Private Signaling Secure Against Actively Corrupted Servers

Haotian Chu[1], Xiao Wang[1], and Yanxue Jia[2]

[1] Northwestern University
{haotian.chu@,wangxiao@}northwestern.edu
[2] Purdue University
jia168@purdue.edu

Abstract. Private signaling allows servers to identify a recipient's messages on a public bulletin board without knowing the recipient's metadata. It is a central tool for systems like privacy-preserving blockchains and anonymous messaging. However, unless with TEE, current constructions all assume that the servers are only passively corrupted, which significantly limits their practical relevance. In this work, we present a TEE-free simulation-secure private signaling protocol assuming two non-colluding servers, either of which can be actively corrupted.
Crucially, we convert signal retrieval into a problem similar to private set intersection and use custom-built zero-knowledge proofs to ensure consistency with the public bulletin board. As a result, our protocol achieves lower server-to-server communication overhead and a much smaller digest compared to state-of-the-art semi-honest protocol. For example, for a board size of 2^{19} messages, the resulting digest size is only 33.57KB. Our protocol is also computationally efficient: retrieving private signals only takes about 2 minutes, using 16 threads and a LAN network.

1 Introduction

While protecting communication content has been well studied, communication metadata protection—hiding who and when sends or receives which messages—is still challenging and desirable. Private signaling was first introduced by Madathil et al. [20] to protect the metadata privacy of recipients. In this problem, senders and recipients communicate via a public bulletin board where messages are published, and each message corresponds to a location. After posting a message on the public bulletin board, the sender also generates a clue that will be collected by server(s). Then, the server(s) leverages the clue to assist the intended recipient in learning the location of the message (say, signal). During this process, the metadata of the recipient is protected.

Private signaling plays a key role in many applications. Specifically, it can be used to protect the metadata of resource-constrained recipients in privacy-preserving cryptocurrencies, such as Zcash [3] and Monero [22]. By using private signaling, the recipients do not need to download the whole blockchain (i.e., the public board) and find the pertinent transactions through linear scan. In

addition, as discussed by Madathil et al. [20], private signaling can also be used to design anonymous messaging applications (e.g., [7, 27]) where the messages could be collected by service server(s), rather than the public board.

Almost at the same time, Liu and Tromer [17] defined the same problem as Oblivious Message Detection (OMD), and extended it to Oblivious Message Retrieval (OMR), where the recipient can directly obtain the pertinent messages instead of just their locations. Since, after obtaining the locations, the recipient can use existing techniques such as Private Information Retrieval (PIR) [5, 11] to retrieve the corresponding messages without revealing metadata, we focus on private signaling (i.e., OMD) in this work.

While private signaling has been extensively studied, existing works all assume semi-honest server(s), unless relying on trusted execution environments (TEEs). However, given that these applications inherently lack trusted entities, active security is important. Specifically, cryptocurrencies are in an open and untrusted environment where anyone can join without any permissions. In addition, as summarized by Sasy and Goldberg [26], anonymous messaging applications are always used to proactively resist mass surveillance and protect whistleblowers who expose misconduct by governments or employers. Therefore, it is difficult to rely on government credibility or social reputation to prevent malicious behavior.

Next, we categorize the existing approaches into three types and explain why achieving active security is challenging for them.

1. **TEE-based Solutions.** The seminal work by Madathil et al. [20] leveraged a TEE to achieve a highly efficient 1-server solution, which was further improved to achieve even better scalability by Jakkamsetti et al. [13]. Their key idea is to require the server to maintain a table whose each row corresponds to a recipient and records the pertinent locations for the recipient. When a recipient requests a retrieval, the server responds with the corresponding row. To protect the metadata privacy for the recipient, the table is maintained in a TEE. While using a TEE enables the assumption of a malicious server, TEEs are known to suffer from side-channel attacks [6, 10] and memory limitations [8]. Therefore, Madathil et al. [20] also proposed a 2-server design to avoid relying on TEEs, a direction followed by all subsequent works, as described below.
2. **2PC-based Solutions.** To avoid relying on a TEE, Madathil et al. [20] leveraged garbled circuits run by two servers to maintain the table. However, the significant overhead makes it impractical for real-world adoption, let alone achieving active security. Recently, Jia et al. [14] used a completely novel approach to achieve a 2-server solution called HomeRun, whose efficiency is comparable to TEE-based designs. However, extending HomeRun to support active security is challenging. In addition to handling malicious behavior from one of the servers, it is also necessary to address the collusion between the malicious server and a subset of senders and recipients.
3. **FHE-based Solutions.** Another research line leverages fully/leveled homomorphic encryption to achieve 1-server solutions. The previous works

