$\in^{DA}$¢lub

Mr. Chewbacca

December 27, 2024

Abstract

Let's try to delete me from the chat with the approach described in the paper.

1 Introduction

No chats are governed by the community around which they were created. Some administrators with god's rights can decide whether users can be banned or deleted from the group. However, the control over a single message and the full chat history must belong to the users who created it.

Another challenge is having a well-defined community (with eligibility control) while achieving the anonymity of chat participants. Sometimes, it's very important to be able to prove you are part of a particular community but without a trace of the exact person. Bikers usually do that :)

In this paper, we propose a way to organize community chats without any centralized point of limitation and restrictions. We will use NFT ownership as a criterion for chat membership (there are an infinitive amount of criteria objects, but we will show how it works in the most popular way).

We love to drink, and we love crypto, so the target collection we will build a community prototype around be the following:
https://polygonscan.com/address/0x2841fb2d66260652f238b2afd2b232501eaf5303.

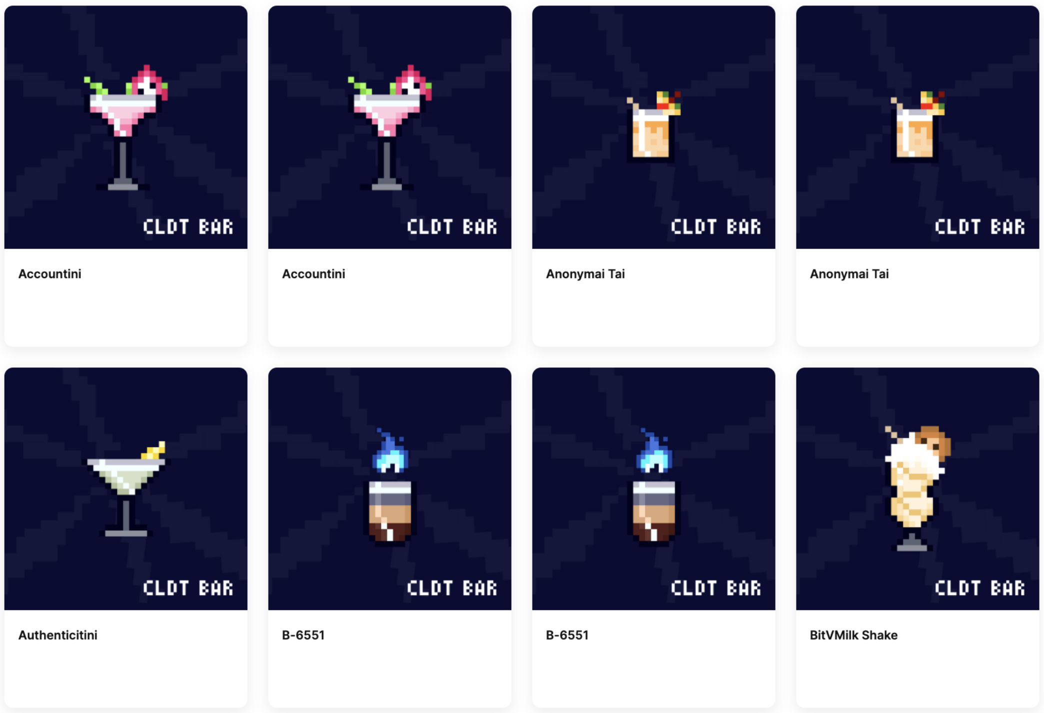

Figure 1: Mr. Cheubacca's Bar

2 Preliminaries

Let $\mathsf{B} = \{0,1\}$ and B^n a binary sequence with the length n. $\mathbb{F}_p$ is a finite field of primer order p. Let zkHash be the zk-friendly hash function that $\mathsf{zkHash} : \mathsf{B}^* \to \mathbb{F}_p$, while the hash_n be a regular cryptographic one-way function $\mathsf{hash}_n : \mathsf{B}^* \to \mathsf{B}^n$.

