AnonRep: Towards Tracking-Resistant Anonymous Reputation

Ennan Zhai[†] David Isaac Wolinsky[‡] Ruichuan Chen[§]
Ewa Syta[†] Chao Teng[‡] Bryan Ford[*]
[†]*Yale University* [‡]*Facebook Inc.* [§]*Nokia Bell Labs* [*]*EPFL*

Abstract

Reputation systems help users evaluate information quality and incentivize civilized behavior, often by tallying feedback from other users such as "likes" or votes and linking these scores to a user's long-term identity. This identity linkage enables user tracking, however, and appears at odds with strong privacy or anonymity. This paper presents AnonRep, a practical anonymous reputation system offering the benefits of reputation without enabling long-term tracking. AnonRep users anonymously post messages, which they can verifiably tag with their reputation scores without leaking sensitive information. AnonRep reliably tallies other users' feedback (*e.g.*, likes or votes) without revealing the user's identity or exact score to anyone, while maintaining security against score tampering or duplicate feedback. A working prototype demonstrates that AnonRep scales linearly with the number of participating users. Experiments show that the latency for a user to generate anonymous feedback is less than ten seconds in a 10,000-user anonymity group.

1 Introduction

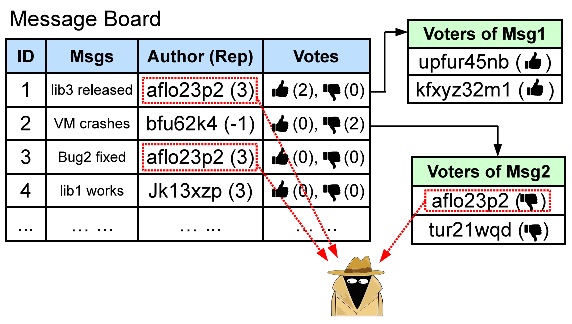
Online services such as eBay, Yelp, and Stack Overflow employ reputation systems to evaluate information quality and filter spam. In Yelp, for example, users post messages (*e.g.*, reviews), and offer feedback on other users' posts (*e.g.*, votes) based on perceived utility. User reputations increase or decrease based on this feedback, and reputation affects how widely a user's future posts are viewed. This long-term linkage between user behavior and reputation, however, can quickly de-anonymize users wishing to hide their true identities [4, 23, 28, 31]. For example, Minkus *et al.* [28] revealed eBay users' sensitive purchase histories by analyzing only pseudonyms' transactions and feedback. As privacy has become a major concern for online users, we raise the question: can we combine the benefits of reputation with the privacy afforded by fully anonymous, unlinkable messaging? Can we build an *anonymous reputation system*?

In an anonymous reputation system, no entity – either users or servers implementing the reputation system – should be able to link posted messages and feedback to any user identity. Maintaining reputation without identity in principle offers the benefits of reputation with unprecedented user privacy [5]. Achieving this goal is challenging, however, since the requirement to associate users with their historical activities seems to preclude anonymity [38]. Establishing reputation while maintaining unlinkability of activities appears to be a paradox.

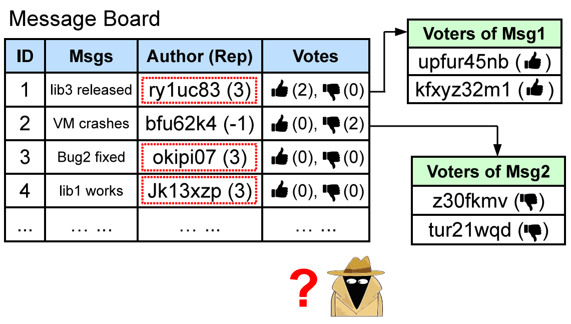
Prior efforts have addressed this problem [2, 5, 7, 15], but none have yet proven practical or sufficiently general for realistic deployment. For example, Androulaki *et al.* [2] proposed blind signatures for anonymous peer-to-peer reputation transactions, but this protocol relies on a centralized honest entity and cannot support negative feedback (*e.g.*, about trolls or otherwise misbehaving users). Similarly, Bethencourt *et al.* [5] proposed to use zero-knowledge proofs to construct signatures of reputation, thus keeping unlinkability of users' historical behaviors. This approach supports limited reputation algorithms, however, and is computationally expensive.

This paper presents *AnonRep*, the first practical anonymous reputation system supporting diverse reputation schemes without leaking sensitive information about users' long-term identities or historical activities. AnonRep represents a novel integration of known cryptographic primitives – verifiable shuffles [32], linkable ring signatures [26], and homomorphic crypto [17] – in a multi-provider deployment architecture. AnonRep builds on the anytrust model [41], like Dissent [42] and Vuvuzela [39], for scalability and robustness to client churn. An AnonRep group consists of a potentially large set of client nodes representing users, and a smaller set of third-party commodity servers implementing the anonymous reputation service. Each client trusts that at least one server is honest and not colluding with the others, but the client need not know which server to trust.

AnonRep operates in a series of *message-and-feedback* rounds. Each round might in practice last a few minutes, hours, or even a full day, depending on the application scenario. At the beginning of each round, the servers maintain a database containing all clients' long-term identities and their respective encrypted reputation scores. During each round, the servers successively run a scheduling protocol based on verifiable shuffles [32], which transforms the reputation list into an anonymously permuted list consisting of a one-time pseudonym for each client and an associated plaintext reputation score. While exact reputation scores could themselves link clients across rounds, AnonRep allows users to reveal only approximate reputations (§6). AnonRep's scheduling protocol is decentralized: neither servers nor clients


Message Board
ID | Msgs | Author (Rep) | Votes
1 | lib3 released | aflo23p2 (3) | (2), (0)
2 | VM crashes | bfu62k4 (-1) | (0), (2)
3 | Bug2 fixed | aflo23p2 (3) | (0), (0)
4 | lib1 works | Jk13xzp (3) | (0), (0)
... | ... ... | ... ... | ... ...
Voters of Msg1
upfur45nb ( )
kfxyz32m1 ( )
Voters of Msg2
aflo23p2 ( )
tur21wqd ( )

(a) A typical message board equipped with conventional reputation technique which potentially has linkability issues.


Message Board
ID | Msgs | Author (Rep) | Votes
1 | lib3 released | ry1uc83 (3) | (2), (0)
2 | VM crashes | bfu62k4 (-1) | (0), (2)
3 | Bug2 fixed | okipi07 (3) | (0), (0)
4 | lib1 works | Jk13xzp (3) | (0), (0)
... | ... ... | ... ... | ... ...
Voters of Msg1
upfur45nb ( )
kfxyz32m1 ( )
Voters of Msg2
z30fkmv ( )
tur21wqd ( )
?

(b) The same message board equipped with AnonRep. Adversary cannot link different activities to any specific identity.

Figure 1: Motivating example. We use a typical message board with different reputation technique. The sample message board evaluates the quality of posted messages based on the reputation of these messages' authors. Feedback is represented as votes.

(other than the owner) can link one-time pseudonyms or reputations to long-term identities.

Clients then post messages anonymously using these one-time pseudonyms. The servers can associate these messages with their corresponding reputation scores without learning clients' sensitive information. Each client may then provide feedback (*e.g.*, votes) on other clients' posted messages. Each vote is signed by a linkable ring signature [26], enabling the servers to verify that each client votes only once without revealing which client submitted each vote. This design enables the servers to tally positive and negative feedback without linking this feedback with long-term identities.

Finally, the servers tally the feedback received for each one-time pseudonym, update the reputation score, and then perform a "reverse scheduling" to transform these one-time pseudonyms and their updated reputation scores back to the original long-term identities and their encrypted updated reputation scores.

We have implemented an AnonRep prototype in Go. Experimental results show that the AnonRep server scales linearly with the number of clients. With a 10,000-client anonymity group, for example, each server's computational cost is about one minute per round. The time required for a client to construct an anonymous vote to provide feedback is less than ten seconds in a 10,000-client anonymity group. While the current prototype has many limitations and would benefit from further development, we nevertheless believe that AnonRep represents a significant step towards building a practical anonymous reputation system for realistic online services.