[16, 17, 18] in this line share the same design framework: Each clue includes ℓ ciphertexts, each encrypting a bit 1 under the public key of the intended recipient, and only using the corresponding secret key can decrypt all the ℓ ciphertexts to 1. With the assistance of the server, the recipients can decrypt to detect the pertinent clues positioned exactly where the corresponding messages reside. To protect the metadata of the recipient, all computations are performed under fully/leveled homomorphic encryption. Therefore, their schemes lack efficiency, especially when a large number of messages are published on the public board. Obviously, achieving active security directly based on the semi-honest designs would further degrade performance.

For all non-TEE-based constructions, designing a practical protocol secure against malicious servers is highly challenging. A malicious server can misbehave at any point during the retrieval process, which means that authentication must cover the entire flow — from the bulletin board to the final message sent to the receiver (namely, digest). Moreover, we need to guarantee that the receiver's cost is significantly lower than that of directly searching from the entire bulletin board. Otherwise, the system loses its intended value.

1.1 Our Contribution

In this work, we propose Two-Face, an actively secure private signaling protocol assuming two non-colluding servers while overcoming the above usability challenges and guaranteeing practical efficiency.

Achieving Full Active Security. To achieve full active security, Two-Face integrates tailored zero-knowledge proofs to ensure server-side inputs remain consistent with the public board, and adopts the dual-execution mechanism that enables the receiver to efficiently detect any malicious behavior with minimal overhead. Moreover, we prove the security via the simulation paradigm, which means that our approach is secure against any active attacks.

Overcoming Usability Challenges. Prior works either (1) incur communication overhead for the recipient that grows linearly with the number of messages on the public board, or (2) impose restrictions on the number of messages the recipient can retrieve. Two-Face addresses both challenges by performing a shuffling process on the server side. Moreover, the shuffling process itself is also utilized to achieve the aforementioned active security guarantee, and thus does not incur additional overhead on the servers.

Practical Efficiency through End-to-End Benchmarking. Finally, we demonstrate the practical efficiency of our approach through end-to-end benchmarking. Specifically, Two-Face achieves lower server communication than all prior work, and produces a digest of 33.57 KB for 50 pertinent signals — only $5\times$ larger than that of the most compact semi-honest protocol. It also processes one million messages in just 4 minutes using 16 threads on moderate hardware.

2 Technical Overview

2.1 Recap HomeRun

HomeRun [14] is the state-of-the-art two-server private signaling protocol in the semi-honest setting. We first provide an overview of their protocol to illustrate the challenges in designing an active private signaling protocol; then, we discuss the intuition of our protocol and how it can achieve active security efficiently.

How HomeRun Works. In HomeRun, two servers, namely S_a, S_b, publish their public keys, namely pk_a and pk_b. The address of a receiver is their public key, namely pk_R. A sender who wants to send a message to the address pk_R would put two ciphertexts on the bulletin board for private signaling: $\mathsf{Enc}_{\mathsf{pk}_a}(L_a)$ and $\mathsf{Enc}_{\mathsf{pk}_b}(L_b)$ such that $L_a + L_b = \mathsf{pk}_R$. Then, the two servers can obtain L_a and L_b, respectively, from the bulletin board.

A receiver with public key pk_R has secret key sk_R such that $\mathsf{pk}_R = g^{\mathsf{sk}_R}$. To retrieve messages designated to it, the receiver sends secret shares of its public key $\mathsf{pk}_R = L_a^* + L_b^*$ to two servers. The two servers, holding secret shares of the destination public key for each message, can compute the shares of difference between the destination public key for each message and pk_R. Now, two servers hold either secret shares of 0, meaning that the destination public key is $\mathsf{pk_R}$, or a non-zero value, meaning it is for other receivers. Two servers then run a two-party private equality check to obtain secret shares of 0 or 1, indicating such facts, and send these bits back to the receiver, who can reconstruct the private signals, i.e., the indices of 1.