Let $\mathscr{T}$ be a Merkle Tree [BBb]. Each leave (element e) consists of a key-value pair (k,v). Merkle audit path $\mathsf{path}(e)$ is the shortest list of additional nodes that allows to compute the root value $\mathsf{Root}_{\mathscr{T}}$:

$$\mathsf{proof}(e) = \begin{cases} \mathsf{MP} & \text{if } \mathsf{path}(e) \to \mathsf{Root}_{\mathscr{T}} \\ \mathsf{NMP} & \text{if } \mathsf{path}(e) \to \mathsf{Root}_{\mathscr{T}'} \neq \mathsf{Root}_{\mathscr{T}} \end{cases}$$

Let $\mathsf{S}(\mathsf{priv}^*, \mathsf{pub}^*, \mathsf{rel}^*)$ be a statement that sets the list of mathematical relations rel^* between private priv^* and public pub^* signals. Let $\mathsf{D} = \langle d_p, d_v \rangle$ be a tuple of keys needed for generating and proving zk-SNARK for the statement S using the trusted procedure[Gro].

$\mathsf{prove}(d_p, \mathsf{priv}^*, \mathsf{pub}^*) \to \mathsf{P}$ is a proving construction verified by the $\mathsf{verify}(d_v, \mathsf{pub}^*, \mathsf{P}) \to \mathsf{B}$, where 0 means the proof correctness, overwise 1.

Let $\mathsf{sig\_gen}(\mathsf{message}, \mathsf{sk})$ be an EdDSA signature generates algorithm top on baby jubjub curve[BBa], operating with $\mathsf{message}$ and a private key sk. $\mathsf{sig\_ver}(\mathsf{sig}, \mathsf{PK}, \mathsf{message}) \to \mathsf{bool}$ is the signature verification algorithm that takes $\mathsf{message}$, public key PK and signature sig as inputs and returns true or false value depending on the signature correctness.

2.1 Shamir's Secret Sharing

Shamir's Secret Sharing [Sha] is a cryptographic algorithm designed to divide a secret into multiple parts, giving each participant its own unique part. To reconstruct the secret, a minimum number of parts is required. This scheme is also known as a (t,n)-threshold scheme, where t is the threshold number of parts needed to reconstruct the secret, and n is the total number of parts distributed.

2.1.1 Setup

Let $s < p$ be the secret we want to share. Lets define a random polynomial $f(x)$ of degree $t-1$ over $\mathbb{F}_p$:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1} \mod p$$

where $a_0 = s$ and $a_1, a_2, \ldots, a_{t-1}$ are randomly chosen coefficients from $\mathbb{F}_p$.

2.1.2 Shares distribution

Each of the n participants is assigned a unique, non-zero value $x_i \in \mathbb{F}_p$, and they receive the corresponding share $(x_i, f(x_i))$.

2.1.3 Secret reconstruction

At least t shares are needed to reconstruct the secret. Given t points $\{(x_1,y_1),(x_2,y_2),\ldots,(x_t,y_t)\}$, we can use Lagrange interpolation to find the polynomial $f(x)$:

$$f(x) = \sum_{j=1}^{t} L_j(x) y_j \mod p, \; L_j(x) = \prod_{\substack{1 \leq m \leq t \\ m \neq j}} \frac{x - x_m}{x_j - x_m} \mod p$$

The secret s is then $a_0 = f(0)$.

3 Rari.chat Protocol

3.1 Keypair generation

First, the user needs to generate the baby jubjub key pair $(\mathsf{sk}, \mathsf{PK})$ that will be used to confirm their actions (sending messages). These keys will be used to verify signatures in circuits more efficiently.

3.2 Creating the verifiable commitment

After generating the key pair, the user must prove the NFT ownership and add the corresponding commitment to the tree. The structure of the data in the commitment is the following:

$$\mathsf{vc} = (\mathsf{contract\_id}, \mathsf{nft\_id}, \mathsf{owner\_eoa}, \mathsf{PK}, \mathsf{timestamp})$$

Then, the user creates the transaction that initiates adding the credential to the tree and includes the following proof π:

$$\left|\begin{array}{l} \textbf{pub\_signals:} \\ \qquad \mathsf{contract\_id}, \mathsf{nft\_id}, \mathsf{owner\_eoa}, \mathsf{vc\_id}, \mathsf{T}_c \\ \textbf{priv\_signals:} \\ \qquad \mathsf{PK}, \mathsf{timestamp} \\ \textbf{circuit\_logic:} \\ \mathsf{vc\_id} = \mathsf{zkHash}(\mathsf{contract\_id}, \mathsf{nft\_id}, \mathsf{owner\_eoa}, \mathsf{PK}, \mathsf{timestamp}) \\ \qquad \wedge \\ \qquad \mathsf{timestamp} \leq \mathsf{T}_c \end{array}\right| \tag{1}$$

