The Pseudonymous Bridge

Lev Soukhanov[1] and Yaroslav Rebenko[2]

[1]Solcery
[2]Yandex Distributed Computing Team

2022

1 Introduction

In this text, we describe an application of zero-knowledge proofs (and a very simplistic MPC protocol) to two practical problems.

First problem is the **permissionless pseudonymous bridge**: given some public social network or forum (say, twitter), create a "pseudonymous bridge" to some new system with the following properties:

1. One can register at most one pseudonymous account for every controlled account in the original public social network
2. One can prove statements regarding their account, in zero knowledge.
3. The anonymity set is the set of all relatively active users of the network (publishing at least one public message in a validation epoch).
4. This new system is auditable, there is no trust in the server required. We typically assume it should run on a public blockchain.

One of the authors (but we won't tell a reader which one) believes that such a system could drastically improve the quality of discussion, striking a nice balance between the anarchy of anonymous forums and dictate of modern social networks.

Provided the public network is sybil-resistant enough, this, ironically, would provide a great alternative to the decentralized identity systems [1].

The term "pseudonymous bridge" was coined by Balaj Srinivasan [2], but we expand it. Balaj, in his talk, suggests "assume there is a crypto-twitter", meaning: "assume every user in a public network has an access to a private key which they use to sign messages". Then, one could use zk-proofs in a trivial manner to achieve the goal.

In practice, there is no crypto-twitter. However, as we show, it is actually not needed: a bit more elaborate protocol allows to create a pseudonymous bridge from *any* public social network, without permission. We stress that it also does not require steganography or even publishing of any cryptographic data in this public social network, and the anonymity set is the set of all relatively active users of the network.

The second problem is **private contact search**. This is a typical problem for privacy-focused messenger applications: for example, Signal does not learn anything about user's contact list, but it needs to expose their phone number to everyone. This is also a core issue preventing us from creation of scalable decentralized webs of trust.

We outline the problem more precisely. **Private contact search** assumes there exists some list of public identities (say, accounts in some messenger). User i, registering in the new, private messenger lists a set C_i of trusted contacts in an old public system. The following properties should hold:

1. If $j \in C_i \wedge i \in C_j$, then i and j learn their respective private identities.
2. If $j \in C_i$, j learns i's intent, even $i \notin C_j$ and vice versa.
3. Otherwise, they learn nothing about each other (not even if other is registered in the system).

This can be trivially (while a bit clunky in practice) done provided users i and j share some secret (say, hash of their first 10 messages in their public messenger). However, an attacker who is able to read their correspondence will also be able to both impersonate users and learn they are, in fact, in contact. A simple trick similar to what we do in the first part prevents this issue.

Our schema is not post-quantum: it is based on DDH assumption. Significantly determined quantum adversary is able to deanonymize the participants. It can be modified for post-quantum resistance, albeit, at a cost of significant reduction in efficiency.

Authors would like to thank participants of D.A.O. cryptographic sessions and Louis Guthmann for their suggestions and useful discussions.

2 Preliminaries

2.1 ZK-proofs

We assume the reader is familiar with ZK-SNARKs, if not one can look at some modern constructions [3], [4]. We will use ZK-SNARKs as a black-box for our construction, so we will only give informal descriptions of their properties:

ZK-SNARK (which means zero knowledge succinct non-interactive argument of knowledge) is a pair of programs, prover and verifier. Suppose we have fixed some computable in polynomial time predicate $f(x, y)$ depending on two parameters, first one referred as "public" and second as "private".

Actually, one can use an universal predicate instead and hide the dependance from f into x, but for the practical constructions it is better to assume that f is fixed.

Prover is a probabilistic machine $P_f : x, y \mapsto P_f(x, y)$ which constructs a "proof" - some sort of cryptographic cerificate.

Verifier $V_f : x, P \mapsto$ Bool takes as an argument the public parameter x and a proof P and either accepts or declines.