Finally, to prevent an adversary from retrieving other receivers' messages/signals, HomeRun requires all receivers to perform proof of knowledge of the secret values. In particular, the receiver needs to prove in zero-knowledge (ZK) that it knows the discrete log of L_a^* to one server and the discrete log of L_b^* to the other server. In this way, a corrupted receiver cannot retrieve messages for an address without knowing the secret key.

Challenges in Designing Actively Secure Private Signaling. Although HomeRun is highly efficient, there are several drawbacks, e.g., the client-server communication is linear to the size of the entire board. Furthermore, it is very difficult to be made actively secure. Indeed, none of the existing private signaling protocols can tolerate malicious adversaries without relying on TEE. Below, we outline some key challenges in achieving active security.

- **Prevent Server-Receiver Colluding.** In HomeRun, ZKPoK of discrete log from the receiver to each server is needed to prevent any party without $\mathsf{sk_R}$ to retrieve the signals for address $\mathsf{pk_R}$. However, if an adversary corrupts a server and the receiver, they can still retrieve the signals from any public-key address. In more detail, an adversary corrupting the receiver and S_a can send a group element L_b^*, with known discrete log and prove it to the uncorrupted server S_b. To fetch signals designated to an address pk, the corrupted server S_a would use $L_a^* = \mathsf{pk} - L_b^*$, which would eventually allow the corrupted receiver

to obtain the signals for pk. This is very difficult to prevent as the server behaves honestly except that it uses a different input and skips a ZKPoK, which is not observable from the honest server.

- **Prevent Deviation from the Bulletin Board.** A corrupted server can introduce false negatives by deviating from the bulletin board. In HomeRun, this kind of deviation is hard to detect because the equality test input is not directly linked to the ciphertexts posted on the board. The server must first decrypt the ciphertext using its own secret key before performing the equality test. This procedure makes it difficult to verify that the input indeed corresponds to the public ciphertext, giving a malicious server the opportunity to cheat and drop potentially pertinent signals.
- **Lack of Output Verification.** Moreover, false negatives can also arise due to the lack of verification on the output. Specifically, a malicious server may manipulate the result of the private equality test by flipping its output bit, causing a valid signal to be dropped. Even if authentication is provided for the output of this test, verifying it would be costly for the client, as the output is already linear to the board size.

2.2 Our Approach

Basic Setup of Our Protocol. The first issue that we focus on is the receiver-server collusion to fetch messages designated to other addresses without their private keys. To solve this, we use a very different setup. Two servers no longer hold any private information, such as server private keys. Instead, to send a message to address pk, one would compute a clue as an ElGamal encryption of 1 using the address public key. For example, the clue to a message designated to address pk_R would be (g^r, pk_R^r) for some r. Since the server has no private information, a receiver colluding with the server cannot help as in HomeRun.

Note that this setup is somewhat similar to FHE-based private signaling protocols, but we argue that in the two-server setting, it could still work without the need for FHE. For a party with address pk_R and secret key $\mathsf{sk}_R = \mathsf{DLog}(\mathsf{pk}_R)$ to retrieve messages, it would send secret shares of sk_R to the two servers, namely sk_a and sk_b such that $\mathsf{sk}_a + \mathsf{sk}_b = \mathsf{sk}_R$. For a message designated to address pk with the clue $c = (g^r, \mathsf{pk}^r)$, two servers can distributively decrypt it by computing $h_a = 1/c[1]^{\mathsf{sk}_a}$ and $h_b = c[2]/c[1]^{\mathsf{sk}_b}$ respectively. We can see that $h_a \cdot h_b = c[2]/c[1]^{\mathsf{sk}_R} = (\mathsf{pk}/\mathsf{pk}_R)^r$. This means that this message is designated to the receiver if and only if $h_a = h_b^{-1}$.