The transaction is signed by the user and calls the method of **Anthill** contract to add the commitment vc to the tree $\mathscr{T}$ with the index $\mathsf{vc\_id}$. The contract:

1. Verifies the TX signature according to the EOA that initiated it
2. Verifies that EOA is an owner of the declared NFT on the declared contract
3. Verifies a zero-knowledge proof π according to the statement mentioned above

Let's note that the user can define any timestamp in the credential that lower the value of the current blockchain timestamp. Digging deeper, the user passes the current time value as a public signal and proves that the timestamp inside the commitment is less than the declared value. The smart contract checks that the time value does not exceed the timestamp of the blockchain itself ($\mathsf{T_c}$ must be less than block.timestamp).

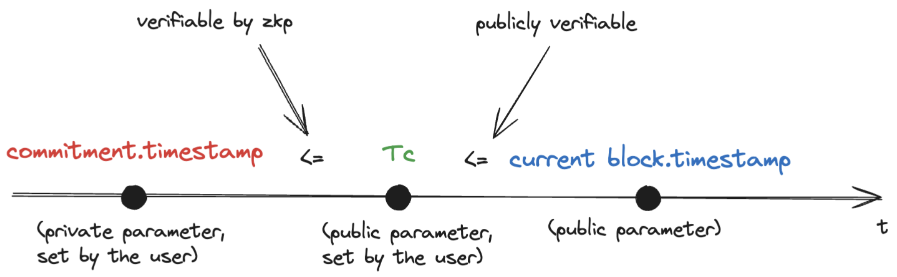

Figure 2: Verifiable commitment timestamp rules

The commitment vc is added to the tree if all verifications are performed correctly. At this stage, it is worth noting that anyone can see which user created a commitment and what specific data they linked it to (except the public key for managing the commitment) — this information is validated by the contract and, therefore, is available to any party that owns the state machine. However, all further operations with the commitment are performed in hidden form, as described in the next section.

3.3 Authentication

When users want to connect to the chat and write messages, they need to prove the validity of their credentials. The chat settings define the validity rules. These criteria include the address(es) of the NFT contract, the list of token identifiers (if we need to provide chat access only to a limited set of NFT owners, not to all of them), and expiration time bounds (the owner of the particular NFT can be changed).

Depending on the working mode, there are some modifications in the proving mechanism (the mechanism of generation of the proof for message sending), but we can formalize the approach as follows:

1. User generates the signature over the message using their commitment key:

$$\mathsf{sig\_gen}(\mathsf{message}, \mathsf{sk}) \rightarrow \mathsf{sig}$$

2. User generate the proof π for the following statement:

$$\left|\begin{array}{l} \textbf{pub\_signals:} \\ \quad \mathsf{contract\_id}, \mathsf{Root}_{\mathcal{T}}, \mathsf{message}, \mathsf{T}_t \\ \textbf{priv\_signals:} \\ \quad\quad \mathsf{vc}^*, \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)), \mathsf{sig} \\ \textbf{circuit\_logic:} \\ \quad\quad \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)) \rightarrow \mathsf{Root}_{\mathcal{T}} \\ \qquad\qquad\qquad \wedge \\ \mathsf{sig\_ver}(\mathsf{sig}, \mathsf{vc.PK}, \mathsf{message}) \rightarrow \mathsf{true} \\ \qquad\qquad\qquad \wedge \\ \qquad\quad \mathsf{vc.timestamp} > \mathsf{T_t} \end{array}\right| \tag{2}$$

3. User sends the proof to the chat service

In other words, when connecting to the chat and sending messages, the user states that they once confirmed ownership of an NFT from the collection (an existing commitment is such a confirmation), and this confirmation has not yet expired (by the relation $\mathsf{vc.timestamp} > \mathsf{T_t}$ we prove that the timestamp in the commitment exceeds the minimal threshold defined by the chat service).