In summary, this paper makes the following contributions. First, we propose the first practical anonymous reputation system, AnonRep, offering the benefits of reputation while maintaining the unlinkability and anonymity of users' historical activities. Second, we provide a fully-functional open-source prototype illustrating AnonRep's functionality and practicality.

2 Motivation and Challenges

This section first presents a simple but illustrative example to motivate AnonRep's goals (§2.1), then discusses key technical challenges (§2.2).

2.1 Motivating Example

Figure 1a shows a typical reputation system, which utilizes a message board to evaluate information quality and filter out spam. The message board maintains a reputation score for each client. Each client has an identity or pseudonym, which remains fixed throughout the client's lifetime. Suppose a client, Alice, posts a message on the message board. The message board associates the message with Alice's reputation score, which other users or content curation algorithms might use to determine how widely Alice's message is seen. Other clients who view Alice's message can then give positive or negative feedback to express subjective opinions on the quality of Alice's message. Based on this feedback, the message board updates Alice's reputation, enabling Alice to post new messages with the updated reputation. Precisely how user feedback affects clients' reputation scores varies depending on the specific reputation algorithm.

Message board reputation systems of this kind have been widely employed by many online services, *e.g.*, eBay, Yelp, and Stack Overflow. However, in such a system, each client's reputation score is associated with either her real identity or a long-lived pseudonym. As a result, even with pseudonyms, a client's historical activities can be easily tracked and linked, leaking sensitive information. For example, in Figure 1a, an adversary can observe that the first and third messages are posted by the same pseudonym `aflo23p2`, and the second message is posted by another pseudonym `dged2p`. The adversary can also learn the voters of each message. For example, the client with pseudonym `aflo23p2` casts a negative vote to the second message. Even in the absence of clients' real identities, Minkus *et al.* [28] have successfully exposed eBay clients' sensitive purchase histories and feedback by analyzing only pseudonyms' transactions.

The goal of this paper is to design a practical anonymous reputation system providing the utility of a reputation system, while hiding clients' sensitive information – including the linkage between messages posted by the same user. With AnonRep, as shown in Figure 1b, a client appears as different one-time pseudonym every time the client posts a message. These one-time pseudonyms avoid revealing information that can link any clients' messages, reputation scores, or votes across posting rounds. Meanwhile, AnonRep can still privately update each client's reputation score without any participants learning sensitive information.

While content-based attacks such as stylometry [30] could still link one user's message across rounds, these techniques are uncertain and prone to false positives, especially operating on short messages (*e.g.*, tweets). Regardless, AnonRep's goal is not to address content-based linkage risks, but to ensure that feedback and reputation management does not leak any *more* sensitive information beyond what the user-provided content itself might.

2.2 Technical Challenges

We face two main technical challenges to build a practical anonymous reputation system.

Challenge 1: Protecting the association between reputation and identity. The calculation of a user's reputation score relies on the historical activities associated with this user's identity. It seems that maintaining this reputation would preclude any possibility of identity anonymity [2, 5]. Two straightforward solutions are 1) to introduce a trusted third party that updates reputation scores for clients, or 2) to use secure multi-party computation (SMPC) [44] to update reputation scores privately. Unfortunately, the former solution offers weak security by requiring every user to trust a third party, while the latter solution is slow and computation-intensive and has not proven scalable in practice. Therefore, keeping the association between reputation and user identity private presents a significant challenge.

Challenge 2: Detection of misbehavior. A centralized reputation system can readily enforce well-defined rules for handling reputation and feedback fairly – such as "one client, one vote" – because a trusted entity can see all clients' activities and enforce these rules. In a decentralized anonymous reputation system without a trusted party, however, clients might misbehave, *e.g.*, by casting multiple votes on the same message to amplify the user's feedback unfairly. Such misbehavior is non-trivial to detect in an anonymous reputation system [5].

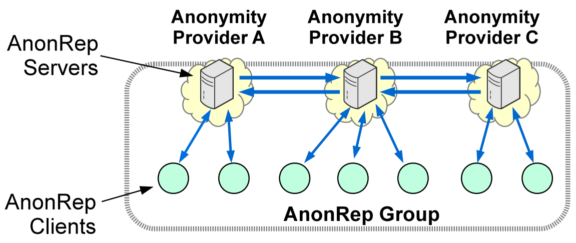


Figure 2: AnonRep's multi-provider deployment model, and its basic communication topology. In an AnonRep group, each client communicates with a single upstream server, while each server can communicate with all other servers.

3 AnonRep Overview

In this section, we first sketch the architecture of AnonRep in §3.1. Then, we present our assumptions and threat model in §3.2. Finally, we give our security goals in §3.3.

3.1 Architecture

As illustrated in Figure 2, AnonRep relies on a *multi-provider* model to achieve scalability and resilience to link failures [16, 41, 42]. A typical AnonRep group includes two types of members: 1) a potentially large number of unreliable *client* nodes representing individual users, and 2) a small number of *servers*, which are assumed to be highly available and well-provisioned.

In practice, each server in an AnonRep group should be operated independently (*i.e.*, each managed by a separate operator) to limit the risks of all servers being compromised or colluding against clients.

Each client directly communicates with at least one *upstream server*, while each server can communicate with any other servers (see Figure 2). Such a communication topology reduces the communication and computational overhead at the clients, and enables the system to tolerate client churn [42]. More specifically, each client does not need to know which other clients are online while posting messages or feedback to the upstream server.

3.2 Threat Model and Assumptions

In an AnonRep group, clients need *not* assume any particular server is trustworthy, and they need not even trust their respective upstream servers. Instead, we assume the anytrust model, *i.e.*, each client trusts only that there exists *at least one* honest server without knowing which this server is [16, 41, 42]. An AnonRep group member (server or client) is *honest* if the member follows the specified protocol faithfully and does not collude with or leak sensitive information to other group members. A member is *dishonest* (or malicious) otherwise.

A malicious client may wish to link or track sensitive information such as reputation scores, messages, and feedback to specific victim clients. Multiple malicious clients may collude with each other.

A malicious server may refuse to service honest clients, but such refusal should not compromise clients' anonymity. Moreover, a malicious server may try to tamper with clients' reputation scores, and even collude with malicious clients.

We assume that public and symmetric key encryptions, key-exchange mechanisms, signature schemes and hash functions are all correctly used. We also assume that public keys of AnonRep servers and clients are publicly available. We assume the network connections between clients and servers are established over anonymous communication channels (*e.g.*, Tor [19], or traffic analysis resistant networks like Dissent [42] and Vuvuzela [39]).

3.3 Security Goals

Anonymity. The main goal of AnonRep is to achieve *anonymity* for its clients in face of a strong adversary, *i.e.*, malicious servers and clients as defined above. In AnonRep, anonymity means not only the privacy of each client's data such as profile and IP location, but also the unlinkability of clients' historical activities. No AnonRep group member should be able to link a specific client's sensitive information such as posted messages, reputation scores, or feedback to the client's identity.

AnonRep provides the above anonymity guarantee among the set of honest group members, that is, members who faithfully follow our protocol. AnonRep does not attempt to provide anonymity to malicious group members as they can may collude with others and reveal their identities and association to their messages themselves.

Other goals. Besides anonymity, AnonRep should ensure that the misbehaviors of malicious group members are detectable. In addition, AnonRep should balance the trade-offs between practicality and security. The more clients an anonymity group contains, the stronger anonymity it can offer but at the cost of higher overhead.

Non-goals. Like many prior reputation systems, AnonRep is not designed against the Sybil attack where attackers generate a large number of fake clients to manipulate the reputation of honest clients. How to make AnonRep resistant to the Sybil attack is out of the scope of this paper. In addition, AnonRep is not resilient to network-level Denial-of-Service (DoS) attacks where attackers, for instance, could target the AnonRep servers. The servers, however, are assumed to be highly available. Nevertheless, some well-known defenses (*e.g.*, server provisioning and proof-of-work challenges) could be deployed to mitigate DoS attacks.

