An Anonymous Reputation System Resistant Against Tracking and Intersection Attacks

CPSC 490 Final Project

Ian Zhou and Tony Jiang
Advisor: Ennan Zhai
Submitted: May 3, 2018

AnonRep is an anonymous reputation system developed by Ennan Zhai in 2016 that enables users to anonymously post new messages, tagged with their reputation scores, to a global message board. Users can provide feedback on previously posted messages without revealing their identity or any otherwise sensitive information [1]. Unlike traditional reputation systems, AnonRep prevents attackers from linking identities between messages by providing stronger guarantees on privacy and anonymity. However, AnonRep is prone to long-term intersection attacks, which can remove the guarantee of anonymity for certain individuals. For example, if a system contains one user with a significantly higher reputation score than all other users, an attacker may be able to link posts of high reputation to the same person. In this paper, we present AnonRep++, an improved version of AnonRep that incorporates blockchain technology to give users control over the amount of reputation that they attach to their posts. Instead of having a reputation explicitly attached to each client in the system, each user has access to a set of reputation wallets. The CoinShuffle protocol is used to preserve anonymity of these client wallets. For this project, I develop the base AnonRep program in Python, from which we implement AnonRep++. Both systems consist of a coordinator, message board, servers, and clients. I also implement the cryptographic primitives needed for the system’s guarantees on security, which include ElGamal encryption and decryption, linkable ring signatures, and verifiable shuffle. I perform analyses on the performance, efficiency, and scalability of both the original AnonRep and our blockchain-based AnonRep++. 
Our implementation can be found on Github. ¹

1 Introduction

Currently, many online services such as eBay, Yelp, and Quora employ reputation systems to evaluate information quality and filter out spam. In a general setting, a reputation system allows users to post messages with their reputation scores and to give feedback (in the form of upvotes or downvotes) on posted messages. However, most practical systems used today do not provide anonymity for their users; any server can examine the actions performed by their clients and link actions together based on users’ long-term identities and reputation scores.

AnonRep operates in a series of message-and-feedback rounds. In practice, each round may last anywhere from a few minutes to an entire day, depending on the application scenario. Each round is split into three phases: announcement, message posting, and feedback. At the beginning of a round, each server maintains a table of all of the clients’ long-term pseudonyms and their corresponding encrypted reputation scores. In the announcement phase, the servers collectively create a table of encrypted one-time pseudonyms (from the clients’ long-term pseudonyms) and their respective decrypted reputation scores. In the message posting phase, clients post messages using these one-time pseudonyms to a global message board maintained by all of the servers. Finally, in the feedback phase, clients anonymously provide feedback to the posted messages. Long-term reputation scores are updated using a reverse-scheduling protocol before the next round begins. For the remainder of this paper, we refer to this protocol as the reverse announcement phase.

AnonRep++ operates very similarly to AnonRep, but does not require the announcement and reverse announcement phases. At the beginning of each round, each server maintains a table of just the clients’ long-term pseudonyms. Reputation is tokenized into “reputation coins” on the blockchain. In the message posting phase, clients post messages using reputation from wallets that they have access to. In the feedback phase, positive and negative feedback given to a message is translated to gains and losses of “reputation coins” to the wallets used for posting the message. Afterwards, the Coin-Shuffle protocol is used to distribute a new set of wallets to active clients for the next round. This eliminates the possibility of an intersection attack.

2 Architecture Overview

The three components used in our implementation of the AnonRep and AnonRep++ models include 1) a coordinator, 2) a small number of servers, and 3) a potentially large number of clients.

¹https://github.com/ianzhou1/anonrep
2.1 Coordinator

The coordinator is a single server that coordinates server and client registration. It also coordinates the announcement (AnonRep only), message-posting, and feedback phases used in our models. The coordinator is responsible for maintaining a list of all active servers. At the start of each round, the coordinator shuffles its list of servers and updates each server with its new neighbors according to the randomly shuffled list. It then proceeds through each aforementioned phase in order.

In our AnonRep and AnonRep++ implementations, the coordinator also maintains the global message board table. Each row of the message board consists of a message id, the message contents, the one-time pseudonym used to post the message, the reputation used to post the message, and the message’s overall feedback.

2.2 Server

Multiple servers are used to achieve scalability and client anonymity. In practice, the servers should be operated independently of one another. Each server is responsible for responding to a specific subset of clients.

In AnonRep, during the announcement phase, servers perform round-based encryption of client identities and decryption of reputation scores. During the reverse announcement phase, servers perform round-based decryption of client identities and encryption of reputation scores. More details are provided under the “System Design” section.