Now, two servers have a list of values, and whether they are equal indicates if the corresponding indices are signals designated to this receiver. In particular, assuming that there are totally N messages on the public board. By the above method, two servers can each locally compute a set of values such that the indices where their respective values are equal should be the signals sent to the receiver. Namely, S_a holds vector $\boldsymbol{A} \in \mathbb{G}^N$ and S_b holds $\boldsymbol{B} \in \mathbb{G}^N$ and the receiver should obtain $\{i | \boldsymbol{A}[i] = \boldsymbol{B}[i]\}$. Of course, one could use private equality test protocols to obtain shares of the output, but making this idea maliciously secure would be

hard, and the protocol would still require linear communication between servers and the receiver.

Retrieving Signals Privately under Semi-honest Security. To make the above protocol scale and eventually actively secure, we adopt a very different approach without using private equality test protocols. The task now is somewhat similar to private-set intersection (PSI), and thus our protocol is adapted from [9], which was originally designed for private set intersection-cardinality (PSI-CA). Roughly speaking, we ask server S_a to encrypt its list and send the encrypted list, namely $\{\mathsf{Enc}(\boldsymbol{A}[i])\}_{i\in[N]}$ to S_b (we ignore the public key for encryption for clarity); then S_b homomorphically applies a list of mask $\boldsymbol{M}[i]$ and then permutes the list based on a permutation σ, namely S_b sends to S_a the list $\{\mathsf{Enc}(\boldsymbol{A}[\sigma(i)] + \boldsymbol{M}[\sigma(i)])\}_{i\in[N]}$. S_b also masks, permutes, and applies random oracle on its own list, and sends it to S_a, namely $\{\mathsf{H}(\boldsymbol{B}[\sigma(i)] + \boldsymbol{M}[\sigma(i)])\}_{i\in[N]}$. Now S_a locally decrypts the first list, applies the random oracle and computes the indices of elements in the intersection of the two lists. These indices look uniform to S_a because it is permuted by σ known only to S_b. The receiver can get the signal by getting the (permuted) intersection from S_a and σ from S_b. Although this protocol is not actively secure yet, it already allows for a small digest from the servers to the receiver since the digest size is linear to the number of relevant signals, not the board size.

Strengthening the Protocol to Active Security. There are multiple ways for a corrupted server to cheat. Our next key idea is to adopt the "dual execution" methods [12, 21] and run the above protocol twice, once with S_b providing the permutation and S_a getting the permuted result as above, and once with their roles swapped. Now, the receiver would obtain two copies of the result and accept only if they are the same. Intuitively, if one of the servers cheats, it can try to introduce false positives or false negatives. However, any cheating behavior after receiving the permuted and masked list requires guessing some information about the permutation, which could be caught with some probability if guessed incorrectly. Given this, we are left with two main challenges to solve:

- How to ensure the first message in each execution (i.e., the encrypted list) is consistent with the clues in the bulletin board. In particular, we would like to prevent S_a from corrupting the encrypted list before S_b permutes it in the first execution and, at the same time, corrupting the encrypted list from S_b in a consistent way in the second execution. This would potentially allow a corrupted S_a to drop a relevant signal.
- How to ensure that any cheating can be detected with overwhelming probability. Although cheating may be caught with some probability, we want to ensure that the chance of undetected cheating is negligible.

Generically, the first problem can be solved by using a ZKP protocol to prove that the encryptions are correctly performed: ciphertext are well formed and the message is consistent with the decryption from the bulletin board. The second problem can be solved by injecting public elements into the two servers' lists so that they are expected to appear in the output; the receiver now reconstructs

the set and also needs to check if all public elements are in the result. However, this may not lead to a practical solution since we now need to use ZKP to prove correct encryption.

Making Everything Concrete Efficient. Our last ingredient is to make the above idea concrete by simplifying the semi-honest protocol so that our ZKP only needs to prove a simple relationship that can be done efficiently.