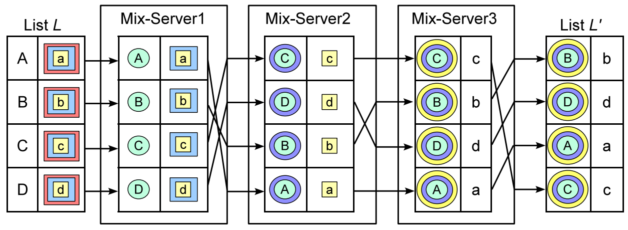


Figure 3: A simple Mix-Net example with three mix-servers. Each mix-server performs the verifiable shuffle protocol. The input is a two-column list with four entries. Each mix-server 1) encrypts and decrypts the elements in the first and second columns with different keys, respectively; 2) permutes the order of entries in the list; and 3) sends the permuted list to the next mix-server.

4 Cryptographic Building Blocks

Before we elaborate on the design details of AnonRep in §5, we first describe several cryptographic techniques that AnonRep builds upon.

4.1 Mix-Net and Verifiable Shuffles

Mix-Net [12] is a decentralized cryptographic protocol that creates hard-to-trace communications by using a chain of servers (called *mixes or mix-servers*) which take in a list of objects, shuffle them, and then output them in random order. As shown in Figure 3, such a primitive ensures unlinkability between the source and the destination of the list.

The shuffle phase in a Mix-Net protocol contains encryption, decryption and permutation operations. For example in Figure 3, each mix-server adds one ciphertext layer on each element in the first column of the received list, strips one ciphertext layer from each element in the second column, then permutes entries in the list, and finally sends the resulting list to the next mix-server.

In order to ensure the correctness of the operations performed in shuffle phases, many verifiable shuffle protocols have been proposed [25, 32, 33]. In a typical verifiable shuffle protocol, besides performing the shuffle operartions, each mix-server generates a zero-knowledge proof, which can be used by any observer (*i.e.*, verifier) to check whether the mix-server correctly performed its shuffle.

Here, we detail a verifiable shuffle primitive `Shuffle`$(g_i, \vec{L}_i, z_i, e_i)$ that we use in the AnonRep design (§5.3). Figure 3 presents an example where three mix-servers successively run this primitive. In a typical `Shuffle` primitive execution, each mix-server i performs the following four operations.

1. Use the public key e_i to encrypt each element in the first column of list $\vec{L}_i$; use the private key z_i to strip one ciphertext layer from each element in the second column of the list $\vec{L}_i$;

2. Permute entries in the resulting list, producing $\vec{L_{i+1}}$;
3. Use the public key e_i to encrypt the received generator g_i, *i.e.*, $g_{i+1} = {g_i}^{e_i} \bmod p$; and
4. Generate a (zero-knowledge) proof f_i attesting that the above operations are correctly performed.

After running the primitive, the mix-server i sends $\vec{L_{i+1}}$, g_{i+1} with the proof f_i to the next mix-server $i+1$.

4.2 Linkable Ring Signatures

Liu *et al.* proposed linkable ring signatures [26], a variant of traditional ring signatures [37]. A linkable ring signature allows any of n group members to produce a ring signature on some message such that no one knows *which* group member produced the signature but all signatures from the same member can be linked together.

Each group member holds a public/private key pair (PK_i, SK_i). Member i can compute a ring signature σ on a message m, on input $(m, SK_i, PK_1, ..., PK_n)$. Anyone can check the validity of a ring signature given (σ, m) and the public key list $L = \{PK_1, ..., PK_n\}$ of all group members; however, nobody knows who signed the message m. It is hard for anyone to create a valid ring signature on any message on behalf of some group without knowing at least one secret key of a member of this group. Another important property of linkable ring signature is *linkability*: given any two signatures, a verifier can determine whether they were produced by the same member in the group but still without learning the specific member's identity.

In particular, linkable ring signature consists of the following four steps [26].

Initialization step: Each member i $(i = 1, ..., n)$ has a public key Y_i, and private key y_i, where $(Y_i = g^{y_i})$. Each member knows the list of n members' public keys $L = \{Y_1, ..., Y_n\}$, and a public hash function $H(\cdot)$.

Signature generation step: Suppose a member i, called a signer, wants to use linkable ring signature scheme to sign a message m. She first needs to compute the linkability tag $t = H(L)$. Then, i runs the primitive `Sign`(m, L, y_i, t) to get m's linkable ring signature $\sigma(m)$. Finally, i sends m and $\sigma(m)$ to the verifier.

Verification step: The verifier receives the message m and the signature $\sigma(m)$. He knows the public key list L. The verifier runs `Verify`(t, m, L, $\sigma(m)$) to check whether $\sigma(m)$ is produced by one of the members in the group specified by L.

Linkability checking step: Given two signatures $\sigma'(m')$ and $\sigma''(m'')$, the verifier can check whether the two signatures are from the same signer by running `Check`($\sigma'(m')$, $\sigma''(m'')$). Because each linkable ring signature is generated based on a linkability tag $t = H(L)$ and the private key of signer y_i, if the two signatures are from the same signer, the verifier would successfully confirm this fact.

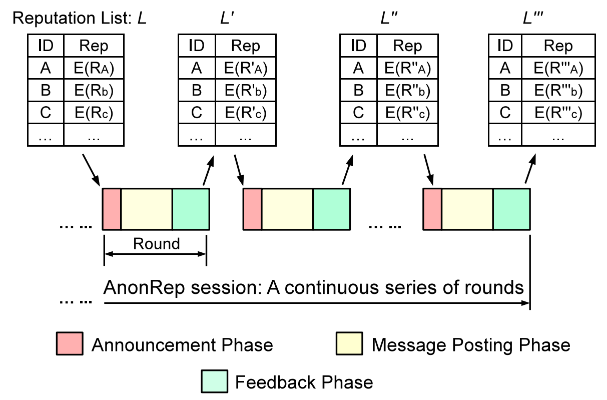


Figure 4: AnonRep's session, rounds and phases. An AnonRep session contains a continuous series of message-and-feedback rounds. Each round has three phases: 1) announcement phase, 2) message posting phase, and 3) feedback phase. All the online (or available) members, including servers and clients, synchronously participate in these rounds.

5 AnonRep System Design

This section details AnonRep's basic design (§5.1-§5.5), followed by practical considerations (§5.6).

5.1 AnonRep Workflow

A typical AnonRep session, as shown in Figure 4, consists of a series of *message-and-feedback* rounds. All online AnonRep group members (including servers and clients) synchronously participate in these rounds. In practice, the duration of each round may be a few hours or even one day, depending on the application scenario.

The input to each round is a two-column *reputation list*. The first column records the long-term identity of each registered client, and the second column is the client's reputation score encrypted by all servers (see §5.2). A client's long-term identity is her public key, which corresponds to a private key maintained by the client herself. The output of each round is a similar reputation list with updated clients' reputation scores. The output list of one round serves as input to the next round, as shown in Figure 4. Any newcomer client can participate in the AnonRep session after she completes the registration process (details in §5.2).

Each round consists of three phases. The duration of each phase may be significantly different.

- **Announcement phase:** Servers run *scheduling protocols* to assign a one-time pseudonym to each client. Only the client knows and can use the one-time pseudonym assigned to her (§5.3).
- **Message posting phase:** Clients anonymously post messages using the assigned one-time pseudonyms, and the upstream servers associate the corresponding reputation scores with the messages, without learning clients' long-term identities (§5.4).