We state informally the conditions on these functions (and encourage to look up references for more formal definitions):

1. (Completeness) $\forall x, y : V_f(x, P_f(x, y))$ - verifier should always accept a correct proof.
2. (Computational knowledge soundness) For a given x and an adversary constructing a proof P such that $V_f(x, P)$ there exists an extractor constructing a witness y from x and a state of adversary.
3. (Succinctness) The size and verification speed of the proof should be fast - typically, constant or at least polylogarithmic in an input size.
4. (Perfect zero-knowledge in random oracle model) This is an intricate notion. We encourage reader to read about it elsewhere, but outline it here briefly. Intuitively, it should capture the property that the proof does not reveal any useful information about the private value y. The non-interactive arguments we use are actually constructed from so-called interactive Σ-protocols, which consist of a sequence of prover's messages and verifier's random responses (called challenges). Such Σ-protocol is called Honest Verifier Perfect Zero Knowledge protocol, if there exists a Simulator program, which produces the correct transcripts of the protocol (playing for both prover and verifier but not knowing y), which are distributed exactly as real protocol transcripts with honest verifier. It captures an intuitive notion of not revealing any information about y - after all, any transcript of the communication obtained by the honest verifier could be instead sampled from the simulator. Now, Σ-protocols are typically compiled to the non-interactive arguments using Fiat-Shamir transformation, which replaces the challenges from the verifier by hashes of the transcripts of the protocol up to a point of challenge. In a random oracle model, hash function is modelled as random function. The simulator, then, is allowed to manipulate random oracle.

We will use this primitive as a black box, writing $P = \text{Prove}(y|f(x, y))$ for a proof, and $\text{Verify}(P, f, x)$ for verification. The concrete implementation is important, but is out of the scope of this paper.

We will also use few very concretely efficient ZK-proofs of some specific statements, which we will outline further.

2.2 Vector Schnorr proofs

Let us fix an elliptic curve E of prime order q such that DDH assumption is hard in it.

We recall that the standard Schnorr proof for a pair of points $A, B \in E$ is a proof of knowledge of $\lambda \in \mathbb{Z}_q$ such that $\lambda A = B$.

We will denote it as Schnorr$(A; B)$, and concretely it is a pair $(s \in \mathbb{Z}_q, R \in E)$ such that $sA = R + \text{Hash}(A; B; R; n)B$, where n is a nonce.

We will need a proof for the following specific statement:

For a set of points $A_1, ..., A_n, B_1, ..., B_n$ prove that one knows $\lambda \in \mathbb{Z}_q$ such that $\lambda A_i = B_i$ for all i. We will denote this proof as VSchnorr$(A_1, ..., A_n; B_1, ..., B_n)$, and propose the following concrete argument:

Algorithm 1 : Vector Schnorr proof

$X \leftarrow \text{ToPoint}\,\text{Hash}(A_1|...|A_n|B_1|...|B_n)$
$Y \leftarrow \lambda X$
for all i from 1 to n **do**
    $\lambda_i \leftarrow \text{Hash}(i|\,\text{Hash}(X|Y|A_1|...|A_n|B_1|...|B_n))$
end for
$\tilde{A}, \tilde{B} \leftarrow X + \sum_i \lambda_i A_i, Y + \sum_i \lambda_i B_i$
output $\leftarrow (Y; \text{Schnorr}(\tilde{A}; \tilde{B}))$

The verification of this argument proceeds by calculating Hash$(X|Y|A_1|...|A_n|B_1|...|B_n)$, then λ_i's according to their definition, then $\tilde{A}$, $\tilde{B}$, then verifying normal Schnorr proof.

The verification of this proof requires linear time (one needs to calculate X and then calculate all the hashes), but reading its public input also requires linear time, so it is efficient.

This proof is zero knowledge because the normal Schnorr proof is zero knowledge; the fact that it is sound is left as an exercise to the reader (the proof is similar to the standard Schnorr case).

2.3 Anonymous public board

We assume there exists a public board, in which each participant of a protocol can publish messages anonymously. Currently existing public blockchains are not a good model of this primitive, for their privacy is severely lacking.

Practically, it can be implemented as a network of nodes using Dandellion++ protocol, combined with a simple PoA blockchain.

3 Actors of the protocol

3.1 System

This is an authority in case of PoA. It accepts messages if they are constructed according to the protocol rules.

3.2 Applicants

These are all users attempting to register in a system. Each one has a unique key they use to sign transactions in a system.

3.3 Public network oracle

This is an entity which periodically publishes the Merkle root of all messages in twitter (with subtrees corresponding to individual accounts). This is, obviously, a computationally expensive procedure, but it is greatly parallelizable.

3.4 Salt workers

This is a set of special participants which indulge in a simple MPC protocol. Let us fix an elliptic curve E of prime order q such that DDH assumption is hard in it. Each salt worker i keeps a secret parameter $\lambda_i \in \mathbb{Z}_q$. Honest salt workers do not share their parameters with anyone.