First of all, we replace the encryption in the scheme to a single blind. For example, server S_a now would compute and send $\{(R_i \boldsymbol{A}[i])^{\xi}\}$ where R_i is a public uniform group element and ξ is a global uniformly distributed exponent as a blind. Note that the list is indistinguishable from uniform group elements under DDH [4]; thus, it can serve the purpose of encryption. S_b could mask the list using a single blind ω for the same reason. Now S_b would send to S_a lists $\{(R_{\sigma(i)} \boldsymbol{A}[\sigma(i)])^{\xi\omega}\}$ and $\{\mathsf{H}((R_{\sigma(i)} \boldsymbol{B}[\sigma(i)])^{\omega})\}$ instead. S_a could still eliminate the ξ and compute the indices of relevant signals after permutation. The second execution would be changed similarly.

Now, let us look at the details of the ZKP protocol. Assuming the list of clues are $c_1, \ldots, c_N$, where each is a tuple of two group elements. In this case, S_a sends to S_b, for each i, $(R_i \boldsymbol{A}[i])^{\xi}$, where $\boldsymbol{A}[i] = c_i[1]^{\mathsf{sk}_a}$; thus we can write the group element S_a sends as $\boldsymbol{U}[i] := R_i^{\xi} c_i[1]^{\mathsf{sk}_a \xi}$. Since R_i and $c_i[1]$ are public, we essentially want to prove in ZK a variation of the discrete logarithm: the prover has a witness consisting of two exponents, namely $x = \xi$ and $y = \mathsf{sk}_a \xi$, and the prover wants to prove that the publicly claimed values $\boldsymbol{U}[i] = R_i^{\xi} \cdot c_i[1]^{\mathsf{sk}_a \xi}$ holds for all i. It turns out that this can be proven efficiently using a variation of the Schnorr's protocol.

As a result, our protocol is highly efficient even in the malicious setting. The servers exchange 458 bytes with each other for each message on the board, and send 33.57 KB to the client when retrieving 50 pertinent signals.

3 Preliminaries

3.1 Notation

Let κ be the computational security parameter and λ be the statistical security parameter. For an ElGamal ciphertext $c := (g^r, \mathsf{pk}^r \cdot m)$, we denote the first component g^r as $c[1]$, and the second component $\mathsf{pk}^r \cdot m$ as $c[2]$. Decryption with the secret key sk is written as $\mathsf{Dec}_{\mathsf{sk}}(c) = c[2]/c[1]^{\mathsf{sk}}$. We use $x \xleftarrow{\$} S$ to denote sampling x uniformly from a set S. We use $[N]$ to denote the set $\{1, 2, \ldots, N\}$. We use bold upper-case letters like $\boldsymbol{A}$ to denote a vector and $\boldsymbol{A}[i]$ to denote the i-th component of $\boldsymbol{A}$. If the vector $\boldsymbol{A}$ is of size N, we use $\boldsymbol{A}^{\xi}$ to denote the set $\{\boldsymbol{A}[1]^{\xi}, \ldots, \boldsymbol{A}[N]^{\xi}\}$. Given a permutation σ of size N, we use $\sigma(\boldsymbol{A})$ to denote the set $\{\boldsymbol{A}[\sigma(1)], \ldots, \boldsymbol{A}[\sigma(N)]\}$.

3.2 Ideal Functionality $\mathcal{F}_{\mathsf{retrieve}}$

We define the functionality $\mathcal{F}_{\mathsf{retrieve}}$ in Figure 1. Given a list of signals in the form of distinct ElGamal ciphertexts (by saying "distinct", that both components of

Functionality $\mathcal{F}_{\mathsf{retrieve}}$

Participants: Two servers S_a, S_b, and a recipient R.

Retrieving signals: Given N distinct ElGamal ciphertexts $(c_1, \ldots, c_N)$ obtained from the ledger, the functionality receives $(\mathsf{Retrieve}, \mathsf{sk}_R)$ from a recipient R, and calculates the set $\mathcal{P} := \{i \,|\, \mathsf{Dec}_{\mathsf{sk}_R}(c_i) = 1\}$. It then sends $(\mathsf{Retrieve}, \mathcal{P})$ to R and $(\mathsf{Retrieve}, |\mathcal{P}|)$ to the adversary and servers.