2.3 Client

A potentially large number of clients are used. Each client is assigned to a single server. When posting messages and providing feedback, a client must do so through its assigned server rather than directly through the coordinator.
3 Details of Cryptographic Building Blocks

Before elaborating on the designs of AnonRep and AnonRep++, we present the cryptographic primitives used in the system. In our implementations, we use a generator $g$ of prime multiplicative order $q$ in the multiplicative group of integers modulo prime $p$. For our purposes, we set $g = 2203$, $p = 16000393$, and $q = 666683$. In practice, significantly larger values of $g, p, q$ should be chosen to ensure the security of the cryptographic methods used.

3.1 ElGamal

Details on ElGamal encryption, decryption, signing, and verification can be found in Tony’s report.

3.2 Mix-Net and Verifiable Shuffle

Mix-Net is a routing protocol that creates hard-to-trace communications by using a chain of servers. Each server takes in a list of objects, randomly shuffles them, and then returns the shuffled list to the next destination. Assuming that at least one server is honest, this ensures unlinkability between the starting and ending lists.

In AnonRep, the Mix-Net protocol is used in the announcement and reverse announcement phases. This protocol involves encryption, decryption, and permutation operations. The starting list contains client long-term pseudonyms and their corresponding encrypted reputation scores. In the shuffle phase, each server adds one layer of encryption to the pseudonyms and decrypts one layer of the reputation scores. The list is permuted using a verifiable shuffle before being passed to the next server in the Mix-Net. In the reverse announcement phase each server decrypts one layer of the pseudonyms and adds one layer of encryption to the reputation scores before the verifiable shuffle is performed.
Our implementation of verifiable shuffle is based off the protocol outlined by Andrew Neff [2]. An outline of the protocol used is provided below.

- Prover: encrypt/decrypt pseudonyms using modular exponentiation.
- Prover: decrypt/encrypt reputation scores using ElGamal.
- Prover: shuffle pseudonyms and reputation scores using a randomly generated permutation index array.
- Prover: generate an honest-verifier zero-knowledge proof that the above operations were completed correctly.
- Verifier: verify the above options were completed correctly using a combination of the general $k$-Shuffle protocol and ElGamal $k$-shuffle protocol.

In AnonRep++, the announcement and reverse announcement phases have been removed. The Mix-Net protocol is no longer needed.

### 3.3 Linkable Ring Signature

A ring signature is a type of digital message signature that can be performed by any member of a group of users while maintaining anonymity of the signing member. A linkable ring signature introduces the added constraint of being able to link signatures made from the same user [3]. Linkable ring signatures are used by clients to sign votes made during the feedback phase. If a client signs its vote incorrectly or has already provided feedback for a particular message, the server rejects the signature and does not post the feedback to the message board.

In AnonRep, a linkable ring signature is created using the client’s private key and the list of all one-time pseudonyms for the current round. Since the one-time pseudonyms are made public, anyone can check the validity of any signature.

In AnonRep++, the announcement and reverse announcement phases have been removed. Instead, a linkable ring signature is created using the client’s private key and the list of all long-term pseudonyms. As in the previous case, all long-term pseudonyms are made public, so anyone can check the validity of any signature.

### 3.4 CoinShuffle

CoinShuffle is a decentralized shuffling protocol that removes links between transactions made among users. A basic version of coin shuffle is presented in Figure 3. Details on the CoinShuffle protocol can be found in Tony’s report.

### 4 System Design

The overall workflows for AnonRep and AnonRep++ are described below.
Figure 3: A diagram illustrating the coin shuffle protocol among four clients. In Phase 1, each client (other than the first) is given a public key. In Phase 2, each client adds an encrypted version of its wallet key to a list, randomly shuffles the list, and then passes the shuffled list to the following client. In Phase 3, wallets from the inputs are matched to wallets from the outputs.
4.1 AnonRep

4.1.1 Server Registration

A new server starts by generating a random private key \( z_j \) uniformly from \( \mathbb{Z}_q \) and computing its public key \( Z_j \in \mathbb{Z}_p \) such that \( Z_j = g^{z_j} \). The server registers itself with the coordinator by sending its IP address and public key \( Z_j \). The server also generates a random ephemeral (round-based) key \( e_j \) uniformly from \( \mathbb{Z}_q \).