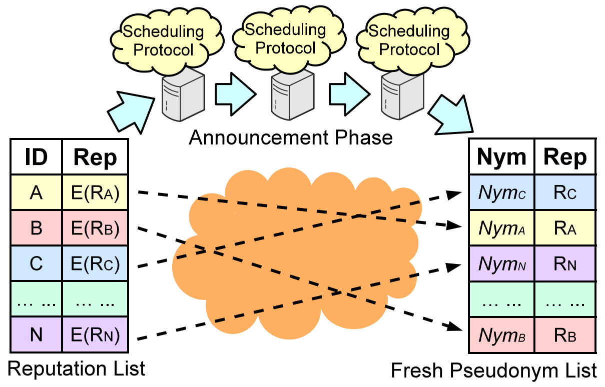


Figure 5: AnonRep's announcement phase via scheduling protocols. Each entry in the reputation list records some client's long-term identity and the ciphertext of her reputation score $E(R_i)$, which has been encrypted by all the servers. On the other side, each entry in the fresh pseudonym list records some client's one-time pseudonym for this round and the plaintext of her reputation score R_i.

- **Feedback phase:** Clients anonymously provide feedback to posted messages. At the end of this phase, servers update each one-time pseudonym's reputation score based on the received feedback, and then update the long-term scores in the reputation list by running "reverse scheduling" protocols (§5.5).

With the system overview in place, we now describe the details of the system design.

5.2 Client Registration

Any newcomer client who wants to use AnonRep needs to register with the servers.

Specifically, each new client i *first* generates a key-pair $\langle Y_i = g^{y_i}, y_i \rangle$, where g is a generator shared among servers and clients. Here, Y_i and y_i are the client i's long-term public and private keys, respectively. *Then*, the client i uploads the public key Y_i to a randomly selected AnonRep server S_j, called the client i's upstream server. *Next*, S_j creates an initial reputation score r_i for the client i, and then encrypts this initial reputation score by using its public key Z_j, and sends the ciphertext to the next server, *i.e.*, S_{j+1}. All servers use their public keys to encrypt this initial reputation in sequence. Once the client i's upstream server S_j receives the client's reputation score $E(r_i)$ which has been encrypted by all servers, S_j creates a tuple $\langle Y_i, E(r_i) \rangle$, and broadcasts this tuple to the servers. *Finally*, each AnonRep server appends this tuple to a local reputation list.

5.3 Announcement Phase

As shown in Figure 4, the announcement phase is the first phase of a message-and-feedback round. In a typical announcement phase, as shown in Figure 5, the servers take the reputation list as input, and successively perform *scheduling protocols* to generate a *fresh pseudonym list*, in order to enable each client to have a "temporary identity" to post message and provide feedback in the following phases. The entries in the fresh pseudonym list are in a permuted order. Each entry corresponds to one client, and contains a one-time pseudonym as well as the plaintext reputation score for this client. Because the announcement phase is executed by multiple independent servers, no server can link the original reputation list to the generated fresh pseudonym list as long as at least one server does not collude with others. Moreover, each client only knows her own entry in the fresh pseudonym list, and cannot learn the associations between other clients and their pseudonyms.

At the beginning of an announcement phase, each server j locally maintains an ephemeral secret e_j (different in each round), a public generator g, and its own private key z_j, which corresponds to a public key, Z_j, used to encrypt the new client's reputation score during the client registration. The servers perform the scheduling protocol (shown in Algorithm 1), transforming the input reputation list $\vec{L}$ and the public generator g into the fresh pseudonym list $\vec{pk}$ and the final generator $g_{m+1} = g^{e_1 \dots e_m}$.

After all the servers finish the scheduling protocol, each client learns the fresh pseudonym list, *i.e.*, $\vec{pk}$, and the final generator, g_{m+1}, from her upstream server. Then, each client i is able to compute and find her own one-time pseudonym, $pk_{\pi(i)}$, in $\vec{pk}$ by: $pk_{\pi(i)} = {g_{m+1}}^{y_i}$, where y_i is the client i's private key (corresponding to her long-term public key Y_i, defined in §5.2), and $\pi(i)$ denotes the location of client i's one-time pseudonym in the fresh pseudonym list $\vec{pk}$.

Based on the working principle of verifiable shuffle primitive (see §4.1 and Algorithm 1), client i's long-term pseudonym key Y_i is encrypted by all the servers, *i.e.*, ${Y_i}^{e_1 \dots e_m} = pk_{\pi(i)}$. Because $Y_i = g^{y_i}$ (see §5.2), we have:

$$pk_{\pi(i)} = {Y_i}^{e_1 \dots e_m} = (g^{y_i})^{e_1 \dots e_m} = (g^{e_1 \dots e_m})^{y_i} = {g_{m+1}}^{y_i}$$

Only the client i learns $pk_{\pi(i)}$, since only i knows her private key y_i. In the current round, each client i is assigned a new public/private key-pair $\langle pk_{\pi(i)}, y_i \rangle$ based on the final generator g_{m+1}: $pk_{\pi(i)} = {g_{m+1}}^{y_i}$. Each client can use this one-time pseudonym to post messages and provide feedback later, without leaking the long-term identity.

The scheduling protocol uses verifiable shuffles [32]. Therefore, during the announcement, each server computes and attaches a zero-knowledge proof of correctness to each "intermediate list" sent to its successive server. This step ensures that if a server misbehaves, it will be detectable by other servers.

Algorithm 1 Scheduling protocol.

All members (including clients and servers) know the reputation list $\vec{L}$ and the public generator g. Each server j knows its private key z_j and an ephemeral secret e_j, and each client i knows her public key $Y_i = g^{y_i}$, where y_i is i's private key.

1. The first server S_1 takes the reputation list $\vec{L_1} = \vec{L}$ as input, and performs the verifiable shuffle primitive, `Shuffle(`$g = g_1$, $\vec{L_1}$, z_1, e_1`)`, to obtain outputs: g_2, $\vec{L_2}$ and a proof. Then, S_1 sends $\vec{L_2}$, g_2 with the proof to the next server S_2.
2. Each server S_j ($j = 2, ..., m$) successively runs the same verifiable shuffle primitive as S_1. Namely, S_j runs `Shuffle(`g_j, $\vec{L_j}$, z_j, e_j`)` to obtain g_{j+1}, $\vec{L_{j+1}}$ and a proof about the correctness. Then, S_j sends $\vec{L_{j+1}}$, g_{j+1} with the proof to the next server S_{j+1}.
3. After the final server S_m performs the shuffle primitive, S_m outputs the fresh pseudonym list, $\vec{pk} = \vec{L_{m+1}}$, and the final generator, g_{m+1}, which has been encrypted by all the servers. *Note:* Reputation scores in $\vec{pk}$ are plaintexts now, since all the ciphertext layers have already been decrypted by all the servers. Then, servers distribute all results ($\vec{L_j}, g_j \forall j \in m$ with proofs) to all the other servers.
4. Each server k verifies each $\vec{L_j}$, g_j, and proofs. If they match, server k transmits a signature, sig_k, of $\vec{pk}$ to all other servers.
5. Upon collecting a signature sig_j from every other server j, servers distribute $\vec{pk}$, g_{m+1}, and $sig_j \forall j \in m$ to their clients.

5.4 Message Posting Phase

After the announcement phase, group members enter the message posting phase. A client's message posting process is in principle a public key signature verification procedure. A given client signs her message using her private key, and then submits it to her upstream server. The server verifies the signature using a public key from the fresh pseudonym list. If the verification succeeds, the server posts the message and associates the corresponding reputation score with the message. This works, because each client has been assigned a "temporary" public-private pair, $\langle pk_{\pi(i)}, y_i \rangle$ based on g_{m+1} in the announcement phase (§5.3).

AnonRep uses ElGamal signature scheme for message verification. Suppose some client i wants to post a message m. She first chooses a random k so that $1 < k < p-1$ and $\gcd(k, p-1) = 1$. Then, the client computes $r = {g_{m+1}}^k \bmod (p-1)$, where g_{m+1} is the final generator obtained from the announcement phase. With r in hand, the client computes $s = (H(m) - y_i \cdot r)k^{-1} \bmod (p-1)$, where y_i is i's private key and $H(m)$ is the message m's hash value. Finally, the client sends her upstream server the signature $\sigma = (r, s)$ and message m through anonymous communication tool (*e.g.*, Tor [19] or Vuvuzela [39]), which can hide the client's local information.