4 The protocol

Naive approach to the problem would be forming some commitment to a future message m (here and after, we assume that messages include some metadata, at least the sending user and timestamp). However, if this commitment is hiding (say, Hash$(m|r)$), where r is some random string, one could commit to the same message twice and register two accounts. If the commitment would be linkable (say, Hash(m)), then an adversary could post-factum calculate hashes of all new messages in public social network, and check whether a user have registered or not.

The proposed solution involves forming a commitment deterministically using some simple MPC protocol; attacker will not be able to query MPC after the commitment phase have ended (and, therefore, when it knows any of the potentially committed messages).

Actually, we have two versions of this protocol, with different trust assumptions and tradeoffs. We will call them "1-of-n version" and "commitee version". They differ only

slightly in a setup phase, and in a final phase. We will primarily discuss 1-of-n version, and briefly outline the differences.

Algorithm 2 : Setup phase, 1-of-n version

Each salt worker i generates their private value λ_i independently
System generates single random point X of an elliptic curve
Each salt worker publishes $Y_i = \lambda_i X$

In a committee version, the only difference that there is a committee of some size (typically $n/2$), and the values λ_i are not independent but created using Shamir's secret sharing. If new salt workers are added or old are removed, re-sharing is required. In 1-of-n version, salt workers are fully independent and are free to leave or come every epoch without any communication.

Let us denote the set of applicants for the epoch k as $A_1, ..., A_m$.

Algorithm 3 : Commitment phase (goes for the whole epoch $k-1$)

for i from 1 to m **do**
Applicant A_i generates some private future message m_i
$r_i, r'_i \leftarrow$ Random
$C_i \leftarrow \text{Hash}(m_i|r_i)$ // commitment
$Q_i \leftarrow r'_i \,\text{ToPoint}(\text{Hash}(m_i))$
public board update $\leftarrow C_i, Q_i$
end for ▷ At this point, everybody is committed to some future message. This commitment is hiding - so somebody can potentially commit twice to the same future message. The values Q_i will be given to salt workers to process.
for j from 1 to n **do**
Salt worker S_j does the following:
for i from 1 to m **do**
$Z_{ij} \leftarrow \lambda_j Q_i$
end for
public board update $\leftarrow (Z_{1j}, ..., Z_{mj}), \text{VSchnorr}(X, Q_1, ..., Q_m; Y_j, Z_{1j}, ..., Z_{mj}))$
System accepts if a proof is correct
end for ▷ At this stage, some salt workers may have gone offline, or not manage to fulfill their obligations. Their contributions are deemed zero in the formulas below.
for i from 1 to m **do**
Applicant A_i does the following:
$F_i \leftarrow r_i'^{-1} \sum\limits_j Z_{ij}$
$P_i \leftarrow \text{Prove}(r'_i, r_i, m_i|$
$[\text{Hash}(m_i|r_i) = C_i] \wedge [r'_i \,\text{ToPoint}(\text{Hash}(m_i)) = Q_i] \wedge [r'_i F_i = \sum\limits_j Z_{ij})])$
System accepts if a proof is correct, and F_i did not occur previously.
end for

At the end of this phase, every applicant i has obtained an element F_i which is uniquely determined from their committed message m_i. However, we argue that in random oracle model and under DDH assumption an adversary having a message m can not, post-factum, determine if F_i corresponds to it (i.e., whether $m = m_i$). Indeed, if it could do it against a single salt worker (suppose all others are cooperative with an attacker), it would be able to solve DDH problem: in random oracle model the point $P = \text{ToPoint}(\text{Hash}(m_i))$ is distributed uniformly at random. Denote $Q = \text{ToPoint}(\text{Hash}(m))$. Then, adversary is given the following quadruple: $Q, \lambda P, r'_i P, \lambda r'_i P$, and adversary needs to determine whether it is a DDH quadruple. λ and r'_i are also distributed uniformly at random.

This phase is also the most computationally consuming, though it is greatly parallelizable.