4.1.2 Client Registration

A new client starts by generating a random private key \( y_i \) uniformly from \( \mathbb{Z}_q \) and computing its public key \( Y_i \in \mathbb{Z}_p \) such that \( Y_i = g^{y_i} \). The client sends its public key (long-term pseudonym) to its assigned server. The server generates an initial reputation score for the client. Each server \( S_j \) then sequentially encrypts the reputation score using the ElGamal encryption scheme with its public key \( Z_j \). Once all servers have finished encryption, the long-term pseudonym and respective encrypted reputation score are broadcast to all servers.

4.1.3 Announcement Phase

At the beginning of the announcement phase, each server maintains a round-based generator (initialized to \( g \)) and a table of the clients’ long-term pseudonyms and their corresponding encrypted reputation scores. These elements are passed around sequentially among all of the servers. Each server \( S_j \) performs the following series of events.

- Encrypt one layer of the list of pseudonyms using modular exponentiation by \( e_j \).
- Update the round-based generator using modular exponentiation by \( e_j \).
- Decrypt one layer of the list of reputation scores using ElGamal with private key \( z_j \).
- Generate a permutation \( \pi_j \) of length equal to the number of clients.
- Verifiably shuffle the entries in the table using \( \pi_j \).

Once all servers have finished this procedure, the round-based generator \( g^{\prod_j e_j} \) and table of one-time pseudonyms and their respective (decrypted) reputation scores are broadcast to all servers and clients.

4.1.4 Message Posting Phase

In the message posting phase, clients start by determining their one-time pseudonyms. For any particular round, a client with private key \( y_i \) has the one-time pseudonym \( g^{y_i \prod_j e_j} \). So, the client can compute its one-time pseudonym by taking the modular exponentiation of the round-based generator by its private key \( y_i \). To post a message, the client uses its one-time pseudonym and attaches a signature made using \( y_i \) to the
message. In our implementation, a standard ElGamal signature scheme was used. A server verifies that the message and one-time pseudonym match their corresponding signature. If they do, the message is passed to the coordinator, which adds it to a global message board.

4.1.5 Feedback Phase

In the feedback phase, clients submit feedback to their assigned servers. In our implementation, we limit our feedback to +1 and -1 for positive and negative feedback, respectively. Clients sign their feedback using a linkable ring signature, which hides the voting client’s identity and prevents duplicate votes from clients. Afterwards, the reverse announcement process is performed to update the initial reputation table. Each server $S_j$ performs the following series of events.

- Generate a new ephemeral key $e_j$ uniformly from $\mathbb{Z}_q$.
- Decrypt one layer of the list of pseudonyms. In practice, this can be done by recording the corresponding change during the announcement phase.
- Encrypt one layer of the list of reputation scores using ElGamal with ephemeral key $e_j$ and public key $Z_j$.
- Generate a permutation $\pi_j$ of length equal to the number of clients.
- Verifiably shuffle the entries in the table using $\pi_j$.

4.2 AnonRep++

4.2.1 Server Registration

Same as in AnonRep.

4.2.2 Client Registration

Same as in AnonRep, but there is no longer a need for sequential encryption of the initial reputation score. Instead, reputations are stored on the blockchain.

4.2.3 Message Posting Phase

Unlike in AnonRep, clients in AnonRep++ do not post messages using a one-time pseudonym. Instead, messages are posted using a number of “reputation coin” wallets. A client can prove its ownership of a wallet by including a digital signature made with the wallet’s private key. The amount of reputation given to the posted message is decided by the client, and verified by the server. For the verification step, the server checks that the sum of the reputations in all of the client’s provided wallets is at least the reputation amount specified by the client.
4.2.4 Feedback Phase

Unlike in AnonRep, reputations in AnonRep++ are stored on the blockchain. During the feedback phase, the coordinator maintains a list of positive and negative votes made for each message. At the end of the round, these votes are applied to the wallets used to post the corresponding message. In our implementation of AnonRep++, all wallets are capped at a minimum of 0 reputation.

4.2.5 CoinShuffle

Details on the CoinShuffle implementation for AnonRep++ can be found in Tony’s report.

5 Evaluation

5.1 Technical Specifications

Our experiments were performed on Zoo machines at Yale University. The coordinator server was run on ladybug. Four servers were used for benchmarking, and they were run on tiger, turtle, aphid, and dolphin. All clients were created on swan, since client operations were performed sequentially and interactions among clients was minimal. Each Zoo machine runs on Fedora 26 with 20 Intel Xeon(R) CPU E5-2650 v3 @ 2.30GHz cores and 62.8 GB of RAM. A network bandwidth of 76 Mbps was used.

