will just be a random bit-string, and the check at step 3 will fail.

Claiming of deposits We assume that Bob has read the values CT and D from the Bank3 program and verified that the corresponding deposit was made in his favour. He now has the following data:

- The value r obtained from decrypting CT (which is such that $D = H(uid)^r$)
- The value D read from the blockchain
- the address $addr$ to which he wants to send the assets

The blockchain account $addr$ can belong to Bob or to a third party. The procedure is as follows:

1. choose a random value $s \in \mathbb{Z}_p$
2. compute $E = D^s$
3. produces a non-interactive Schnorr proof of knowledge π of the discrete long of E in base D (i.e., knowledge of s), tying the withdrawal address $addr$ to this proof by adding it to the challenge string
4. compute the re-randomized token $T' = (T_{Bob})^{r \cdot s)}$
5. send $(E, \pi, T', addr)$ to the Bank3 program.

The Bank3 program does the following checks:

1. verify the validity of π
2. check if $e(T', g_2) \stackrel{?}{=} e(E, MPK)$
3. if the verification was successful, assets are transferred to $addr$

The verification of π proves that Bob knows s such that $E = D^s = H(uid)^{r \cdot s}$. This verification is only possible if the correct $addr$ was used when computing the challenge, so it also serves to verify that it s the correct withdrawal address. For the second check, we have:

$$\begin{aligned} e(T', g_2) &= e((T_{Bob})^{r \cdot s}, g_2) \\ &= e((H(uid)^{MSK})^{r \cdot s}, g_2) \\ &= e(H(uid)^{r \cdot s}, g_2^{MSK}) \\ &= e(E, MPK) \end{aligned}$$

If these checks pass then T' is a correct re-randomized token, and this convinces the verifier that Bob knows T_{Bob} which is a valid token for the identity uid.

Withdrawals by and/or in favour of non-crypto users A natural extension of Bank3 is to enable withdrawal by, or in favor of, non-crypto users. Assume that a deposit has been made in favour of Bob. There are three scenarios that are not covered by the Bank3 program described above::

1. Bob has a web3 wallet, and wishes to transfer the deposit to another email or social media account
2. Bob does not have a web3 wallet, and would like to transfer the assets to some *addr* on the blockchain
3. Bob does not have a web3 wallet, and would like to transfer the deposit to another email or social media account (whose owner may not have a web3 wallet either)

Scenario 1 can be dealt with easily, by adding a "withdraw+redeposit" function to the Bank3 program. For the other two scenarios, we will need a service that will interact with the Bank3 program on behalf of users, paying the blockchain transaction fees. This service could be run by what we call "paymasters"[2]. Paymasters would be compensated with a small amount of the assets deposited in Bank3. To avoid abuse, the maximum amount that a paymaster can charge for a given transaction could be added to the proof π, in the same way as we did with *addr*.

Partial or Batch withdrawals The system can be generalized to allow for partial or batch withdrawals.

4 PoC implementations

In [8] we propose a PoC implementation of the system described in this paper along with the corresponding documentation. In [9] we additionally implemented smart contracts of the AIBP system of Section 3 and of a DAO for organisations using Google accounts.

A demo of Bank3 system is accessible at `https://bank3.azkr.org` and supports payments in favour of Gmail accounts.

References

[1] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology - CRYPTO 1984*. Springer, 1985.

[2] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology - CRYPTO 2001*. Springer, 2001.

[2] This is different from concepts like "gas stations" on Near Protocol, or "relayers" on some other chains. There, users may not need to pay for transactions, but they still need a wallet to interact. With our concept of paymasters, users will not even need to have a wallet.

[3] Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Yan Ji, Jonas Lindstrøm, Deepak Maram, Ben Riva, Arnab Roy, Mahdi Sedaghat, and Joy Wang. zklogin: Privacy-preserving blockchain authentication with existing credentials. arXiv, 2401.11735, 2024.

[4] Privacy Preserving Exploration Team. Anon Aadhaar. `https://github.com/anon-aadhaar/anon-aadhaar`, 2023.

[5] Florent Tavernier. proof-of-passport. `https://github.com/zk-passport`, 2024.

[6] drand. Distributed randomness beacon. `https://drand.love/`.

[7] Wikipedia. BLIK. `https://en.wikipedia.org/wiki/Blik`.

[8] Vincenzo Iovino. League of Identity - PoC. `https://github.com/aragonzkresearch/leagueofidentity`, 2024.

[9] Vincenzo Iovino. LoI.SmartContracts. `https://github.com/vincenzoiovino/LoI.SmartContracts`, 2024.

A Bound Schnorr Proofs

We recall the Schnorr proof of knowledge of exponent in a group of order p and generator g. The prover wants to prove to the verifier knowledge of the exponent of $y = g^x$. The prover knows x, but the verifier knows only y. The interactive version of the proof is shown in Table 1.

Prover		**Verifier**
$e \xleftarrow{\$} \mathbb{Z}_p$		
$t = g^e$		
	$\xrightarrow{\text{Commitment } t}$	
		$c \xleftarrow{\$} \mathbb{Z}_p$
	$\xleftarrow{\text{Challenge } c}$	
$d = e + cx$		
	$\xrightarrow{\text{Response: } d}$	
		$g^d \stackrel{?}{=} t \cdot y^c$

Table 1: Interactive Schnorr Protocol

If the prover is correct, we will indeed have $t \cdot y^c = g^e \cdot g^{cx} = g^{e+cx} = g^d$

In the non-interactive version, the prover creates the challenge by hashing the commitment t. At that stage, it is possible to concatenate t with some arbitrary string m before hashing, this "binds" the string m to the proof.

Prover		**Verifier**
$e \xleftarrow{\$} \mathbb{Z}_p$		
$t = g^e$		
$c = H(t\|\|m)$		
$d = e + cx$		
	$\xrightarrow{(t,c,d,m)}$	
		$c \stackrel{?}{=} H(t\|\|m)$
		$g^d \stackrel{?}{=} t \cdot y^c$

Table 2: Non-Interactive Schnorr Protocol with Extra Data