After the end of epoch $k-1$, each applicant is supposed to publish the committed message m_i as their first in the epoch k, and in the next epoch participate in the Revelation phase:

Algorithm 4 : Revelation phase (occurs in the end of epoch k)

Public network oracle updates the Merkle tree.
Each applicant A_i proves in zero knowledge that there exists an account, such that its first published post in epoch k coincides with their committed message m_i
Successfully proving such statements grants registration in the system

5 Modifications in the committee case

The protocol above enjoys 1-of-n trust: it works if only a single salt worker is honest. It is also fault tolerant to a largest degree: because submissions from salt workers are accepted atomically, the salt workers who go offline during the work of a protocol do not change its behavior. However, there are also pretty significant downsides:

1. Bad user UX due to the necessity of passing each epoch's validation. If we say that conservatively each epoch lasts for a month, then the whole ritual needs to be passed every month, and the time for the first user registration is 2 months.
2. No negative reputation. There is no possible mechanism that prevents a user from dropping an account and creating a new one.

Both of these problems can be solved if one makes the result of salting consistent between epochs. But that comes at a cost.

Main change is, instead of choosing λ_i's independently, salt workers now use Shamir's secret sharing. In case the salt worker set changes, key redistribution is necessary.

Provided this is done, one can also greatly simplify the protocol, by making user first commit to a further message from some account and treat is a "log in" (non-linkable proof of controlling an account). Then, "logged in" users get an access to an MPC, to which they just give the hash of their unique identifier in the public social network:

Algorithm 5 : Log in phase

Applicant A sends a commitment $C = \text{Hash}(m|r)$ of some further message m.
In some further next epoch, they provide the proof P that there exists a message m in the Merkle tree of all posts of the later epoch than the epoch of commitment

Algorithm 6 : MPC phase

Applicant A provides to the MPC committee the value UidHash $= r'\,\text{ToPoint}(\text{Hash}(\text{uid}(m)))$, together with (C, P), and proof of the fact that the same m is used in C and UidHash.
MPC computes $Q = \lambda\,\text{UidHash}$, where λ is Shamir shared value
Applicant computes $F = r_i^{-1}Q$, and this becomes their unique identifier

The downside of this approach is pretty clear - if the deciding subset of committee is compromised, every user is deanonymized.

Another suggestion, made to us by Louis Guthmann, suggests that instead of predicting future messages, one could "log in" by wrapping JSON web token in a zero knowledge proof of validity. This would improve UX even more (and remove the burden of computing the Merkle tree of the whole twitter), but the downside is that an operator of the public network then could possibly attack a system without anyone recognizing.

Adding this as an additional check, though, is valuable because some accounts have predictable message pattern, and so can be captured because an adversary can guess a future message m. This provides a layer of protection versus such attempts.

6 Private contact search

One of the most exciting, in our opinion, applications of a (very similar) construction is "private contact search". Current contact search in messengers typically reveals a lot of data to a server (and in some cases, like Signal, to everyone in the app). This is a serious obstacle for the scaling of webs of trust, decentralized messengers and F2F networks: all of them typically rely on "small world" properties of the social graph, however in practice they have almost tree-like topology, most of the contacts being the one who invited the person, and their own invitees.

One possible approach to overcoming this obstacle could be the system which allows a person to list some "trusted contacts". These trusted contacts, then, would be able to recognize they are being searched by one of their peers.

Now, suppose for each two persons A, B there exists a common secret $S(A, B)$, not known to anyone else. Then, the problem is solved: everyone can publish (hashes of) their common secrets with trusted contacts in some public board.

If, for example, everybody have used a single, end-to-end encrypted messenger, A and B would be able to use as a common secret the hash of the first 100 messages of their correspondence.

However, in practice most of the messengers are not end-to-end encrypted (Telegram is not, and WhatsApp is, but most of people back up messages in cloud anyways).

The following trick solves the problem: to "log in", Alice commits to a future message in their chat with Bob.

The value r ToPoint(Hash(transcript of chat of Alice and Bob up to epoch k)) then is given to an MPC - which runs only if given an additional proof that the transcript contains a future message committed by Alice, at a timestamp later than the point of commitment. The result of an MPC is then divided by r, and used as a common secret of Alice and Bob.

References

[1] Divya Siddarth, Sergey Ivliev, Santiago Siri, and Paula Berman. Who watches the watchmen? a review of subjective approaches for sybil-resistance in proof of personhood protocols, 2020. https://arxiv.org/abs/2008.05300.

[2] Balaj Srinivasan. Towards a pseudonymous bridge, 2019. https://www.youtube.com/watch?v=VujbGiRUF8M.

[3] Jens Groth. On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Paper 2016/260, 2016. https://eprint.iacr.org/2016/260.

[4] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. *IACR Cryptol. ePrint Arch.*, 2019:1021, 2019.