5.2 Benchmarking Results

Six different benchmarking experiments were performed. See Figure 4 for plots of AnonRep and AnonRep++ performance.

5.2.1 Announcement and Reverse Announcement Benchmarking

For a low number of clients, the difference in speed for the announcement and reverse announcement phases is negligible. After a certain threshold (approximately 100), the time spent in these phases scales linearly with the number of clients, as expected.

5.2.2 Linkable Ring Signature Sign and Verify Benchmarking

The time spent in linkable ring signature signing and verifying scales quadratically with the number of clients. This is a result of an unoptimized portion of our hashing algorithm used in the linkable ring signature. As shown in the original AnonRep paper [1], it is possible for linkable ring signature signing and verifying to scale linearly.

5.2.3 Verifiable Shuffle Benchmarking

The time spent in verifiable shuffle scales linearly with the number of clients, as expected.
Figure 4: Benchmarking results of time in seconds vs. number of clients. Axes are displayed using a logarithmic scale, except for those of the CoinShuffle benchmarking.
5.2.4 CoinShuffle Benchmarking

The time spent in CoinShuffle as a function of the number of clients matches the expected time spent illustrated in the original CoinShuffle paper [4].

5.3 Comparison

Since linkable ring signatures are used in both AnonRep and AnonRep++, we focus on the benchmarking results for announcement (AnonRep), reverse announcement (AnonRep), verifiable shuffle (AnonRep), and CoinShuffle (AnonRep++). In general, the announcement phase, reverse announcement phase, and verifiable shuffle protocol take significantly less time to complete than the CoinShuffle protocol. CoinShuffle also fails to scale linearly as the number of clients increases. Methods of mitigating this issue are explored in more detail in the “Discussion” section.

6 Discussion

In this section, we discuss and compare different aspects of AnonRep and AnonRep++.

6.1 AnonRep++ Advantages

As stated previously, AnonRep suffers from intersection attacks. A client with a unique reputation score, (e.g. one that is much higher than all other users), has little anonymity. Any server or client can easily link messages posted from this client because its reputation score is, by assumption, unique. AnonRep++ solves this problem by giving users the option to post messages with reputation less than their total reputation. Since wallets are capped at a minimum of 0 reputation, anonymity can be maximized by posting messages with 0 reputation.

Another issue with AnonRep involves the use of a single ephemeral key for ElGamal encryption of all client reputation scores in each round. For two clients with the same reputation score, their corresponding encrypted reputations are exactly identical. This occurrence can potentially remove anonymity by linking clients with the same reputation scores together. AnonRep avoids this problem by assuming a high-churn reputation system. However, AnonRep++ removes this problem altogether by storing reputations on the blockchain.

6.2 AnonRep++ Disadvantages

Despite the increase in performance from removing the announcement and reverse announcement phases, AnonRep++ runs into a problem of scalability with the CoinShuffle protocol. As shown in Figure 4, CoinShuffle scales worse than linearly and takes significantly longer to complete than any of announcement, reverse announcement, and verifiable shuffle. However, our implementation of CoinShuffle is naive and can be substantially improved and optimized. Faster alternatives can also be used, such as
CoinShuffle++ [5]. Finally, for scalability purposes, clients can be divided into separate CoinShuffle rings and processed in parallel.

6.3 Future Work

To extend AnonRep++ even further, we can modify the system to have the message board moved onto the blockchain. This gives clients public access to all activity regarding the message board, which can help prevent malicious behavior from both servers and clients. Further security can also be added by signing all messages sent throughout the network. Minor optimizations can be made throughout the AnonRep++ codebase to improve overall performance. Additionally, as stated previously, we can parallelize portions of the CoinShuffle protocol to scale more effectively with larger numbers of clients.

7 Contributions

I primarily worked on building the base AnonRep system and implementing cryptographic primitives used throughout AnonRep and AnonRep++. Specific contributions can be seen by examining the commit history at our Github repository.

8 Conclusion

AnonRep++ is the first anonymous reputation system that is resistant against tracking and intersection attacks. AnonRep++ builds off of the system design presented in AnonRep with the inclusion of blockchain and the Coinshuffle protocol. Although our current implementation of Coinshuffle is not scalable, our benchmarking suggests that AnonRep++ with a parallelized Coinshuffle protocol can provide improved anonymity to many reputation-based services.

9 Acknowledgements

We would like to thank Ennan Zhai for the inspiration of this project, for providing insightful answers to our questions about the implementation of this project, and for his general feedback and guidance.

References