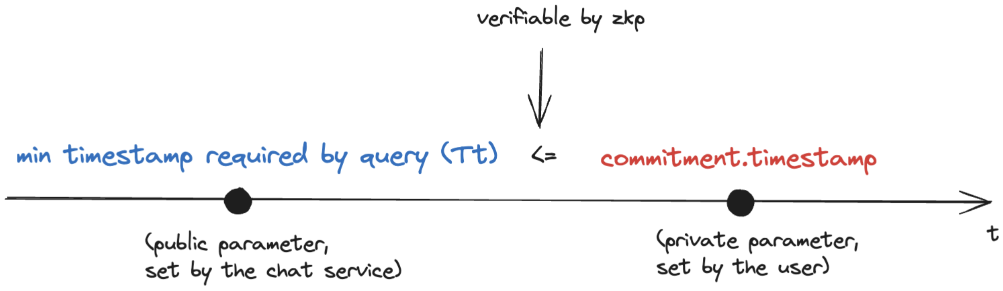
verifiable by zkp
min timestamp required by query (Tt)
<=
commitment.timestamp
(public parameter, set by the chat service)
(private parameter, set by the user)
t

Figure 3: Rules for satisfying a chat's query

The realization of the chat architecture is the topic of a separate paper; the approach proposed here should support options from centralized chat service to some federated social networks.

4 Modes

4.1 Anonymous Capybara

This mode allows the user to be fully anonymous without even connecting messages sent by the same chat participant. It's the most simple and efficient approach that allows complete decentralized chaos without limitations and prohibiting.

To be an Anonymous Capybara, the user should use the authentication approach mentioned in section 3.3.

4.2 Anonymous but Traceable Elephant

This mode presumes users to be anonymous but connects message history to the profiles they were sent from. It allows private profiles to collect a provable reputation without disclosing a particular person who stays behind.

This scheme has a small modification extending the auth method with the nullifier constructed top on the sk. It modifies the algorithm in the following way:

1. User generates the signature over the message using their commitment key:

$$\mathsf{sig\_gen}(\mathsf{message}, \mathsf{sk}) \rightarrow \mathsf{sig}$$

2. User generates the nullifier as

$$\mathsf{zkHash}(\mathsf{sk}) \rightarrow \mathsf{nullifier}$$

3. User generate the proof π for the following statement:

$$\left|\begin{array}{l} \textbf{pub\_signals:} \\ \mathsf{contract\_id}, \mathsf{Root}_{\mathscr{T}}, \mathsf{message}, \mathsf{nullifier}, \mathsf{G}, \mathsf{T_t} \\ \textbf{priv\_signals:} \\ \qquad \mathsf{vc}^*, \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)), \mathsf{sig}, \mathsf{sk} \\ \textbf{circuit\_logic:} \\ \qquad \mathsf{vc.contract\_id} = \mathsf{contract\_id} \\ \qquad\qquad\qquad \wedge \\ \qquad \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)) \rightarrow \mathsf{Root}_{\mathscr{T}} \\ \qquad\qquad\qquad \wedge \\ \qquad \mathsf{sig\_ver}(\mathsf{sig}, \mathsf{vc.PK}, \mathsf{message}) \rightarrow \mathsf{true} \\ \qquad\qquad\qquad \wedge \\ \qquad \mathsf{vc.PK} = \mathsf{sk} \cdot \mathsf{G} \\ \qquad\qquad\qquad \wedge \\ \qquad \mathsf{nullifier} = \mathsf{zkHash}(\mathsf{sk}) \\ \qquad\qquad\qquad \wedge \\ \qquad \mathsf{vc.timestamp} > \mathsf{T_t} \end{array}\right| \tag{3}$$

4. User sends the proof to the chat service

This approach allows the aggregation of all messages with the same nullifier without revealing the owner of the appropriate secret key. The best option for Elephants.