After receiving the message and signature pair (m, σ), the upstream server verifies it by checking ${g_{m+1}}^{H(m)} \overset{?}{=} pk_{\pi(i)}^r r^s$. If the verification is correct, then the server concludes that the message m was sent by some client whose one-time pseudonym is $pk_{\pi(i)}$. Thus, the server associates m with the reputation score corresponding to $pk_{\pi(i)}$ in the fresh pseudonym list. Such message posting design enables servers to attach the corresponding reputation scores to clients' one-time pseudonyms without learning their long-term identities.

If a client posts multiple messages in the same round, each message would be associated with the same pseudonym. Thus, we suggest that clients post one message in each round to avoid tracking at best-effort.

5.5 Feedback Phase

Finally, group members enter the feedback phase. Clients can provide feedback (either positive or negative) to different messages to indicate the quality of the messages. At the end of this phase, the servers update the reputation of each one-time pseudonym based on the feedback on its messages. Then, the servers perform the announcement phase in reverse to "transform" the updated fresh pseudonym list back to the reputation list consisting of clients' long-term identities and their now updated and again encrypted reputations. The feedback phase could start at the same time as message posting phase or after message posting phase, but should not end before message posting phase.

Feedback collecting. In our design, during a feedback phase, any client can submit her upstream servers her feedback on messages posted by other clients. In various applications, feedback may take a different form. For example, in a message board application like Yelp, feedback consists of votes clients cast while in Twitter, feedback is in a form of following another person. In AnonRep, we use +1 and -1 to denote positive and negative feedback, respectively.

Suppose some client wants to provide some feedback F (*i.e.*, +1 or -1) to a message m, she creates a tuple in the form of $\langle F, m \rangle$, and generates a linkable ring signature for this tuple $\sigma(\langle F, m \rangle)$, by following the signature generation step in §4.2. The client uses anonymous communication tool (the same as in the message posting phase) to send the tuple and the signature to her upstream server. *Note:* When the client generates the linkability tag, she needs to use $t = H(H'(m) + \vec{pk})$ rather than $t = H(\vec{pk})$ mentioned in §4.2, where $H'(\cdot)$ is another public hash function, $H'(m)$ is message m's hash value, and $\vec{pk}$ is the fresh pseudonym list in the current round. The goal of this design is to prevent clients from submitting duplicate feedback on the same message. If a malicious client signs and submits duplicate feedback on the same message m, then this behavior would be detected by the `Check` primitive (in §4.2), because both linkable ring signatures are generated by the same $t = H(H'(m) + \vec{pk})$ and private key. In this case, the duplicate feedback would be ignored by AnonRep. On the other hand, if a client signs feedback on different messages, the generated signatures would be different since they have different hash values.

If the upstream server's verifications succeed (including both verification and linkability checking steps in §4.2), the server associates the received feedback with the message m.

In summary, AnonRep derives two capabilities from linkable ring signatures. First, no member learns who provided feedback on the messages, except that it came from a member of the AnonRep group. Thus, each client's activities remain unlinkable, even if she provides feedback on multiple messages in the same round. Second, if some dishonest client submits duplicate feedback to the same message, such behavior is detected.

Reputation updating. At the end of the feedback phase, the servers update the reputation of every one-time pseudonym based on the feedback received on the messages associated with it. Because the collected feedback is stored in plaintext, AnonRep can utilize diverse reputation algorithms. For example, one-time pseudonym x's message receives 3 positive and 2 negative votes. If x's current reputation score is 4, then AnonRep will update x's reputation to 4 + (3-2) = 5.

After the reputation updates, servers successively perform the *reverse* scheduling protocol to "transfer" the current fresh pseudonym list containing each client's one-time pseudonym and *updated* reputation score back to the reputation list. *Note:* This new reputation list is similar to the input reputation list for the current round, but the encrypted reputation score of each long-term identity in the new reputation list has been updated.

So far, we described one message-and-feedback round, which has updated clients' reputation scores based on their activities while protecting their privacy. The new reputation list would be used as the input for the next round (see Figure 4).

5.6 Practical Considerations

This section presents several practical issues AnonRep faces and possible solutions to address them.

5.6.1 Performance Optimization

The announcement phase and the feedback collecting phase may cause long latencies when the client population is large, *e.g.*, $\geq$ 100,000. This is because the cryptographic primitives used in these two phases – verifiable shuffle and linkable ring signatures – become more computationally expensive as the number of clients grows.

AnonRep addresses this issue by randomly assigning clients into multiple sub-groups, and each sub-group operates in parallel. For instance, with 100,000 clients in total, the original design takes about 15 minutes on each server to run the scheduling protocol. With 20 sub-groups of 5,000 clients each, running scheduling protocol on each server takes only 50 seconds thanks to the parallelization. There is a trade-off to make, however. Larger sub-groups provide better anonymity while smaller sub-groups provide better performance.

5.6.2 Misbehavior Detection

Under our trust model, servers may misbehave. There are two possible cases. First, honest servers may notice that some announcement(s) have not been performed correctly by checking the zero-knowledge proofs of correctness generated in the announcement phase when performing the scheduling protocol. It is straightforward for honest servers to check the proofs to detect misbehaving servers. If a proof produced by server j fails, this indicates the server's misbehavior.

Second, clients may find that their reputation scores appear incorrect. There can be multiple causes: a) an upstream server incorrectly attaches the client's reputation score to its posted message; b) the reputation update is performed incorrectly; or c) a reputation is incorrectly initialized during the registration process. In order to detect these types of misbehaviors, the victim client enters into a blame phase where AnonRep randomly selects a witness (*e.g.*, an AnonRep server) to replay the corresponding operations and check all the signatures during the replay. Specifically, for case a) and b), the witness checks the corresponding upstream server who attaches and updates the reputation score, and then identifies whether the server is at fault. Even if the selected witness is dishonest and it does not perform the blame phase properly, no sensitive information is leaked, and the victim client simply needs to re-launch the blame process until she finds an honest witness. For case c), we discuss the solution in §5.6.3.

5.6.3 Registration Verification

The reputation score might be incorrectly initialized during the registration phase, in two ways: 1) a malicious server initializes an incorrect reputation score for a honest newcomer (mentioned in §5.6.2); and 2) a malicious newcomer colludes with a malicious server to assign herself a very high reputation score.

AnonRep can address this problem by asking each server to additionally run a verifiable encryption shuffle [32], which is similar to the version described in §4.1, but only includes the encryption operation. This is because performing a verifiable encryption shuffle enables a server to generate a proof on whether the encryption operation is correctly performed and whether the encrypted value has a desired value (*i.e.*, a correct, initial reputation in this case). In particular, for any newcomer client, each server first adds one ciphertext layer on her initial reputation score, which is a public value in the system setting, then produces a corresponding proof based on verifiable encryption shuffle, and finally sends the above results (ciphertext and proof) to the next server. If some malicious server does not use a correct initial reputation score or does not correctly perform the encryption, then it would be detected by some honest server(s) (at least one).

6 A Security-Enhanced AnonRep

In the design of AnonRep described so far, the reputation scores of AnonRep clients are operated as plaintexts during each round. Such a design, however, may introduce some potential information leakage in certain situations. Suppose in a certain AnonRep group, for instance, a client has a significantly higher reputation score (*e.g.*, 1000) than all the other clients' reputations (*e.g.*, lower than 10). Even though AnonRep enables clients to post messages with different one-time pseudonyms in different rounds, this client's messages could still be tracked across rounds, since her reputation score is too exceptional to hide herself in this group.[1]

The insight on avoiding the privacy leakage through exceptional reputations is to encrypt reputation scores. Thus, we propose a security-enhanced system design called the *reputation budget scheme*. Below we present the design of the security-enhanced AnonRep.