Fig. 1. The ideal functionality for retrieval.

the two ElGamal ciphertexts differ), a recipient R can query $\mathcal{F}_{\mathsf{retrieve}}$ to help retrieve the pertinent signals for them. The recipient sends their own secret key to $\mathcal{F}_{\mathsf{retrieve}}$, and the functionality will then collect all the indices i for which the i-th ciphertext decrypts to 1, and return the list to the recipient. Also, the functionality will notify the adversary that a private list of signals has been accessed, including the size of the list $|\mathcal{P}|$. Revealing $|\mathcal{P}|$ is the result of a trade-off between security and efficiency. The similar trade-offs also exist in the previous works. Specifically, to improve the efficiency, the works [16, 17, 18, 20] limit the number of pertinent messages for a receiver (e.g., up to 50), which allows a malicious sender to easily send a large number of messages to launch a DoS attack [3]. Jia et al. [14] pointed out and eliminated this limitation, but at the cost of $O(N)$ server-client communication cost (i.e., digest size). Our work retains all the merits of Jia et al. [14], and chooses to reveal $|\mathcal{P}|$ to avoid the $O(N)$ cost. Note that the servers in our design cannot link different requests from the same receiver [4], and thus cannot deduce the receiver-message linkability based on the difference in the numbers of messages retrieved across requests.

In addition, our solution can be lightly modified to hide $|\mathcal{P}|$ but with $O(N)$ server-client communication cost. We present our volume-hiding protocol in Appendix A, with only one-step difference with our main protocol. With board size $N = 2^{19}$, the total server-client communication for this active protocol is 48MB (which is still smaller than FHE-based construction with the detection key of size 140MB) and the overhead on the reconstruction time (the time for the client to learn the pertinent indices) is only 10ms, comparing to our main protocol.

Private Signaling in One Step. Given the functionality $\mathcal{F}_{\mathsf{retrieve}}$, private signaling can be accomplished in a single step under the DDH assumption. In this system, each client generates an ElGamal key pair and publishes their public key for potential contactors. To send a message, the sender encrypts the value 1

[3] Liu and Tromer [17] discussed that to address the overflow issue, the receiver can (1) repeatedly request retrieval with a larger upper bound or (2) request the number of pertinent messages with a server and then retrieve the messages with another server. These methods degrade the performance and lead to information leakage, as discussed in their paper.

[4] We do not consider the network-level leakage as in the previous works [16, 17, 18, 20], which is beyond the scope of our paper.

Functionality $\mathcal{F}_{\mathsf{Ledger}}$

This functionality is globally available to all participants.

Parameter:

- state: initialized as ϵ;
- Validate(ctx, id, state): verify if a transaction is valid;

Functionality:

- Submit: Upon receiving (SUBMIT, ctx, id):
  1. If Validate(ctx, id, state)=1, then state = state $\cup$ (ctx, id);
  2. Send (SUBMIT, ctx, id) to $\mathcal{A}$;
- Read: Upon receiving READ from a party P_i, send state to P_i and $\mathcal{A}$.

Fig. 2. Ledger functionality.

using the recipient's public key and posts the ciphertext on a public ledger. The recipient can then retrieve relevant messages by either decrypting all ciphertexts on the board or by providing their secret key to $\mathcal{F}_{\mathsf{retrieve}}$.

3.3 Threat Model

We assume at least one of the server is honest and can not be colluded. Furthermore, we allow any collusion between recipients and one of the servers and prove that we achieve correctness and privacy against malicious adversaries. Correctness means that a recipient R should learn all signals that are intended for them, and be able to abort with overwhelming probabilities if the signals are not correct. Privacy means that by looking at the messages exchanged in the protocol no one except R and the senders of the signals should distinguish which signals are directed to R. Furthermore, we want to capture the inherent leakage that a protocol participant can learn that a certain recipient is trying to retrieve their own signals, as well as the number of signals being retrieved.

3.4 Ledger

A bulletin board can be abstracted as a global ledger, which can be achieved by blockchain techniques. Any party can submit messages to the ledger and read the state of the ledger. The underlying consensus protocols can guarantee that all the parties can read the same state. We adopt the global ledger definition in [15], and we give an abridged version in Figure 2.