4.3 Public Alligator

It's possible to make totally public and traceable of all message senders (Alligators have nothing to hide). That's not the main idea of the protocol, but no significant modifications are required to make it possible. The scheme of relations changes to:

$$\left|\begin{array}{l} \textbf{pub\_signals:} \\ \mathsf{contract\_id}, \mathsf{Root}_{\mathscr{T}}, \mathsf{message}, \mathsf{PK}, \mathsf{T_t} \\ \textbf{priv\_signals:} \\ \qquad \mathsf{vc}^*, \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)), \mathsf{sig} \\ \textbf{circuit\_logic:} \\ \qquad \mathsf{vc.contract\_id} = \mathsf{contract\_id} \\ \qquad\qquad\qquad \wedge \\ \qquad \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)) \to \mathsf{Root}_{\mathscr{T}} \\ \qquad\qquad\qquad \wedge \\ \qquad\qquad \mathsf{PK} = \mathsf{vc.PK} \\ \qquad\qquad\qquad \wedge \\ \quad \mathsf{sig\_ver}(\mathsf{sig}, \mathsf{PK}, \mathsf{message}) \to \mathsf{true} \\ \qquad\qquad\qquad \wedge \\ \qquad\quad \mathsf{vc.timestamp} > \mathsf{T_t} \end{array}\right| \tag{4}$$

It's not so difficult, right? This mode doesn't allow the non-community participant to send the message, so full verifiability and transparency are met. Using an EOA identifier instead of PK is also possible. It's up to you.

4.4 Confidential Hyena

All previous approaches presumed messages transfer in the open form, making them publicly auditable. This approach doesn't suit private organizations, where the message content must only be available to community members.

For organizing confidential messaging, the most efficient approach is for the first chat participant to generate the secret key and then encrypt it using other users' public keys (asymmetric encryption). Users can use new keypairs for that and authenticate them using the signature generated by the commitment's key.

All processes with expiration and revocation of the encryption key depend on the chat members. They have enough to use different governance protocols that are managed by their commitment keys.

4.5 Rate-Limiting Penguin

Sometimes, it makes sense to ban spammers. But again, with no administrators, only using defined chat rules and cryptography. We can use rate-limiting nullifiers for this purpose[Bla]. It extends our verifiable commitment with the new field

$$\mathsf{sk\_null} = \mathsf{zkHash}(\mathsf{sk})$$

At the same time, let's define the linear polynomial $\mathsf{f(x) = ax + b}$, meaning the secret b can be reconstructed by having evaluations in only two points.

When the user wants to send a message, it should calculate the following point:

$$\mathsf{x} = \mathsf{zkHash}(\mathsf{message}), \;\; \mathsf{y} = \mathsf{f}(\mathsf{x})$$

Then, the user sends this share with its validness proof π.

$$\left|\begin{array}{l}\textbf{pub\_signals:}\\ \quad \mathsf{contract\_id}, \mathsf{Root}_{\mathscr{T}}, \mathsf{message}, \mathsf{topic}, \mathsf{y}, \mathsf{T_t}\\ \textbf{priv\_signals:}\\ \qquad \mathsf{vc}^*, \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)), \mathsf{sig}, \mathsf{sk}\\ \textbf{circuit\_logic:}\\ \qquad \mathsf{vc.contract\_id} = \mathsf{contract\_id}\\ \qquad \wedge\\ \qquad \mathsf{path}(\mathsf{zkHash}(\mathsf{vc}^*)) \to \mathsf{Root}_{\mathscr{T}}\\ \qquad \wedge\\ \qquad \mathsf{sig\_ver}(\mathsf{sig}, \mathsf{vc.PK}, \mathsf{message}) \to \mathsf{true}\\ \qquad \wedge\\ \mathsf{y} = (\mathsf{zkHash}(\mathsf{topic}, \mathsf{sk})) \cdot \mathsf{zkHash}(\mathsf{message}) + \mathsf{sk}\\ \qquad \wedge\\ \qquad \mathsf{sk\_null} = \mathsf{zkHash}(\mathsf{sk})\\ \qquad \wedge\\ \qquad \mathsf{vc.timestamp} > \mathsf{T_t}\end{array}\right| \tag{5}$$

If the user wants to send another message for the same topic, its secret key will be corrupted. For example, we have two messages m_1, m_2 and corresponding shares:

$$x_1 = \mathsf{zkHash}(m1), y_1 = \mathsf{zkHash}(\mathsf{topic}, \mathsf{sk}) \cdot \mathsf{zkHash}(m1) + \mathsf{sk}$$

$$x_2 = \mathsf{zkHash}(m2), y_2 = \mathsf{zkHash}(\mathsf{topic}, \mathsf{sk}) \cdot \mathsf{zkHash}(m2) + \mathsf{sk}$$