Client registration. When a client i registers, her upstream server S_i generates this client's initial reputation score. Then, all the servers successively encrypt this score using a homomorphic encryption scheme (*e.g.*, ElGamal [20] or Paillier [34]). We use $E_{Hom}(r_i)$ to denote the client i's reputation score r_i which has been homomorphically encrypted by all the servers. Once S_i receives $E_{Hom}(r_i)$, it takes $E_{Hom}(r_i)$ as the input to perform the basic client registration protocol as usual (§5.2), finally obtaining $E(E_{Hom}(r_i))$.

Announcement phase. The servers perform the same announcement phase as in the basic design. Notice however that, the reputations in the generated fresh pseudonym list are no longer in plaintext. Rather, the reputation scores are homomorphically encrypted by all the servers. This means servers in security-enhanced AnonRep no longer can see reputation scores in plaintext.

Message posting phase. When a client i wants to post a message in some round, she may leverage the Camenisch *et al.* proof system [11] or Peng *et al.* [36] to generate a zero-knowledge *reputation budget proof*, PoK, claiming that 1) her actual reputation score is not lower than a budget b, and 2) she wants to use b as the reputation score to post this message. For example, if a client has a reputation score 5, she can use any score no higher than 5 to post her message, *e.g.*, $b = 2$. b is called the *reputation budget*, and is plaintext. *Note:* we just apply the above proof systems [11, 36] as black-box to construct the needed reputation budget proofs. These two proof systems correspond to differnet homomorphic encryption schemes (*i.e.*, ElGamal and Paillier) used during the client registration, respectively.

After generating the reputation budget proof, the client i sends her upstream server a tuple containing the reputation budget and its proof.[2]

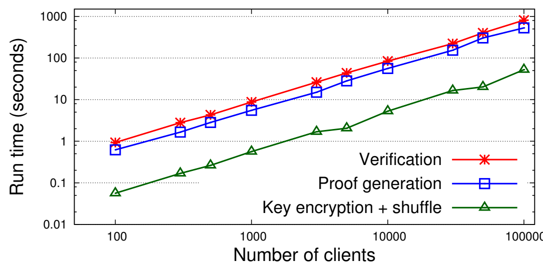
Upon receiving the tuple, the upstream server verifies the proof contained in the tuple. The server learns two things: 1) whether the client i is the owner of her claimed one-time pseudonym; and 2) whether the client i's reputation budget is no more than her actual reputation score. If the verification passes, the server posts the client's message with the reputation budget b.

Because the reputation budget proof is zero-knowledge proof, servers cannot know clients' actual reputation scores. As a result, for a client who has a distinctive reputation score (say, 1000), she can use a relatively low reputation budget (*e.g.*, 5) to post messages, hiding herself in the group. In practice, how to choose an optimal reputation budget depends on the security considerations of specific scenarios.

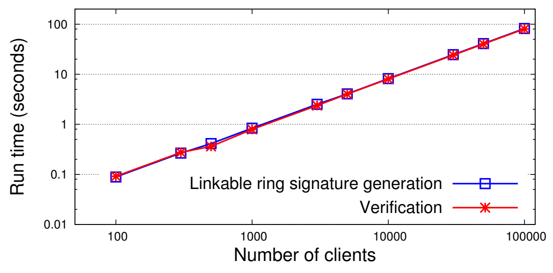
Feedback phase. Feedback collection is the same as the basic design. Reputation updating is different. In particular, at the end of feedback phase, servers first successively encrypt the received feedback (*e.g.*, votes) leveraging the same homomorphic scheme as used in the registration phase. Servers then update the clients' encrypted reputation scores in the fresh pseudonym list. *Recall:* both the reputation scores in the fresh pseudonym lists and the reputation values from the feedback are encrypted by the same homomorphic encryption scheme. Thus, servers can directly operate the ciphertexts to update the reputation scores for clients due to the homomorphic property.

[1] We make no claim that such situations always happen in reality, because: 1) most reputation systems have an upper bound for reputation scores, and 2) the number of clients with "the highest reputation scores" is normally not too low, as shown in the Stack Overflow dataset [1]. Nevertheless, we still manage to enhance AnonRep to accommodate such situations.

[2] For different zero-knowledge proof constructions, the tuple may be different. For example, for an ElGamal-style construction (*i.e.*, choosing Camenisch *et al.* proof system [11]), the tuple is $\langle R_i, C_i, b, \mathsf{PoK} \rangle$, where C_i is the i's actual reputation in ciphertext (*i.e.*, $E_{Hom}(r_i)$), and b is the reputation budget, R_i is another ciphertext serving for the proof.



(a) Neff verifiable shuffle.



(b) Linkable ring signature.

Figure 6: Microbenchmark (run time) evaluation of cryptographic operations.

This concludes our enhanced system design. Throughout the process, no server learns the actual reputation score of any client, only the reputation budget, thus preserving clients' information even if some clients' reputation scores are significantly different from all others.

7 Implementation and Evaluation

In this section, we first describe our prototype implementation, and then evaluate the prototype.

7.1 Implementation

We have implemented a functional AnonRep prototype. The prototype consists of 2700 lines of Go code as measured by CLOC [18]. Our implementation heavily depends on an open-source Go library of advanced crypto primitives including verifiable Neff shuffle [32], linkable ring signature [26], and various zero-knowledge proofs [11]. More specifically, our prototype implements the complete basic AnonRep design, and all group members in our prototype use UDP to communicate. We support only a limited reputation budget proof construction. The source code of our prototype is available on GitHub.[3]

7.2 Evaluation

Our experiments used NIST QR-512 quadratic residues, although the implementation also works and has been tested with other options such as NIST QR-1024 and QR-2048. We deployed servers in Amazon EC2 as virtual machines. In particular, we used the `c4.8xlarge` instances each with 36 Intel Xeon E5-2666 v3 CPU cores, 60 GB of RAM, and 10 Gbps of network bandwidth. We used laptops as the AnonRep clients each equipped with Intel Core i7 2.6 GHz and 16 GB of RAM.

7.2.1 Microbenchmark

A major performance bottleneck in AnonRep is the overhead of the two cryptographic operations: verifiable shuffle and linkable ring signature.

Figure 6a shows the computational overheads of Neff verifiable shuffle's three main building blocks: 1) key encryption and shuffle, 2) proof generation, and 3) verification. The key encryption and shuffle operation is very efficient, since it only involves simple ElGamal encryptions and element permutations. On the contrary, the proof generation and verification are more expensive, since they need to generate and verify a non-interactive zero knowledge proof, respectively.

Figure 6b shows the run time of generating and verifying linkable ring signatures with different number of clients. Both operations are of very similar cost, and they cost less than 100 seconds even with 10,000 clients. Furthermore, we observe that the computational overheads of both Neff verifiable shuffle and linkable ring signature increase linearly with the number of participating clients.

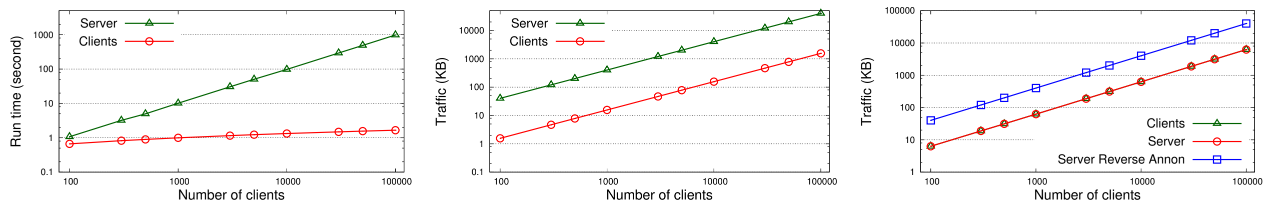
7.2.2 System Overheads

To understand the practicality of AnonRep, we measured server's and client's computational and bandwidth overheads during each phase.