Everyone can reconstruct the polynomial by:

$$f(x) = y_1 \cdot \frac{x - x_2}{x_1 - x_2} + y_2 \cdot \frac{x - x_1}{x_2 - x_1}$$

Wrapping $\mathsf{zkHash}(\mathsf{topic}, \mathsf{sk})$ to t we receive:

$$f(x) = (t \cdot x_1 + \mathsf{sk}) \cdot \frac{x - x_2}{x_1 - x_2} + (t \cdot x_2 + \mathsf{sk}) \cdot \frac{x - x_1}{x_2 - x_1}$$

$$f(x) = \frac{tx_1x - tx_1x_2}{x_1 - x_2} + \mathsf{sk} \cdot \frac{x - x_2}{x_1 - x_2} + \frac{tx_2x - tx_1x_2}{x_2 - x_1} + \mathsf{sk} \cdot \frac{x - x_1}{x_2 - x_1}$$

$$f(x) = \mathsf{sk} \cdot (\frac{x - x_2}{x_1 - x_2} - \frac{x - x_1}{x_1 - x_2}) + t \cdot (\frac{x_1x - x_1x_2}{x_1 - x_2} - \frac{x_2x - x_1x_2}{x_1 - x_2})$$

$$f(x) = \mathsf{sk} + t \cdot x$$

So we see that the polynomial was reconstructed correctly with revealing the user's secret key. This approach doesn't limit the user's actions but allows to steal an identity if the user violates the chat rules.

5 Implementation Notes

1. Depending on the implementation of the time-checking logic on the contracts and circuits, different strategies for user eligibility can be considered. In this paper, the static approach is described when, in the public domain, such as the Ethereum network, on the contract, a specific timestamp is statically stored, which specifies after which point the credentials are valid. Also, it is possible to define the validity of credentials for a specific period, for example, one month.

2. During the registration period, the Sparse Merkle Tree is used. Therefore, adding a tree, both index and value is required. As index the $\mathsf{vc\_id}$ is used, and value is constructed as follows: $(\mathsf{zkHash}(\mathsf{contract\_id}, \mathsf{nft\_id}, \mathsf{owner\_eoa}))$ This allows for possible query proofs, where the participant can prove that some of these values are in the tree without revealing them. This can be used to filter out desired participants based on the data in the value.

3. As we mentioned at the start of the article — NFT isn't the single artifact the user can be connected to. There can be verifiable commitments connecting to certain balances, network activity, attestations, and other credentials and commitments.

6 Summary

This paper described how private chat can be built on top of the public community with deterministic criteria. This proposal extends and clarifies the approach of the Rarimo protocol for building a social forest [Rar]. Anthill contract is an instance of a social tree focusing on chat purposes — it reuses public on-chain ownership rights, converting them to leaves of the social anonymous tree.

Awwwahahahweheehwhehhe! (Chewbacca's roar)

References

[BBa] Jordi Baylina1 and Marta Bell´es. *EdDSA For Baby Jubjub Elliptic Curve with MiMC-7 Hash*. URL: https://iden3-docs.readthedocs.io/en/latest/_downloads/a04267077fb3fdbf2b608e014706e004/Ed-DSA.pdf.

[BBb] Jordi Baylina1 and Marta Bell´es. *Sparse Merkle Trees*. URL: https://docs.iden3.io/publications/pdfs/Merkle-Tree.pdf.

[Bla] Blagoj. *Rate Limiting Nullifier: A spam-protection mechanism for anonymous environments*. URL: https://medium.com/privacy-scaling-explorations/rate-limiting-nullifier-a-spam-protection-mechanism-for-anonymous-environments-bbe4006a57d.

[Gro] Jens Groth. *Short Pairing-based Non-interactive Zero-Knowledge Arguments*. URL: https://www.iacr.org/archive/asiacrypt2010/6477323/6477323.pdf.

[Rar] Rarimo. *Rarimo: A privacy-first (zk) social protocol*. URL: https://docs.rarimo.com/files/Rarimo_whitepaper_v3.pdf.

[Sha] Adi Shamir. *How to share a secret*. URL: https://dl.acm.org/doi/10.1145/359168.359176.