Announcement phase. Figure 7a and Figure 7b show the computational and bandwidth overheads in the announcement phase. Here, each server performs the scheduling protocol, which contains 1) verifying the proof from the former server, 2) encrypting keys and striping one layer from the reputation ciphertext, and 3) generating the proof. To speedup the system, the server performs the proof generation (*i.e.*, step 1 and 2) and the verification (*i.e.*, step 3) in parallel. As shown in Figure 7a, with 100,000 clients, each server needs about 1,000 seconds to execute the scheduling protocol. The computational overhead at client is much less. This is because each client only needs to find its fresh pseudonym whose complexity is $O(\log n)$.

Regarding the bandwidth overhead, each server needs to send its successive server an "intermediate" list containing all clients' keys and encrypted reputation scores, as well as a proof, as shown in Figure 7b. This results in about 40 MB bandwidth overhead if there are 10,000 clients in the network. This is acceptable in practice given the fact that servers are reliably connected. Figure 7b also shows that the client's bandwidth overhead is about 1.5 orders of magnitude smaller than server's

[3]https://github.com/anonyreputation/anonCred



(a) Announcement phase: computational overhead (b) Announcement phase: bandwidth overhead (c) Feedback phase: bandwidth overhead

Figure 7: Comparison of computational and bandwidth overheads between server and client in different phases.

bandwidth overhead. This effectively allows even mobile devices to join our system as the clients.

Message posting phase. In the message posting phase, the only crypto operations are the well-known ElGamal signature generation (client side) and verification (server side). In particular, we want to understand the client's throughput, *i.e.*, how many messages a client can create and sign per second. With different message lengths, we find that a client can generate and sign ten 10MB messages per second, and a server can verify about one hundred 1MB messages per second.

Feedback phase. The feedback phase consists of two steps: feedback collection and reverse scheduling. We mainly measured the overhead of feedback collection, because the overhead of reverse scheduling is the same as the overhead in the announcement phase.

The overheads in the feedback phase are mainly caused by the linkable ring signature operations whose computational overheads have been shown in Figure 6b. Figure 7c shows the bandwidth overhead of each client and server in the feedback phase. We observe that the bandwidth overhead of each feedback is reasonable. For example, in a scenario with 10,000 clients, the bandwidth overheads at the client and server are 500KB and 5MB, respectively.

7.2.3 Practical Deployment

We now discuss and evaluate the AnonRep's deployment in practice, *i.e.*, how to set the duration of each phase and how many servers should be used for an anonymity group. Answering these questions is not straightforward. It depends on specific application scenarios.

In this section, we take Stack Overflow as a sample scenario. We utilize the analytical results from a large Stack Overflow dataset [1]. In particular, we first extract the analytical results conducted by Bhat *et al.* [6], and then discuss how to deploy AnonRep in Stack Overflow based on these results. Finally, we evaluate this AnonRep deployment.

From the measurement study [6], we extract the following features of Stack Overflow: F-1) more than 80% questions receive the accepted answers within 16 hours, and F-2) questions receiving more positive feedback can get accepted answers more quickly (*e.g.*, less than 10 hours).

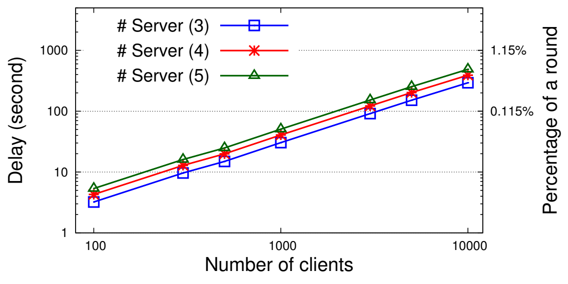


Figure 8: Delay of announcement phase.

According to the feature F-1, we suggest AnonRep set the message posting phase to 16 hours, thus enabling the majority of questions to receive accepted answers within one round. Due to the feature F-2, we allow the feedback phase to start at the same time as the message posting phase in each round. This enables questions to receive answers as soon as possible.

We deployed our prototype on multiple Amazon EC2 `c4.8xlarge` virtual machines, and measured the delay caused by the announcement phase with different numbers of clients and servers. Figure 8 shows that even though we use five servers for the announcement phase, the delay caused by the announcement phase is within 1% of a 24-hour round.

8 Discussion and Limitations

This section discusses some of AnonRep's limitations, and potential solutions.

Intersection attacks via online status. Current AnonRep cannot defend against long-term intersection attacks [24] which target otherwise-honest clients who repeatedly come and go during an interaction period, leaking information to an adversary who can correlate online status with activities across multiple rounds. There is no perfect defense against such intersection attacks when online status changes over time [24]. AnonRep may adopt a *buddy system* [43] whereby a client posts messages and feedback only when *all* of a fixed set of buddies are online. With certain caveats, this discipline ensures that a client's anonymity set includes at least his honest buddies, at the availability cost of making the user unable to transmit (safely) when *any* buddy is offline.

Weighted feedback. Our current design does not support weighted feedback. Namely, all the feedback in our current design has equal impact. Weighted feedback can differentiate clients' feedback. To enable this, the system needs to know the reputation scores of clients who are providing feedback. A possible solution is to introduce another announcement phase between the message posting phase and feedback phase, associating each feedback with its corresponding weight (*i.e.*, reputation score).

Malicious servers. Given the fact that AnonRep applies verifiable shuffle, any malicious servers can be detected and it is possible for an honest server to reveal a malicious server. However, it does not yet have a mechanism to prevent $m-1$ malicious servers from claiming that the other honest server is malicious, since the clients do not know the identity of the truly honest server. Although this can hardly happen as the servers in AnonRep are assumed to be managed by separate operators, it would obviously be better to be able to defeat malicious servers.

9 Related Work

Building an anonymous reputation system is challenging [5, 28]. To our knowledge, AnonRep is the first practical system in this domain.

Electronic cash based schemes. Various anonymous electronic cash (e-cash) protocols [3, 8, 10, 27] have been proposed to maintain the unlinkability of individual users' Peer-to-Peer transactions [13]. Some of them have been applied to build anonymous reputation systems [2, 9, 29]. For example, Androulaki *et al.* proposed RepCoin [2], which attempts to achieve a goal particularly close to AnonRep. However, a general disadvantage of e-cash based anonymous reputation systems, including RepCoin, is that they are incapable of supporting negative feedback, which means reputation of malicious users cannot be confiscated [45]. In addition, e-cash based systems cannot offer fine-grained reputation representations and updates, since all the reputation coins have the same value.

Signature-based approaches. Applying reputation signatures is another approach to preventing users from being tracked. Existing efforts [2, 14, 40] leverage blind signatures to hide the origin and destination of each reputation-based transaction. However, blind signature based approaches heavily depend on the assumption that the centralized authority behaves correctly. Another effort close to our work is to use the signature of reputation [5]. Specifically, each user may express trust in others by voting for them, collect votes to build up her own reputation, and attach a proof of her reputation to any message she posts, while maintaining the unlinkability of her activities. Similar to e-cash based approaches, however, with signature of reputation users cannot express negative feedback, and this approach is also computationally expensive.

Electronic voting based schemes. Electronic voting (e-voting) schemes allow the casting of votes while protecting user privacy [21, 22]. However, e-voting schemes are specifically designed for an election scenario where the candidates have no need to track their historical activities or publish any messages with updated reputation.

Others. Pavolv *et al.* [35] proposed a decentralized system allowing for partial privacy preservation on the user side and easy additive aggregation of users' reputation across the system. A malicious user, however, can easily track other users' activities by assigning specific reputation to victims. In addition, Clauß *et al.* [15] proposed two privacy requirements for reputation systems, *i.e.*, *k*-anonymity and weak rating secrecy. These enable the specification of practical reputation systems for providing strong privacy guarantees.

10 Conclusion

AnonRep is the first practical reputation system which supports regular reputation utilities while maintaining the unlinkability and anonymity of users' historical activities. AnonRep achieves this goal by an elegant integration of cryptographic techniques, *e.g.*, verifiable shuffles and linkable ring signatures, with a multi-provider deployment architecture. The experimental evaluation based on our functional prototype suggests that AnonRep can be applied to existing online services to provide the anonymous reputation utility.

Acknowledgements

We thank the anonymous reviewers and our shepherd, Ratul Mahajan, for their insightful comments. We also thank Mahesh Balakrishnan, Henry Corrigan-Gibbs, Michael J. Fischer, Krishna P. Gummadi, Jon Howell, and Nickolai Zeldovich for their valuable feedback on earlier drafts of this paper. This research was supported in part by the NSF under grants CNS-1149936, CNS-1409599, and CNS-1407454.

References

[1] Stackoverflow dataset. `http://www.ics.uci.edu/~duboisc/stackoverflow/`.

[2] Elli Androulaki, Seung Geol Choi, Steven M. Bellovin, and Tal Malkin. Reputation systems for anonymous networks. In *8th Privacy Enhancing Technologies (PETS)*, July 2008.

[3] Man Ho Au, Willy Susilo, and Yi Mu. Practical anonymous divisible e-cash from bounded accumu-

lators. In *12th Financial Cryptography and Data Security (FC)*, January 2008.

[4] Lars Backstrom, Cynthia Dwork, and Jon M. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *16th International Conference on World Wide Web (WWW)*, May 2007.

[5] John Bethencourt, Elaine Shi, and Dawn Song. Signatures of reputation. In *14th Financial Cryptography and Data Security (FC)*, January 2010.

[6] Vasudev Bhat, Adheesh Gokhale, Ravi Jadhav, Jagat Sastry Pudipeddi, and Leman Akoglu. Min(e)d your tags: Analysis of question response time in StackOverflow. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, August 2014.

[7] Johannes Blömer, Jakob Juhnke, and Christina Kolb. Anonymous and publicly linkable reputation systems. In *19th Financial Cryptography and Data Security (FC)*, January 2015.

[8] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *24th International Conference on the theory and Applications of Cryptographic Techniques (EUROCRYPT)*, May 2005.

[9] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Balancing accountability and privacy using e-cash (extended abstract). In *5th Security and Cryptography for Networks (SCN)*, September 2006.

[10] Jan Camenisch, Anna Lysyanskaya, and Mira Meyerovich. Endorsed e-cash. In *28th IEEE Symposium on Security and Privacy (S&P)*, May 2007.

[11] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical Report 260, Dept. of Computer Science, ETH Zurich, March 1997.

[12] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[13] Guihai Chen and Zhenhua Li. *Peer-to-Peer network: Structure, application and design*. Beijing: Tsinghua University Press, 2007.

[14] Delphine Christin, Christian Roßkopf, Matthias Hollick, Leonardo A. Martucci, and Salil S. Kanhere. IncogniSense: An anonymity-preserving reputation framework for participatory sensing applications. *Pervasive and Mobile Computing*, 9(3):353–371, 2013.

[15] Sebastian Clauß, Stefan Schiffner, and Florian Kerschbaum. *k*-anonymous reputation. In *8th ACM Symposium on Information, Computer and Communications Security (AsiaCCS)*, May 2013.

[16] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in Verdict. In *22nd USENIX Security Symposium*, August 2013.

[17] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Conference on Theory and applications of cryptographic techniques (EUROCRYPT)*, May 1997.

[18] Al Danial. Counting Lines of Code. `http://cloc.sourceforge.net/`.

[19] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *13th USENIX Security Symposium*, August 2004.

[20] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George Blakley and David Chaum, editors, *Advances in Cryptology*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1985.

[21] Jens Groth. Evaluating security of voting schemes in the universal composability framework. In *2nd Applied Cryptography and Network Security (ACNS)*, June 2004.

[22] Jens Groth. Non-interactive zero-knowledge arguments for voting. In *3rd Applied Cryptography and Network Security (ACNS)*, June 2005.

[23] Shouling Ji, Weiqing Li, Neil Zhenqiang Gong, Prateek Mittal, and Raheem A. Beyah. On your social network de-anonymizablity: Quantification and large scale evaluation with seed knowledge. In *22nd Network and Distributed System Security Symposium (NDSS)*, April 2015.

[24] Dogan Kedogan, Dakshi Agrawal, and Stefan Penz. Limits of anonymity in open environments. In *Workshop on Information Hiding*, October 2002.

[25] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle: An efficient communication system with strong anonymity. In *16th Privacy Enhancing Technologies (PETS)*, July 2016.

[26] Joseph K Liu and Duncan S Wong. Linkable ring signatures: Security models and new schemes. In *Computational Science and Its Applications (ICCSA)*, May 2005.

[27] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from Bitcoin. In *34th IEEE Symposium on Security and Privacy (S&P)*, May 2013.

[28] Tehila Minkus and Keith W. Ross. I know what you're buying: Privacy breaches on eBay. In *14th International Symposium on Privacy Enhancing Technologies (PETS)*, July 2014.

[29] Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Kim Pecina. Privacy preserving payments in credit networks: Enabling trust with privacy in online marketplaces. In *22th Annual Network and Distributed System Security Symposium (NDSS)*, February 2015.

[30] Arvind Narayanan, Hristo S. Paskov, Neil Zhenqiang Gong, John Bethencourt, Emil Stefanov, Eui Chul Richard Shin, and Dawn Song. On the feasibility on Internet-scale author identification. In *IEEE Symposium on Security and Privacy (S&P)*, May 2012.

[31] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *29th IEEE Symposium on Security and Privacy (S&P)*, May 2008.

[32] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *8th ACM Conference on Computer and Communications Security (CCS)*, November 2001.

[33] Lan Nguyen, Reihaneh Safavi-Naini, and Kaoru Kurosawa. Verifiable shuffles: a formal model and a Paillier-based three-round construction with provable security. *Int. J. Inf. Sec.*, 5(4):241–255, 2006.

[34] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology (EUROCRYPT)*, May 1999.

[35] Elan Pavlov, Jeffrey S. Rosenschein, and Zvi Topol. Supporting privacy in decentralized additive reputation systems. In *2nd Trust Management (iTrust)*, March 2004.

[36] Kun Peng, Colin Boyd, Ed Dawson, and Byoungcheon Lee. Ciphertext comparison, a new solution to millionaire problem. In *7th Information and Communications Security (ICICS)*, December 2005.

[37] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *7th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, December 2001.

[38] Stefan Schiffner, Andreas Pashalidis, and Elmar Tischhauser. On the limits of privacy in reputation systems. In *Workshop on Privacy in the Electronic Society (WPES)*, October 2011.

[39] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Scalable private messaging resistant to traffic analysis. In *25th ACM Symposium on Operating Systems Principles (SOSP)*, October 2015.

[40] Xinlei Oscar Wang, Wei Cheng, Prasant Mohapatra, and Tarek F. Abdelzaher. ARTSense: Anonymous reputation and trust in participatory sensing. In *32nd IEEE International Conference on Computer Communications (INFOCOM)*, April 2013.

[41] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Scalable anonymous group communication in the anytrust model. In *European Workshop on System Security (EuroSec)*, April 2012.

[42] David Isaac Wolinsky, Henry Corrigan-Gibbs, Aaron Johnson, and Bryan Ford. Dissent in numbers: Making strong anonymity scale. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, October 2012.

[43] David Isaac Wolinsky, Ewa Syta, and Bryan Ford. Hang with your buddies to resist intersection attacks. In *20th Conference on Computer and Communications Security (CCS)*, November 2013.

[44] Andrew Chi-Chih Yao. Protocols for secure computations (Extended abstract). In *23rd Annual Symposium on Foundations of Computer Science (FOCS)*, November 1982.

[45] Ennan Zhai, Ruichuan Chen, Zhuhua Cai, Long Zhang, Huiping Sun, Eng Keong Lua, Sihan Qing, Liyong Tang, and Zhong Chen. Sorcery: Could we make P2P content sharing systems robust to deceivers? In *9th IEEE Peer-to-Peer Computing (P2P)*, September 2009.