Protocol $\Pi_{\mathsf{privSignal}}$

Public inputs: Two random oracles H and H'.

Participants: Two servers S_a and S_b, multiple clients that can act as senders or recipients.

Setup: To register in the system, the client R samples $\mathsf{sk}_R \xleftarrow{\$} \mathbb{Z}_p$, and calculates $\mathsf{pk}_R := g^{\mathsf{sk}_R}$, which will be sent to parties from which R is willing to receive messages.

Send: To inform a new message with id $\mathsf{msg\_id}$ to a recipient R, the sender obtains the public key of R, generates the clue $c := (g^r, \mathsf{pk}_R^r)$ with randomness r, and sends $(\mathsf{SUBMIT}, c, \mathsf{msg\_id})$ to $\mathcal{F}_{\mathsf{Ledger}}$.

Retrieve (Semi-Honest): Given $(c_1, c_2, \ldots, c_N)$ as the clue vector containing N distinct ElGamal ciphertexts, to perform a signal retrieval:

1. The recipient R samples $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$, and calculates $(\mathsf{sk}_b, \sigma) \leftarrow \mathsf{H}(\mathsf{seed})$, and $\mathsf{sk}_a = \mathsf{sk}_R - \mathsf{sk}_b$. R sends sk_a to S_a, and seed to S_b;
2. S_a samples $\xi \xleftarrow{\$} \mathbb{Z}_p$, calculates and sends $\boldsymbol{M} := (\mathsf{H}(c_1[1]^{\mathsf{sk}_a})^\xi, \ldots, \mathsf{H}(c_N[1]^{\mathsf{sk}_a})^\xi)$ to S_b;
3. S_b computes $(\mathsf{sk}_b, \sigma) \leftarrow \mathsf{H}(\mathsf{seed})$ and samples $\omega \xleftarrow{\$} \mathbb{Z}_p$. It then computes $\boldsymbol{P} := \sigma(\boldsymbol{M}^\omega)$ and $\boldsymbol{P'} := \sigma(\mathsf{H}'(\mathsf{H}(c_1[2]/c_1[1]^{\mathsf{sk}_b})^\omega), \ldots, \mathsf{H}'(\mathsf{H}(c_N[2]/c_N[1]^{\mathsf{sk}_b})^\omega))$ and sends to S_a;
4. S_a computes $\boldsymbol{E} := \{i \mid \mathsf{H}'(\boldsymbol{P}[i]^{\xi^{-1}}) = \boldsymbol{P'}[i]\}$, and sends to R;
5. R recovers the pertinent set $\mathcal{P} := \sigma^{-1}(\boldsymbol{E})$.

Fig. 3. Our semi-honest protocol for private signaling.

4 One-Face: Semi-Honest 2-Server Private Signaling

We break our semi-honest protocol for private signaling (Figure 3) into three parts: (1) Setup: how clients initialize; (2) Send: how a sender sends a signal; (3) Retrieve: how a recipient obliviously retrieves signals.

4.1 Setup

In our protocol, there are two servers and multiple clients that can act as senders or recipients. In the setup phase, clients generate ElGamal key pairs to join the system at any time. To register in the system, the client R uniformly samples sk_R from $\mathbb{Z}_p$, and then calculates its public key as $\mathsf{pk}_R := g^{\mathsf{sk}_R}$. To receive messages, R must share its public key pk_R with the prospective sender.

4.2 Sending Messages

To inform R of a new message of id $\mathsf{msg\_id}$, the sender should first obtain the public key of R, and use it to construct a valid ElGamal encryption of value 1.

Namely, the sender samples $r \xleftarrow{\$} \mathbb{Z}_p$ and calculates $c := (g^r, \mathsf{pk}_R^r)$ as the clue of this signal. Then the sender publishes the clue on the bulletin board, which will be further read and processed by two servers for retrieval.

4.3 Retrieval

Assuming two servers are semi-honest, we now describe how a recipient R, who holds the secret key sk_R, retrieves the pertinent indices. As previously defined, a valid clue c that is pertinent to R must be an ElGamal encryption of the value 1 under R's public key, i.e. $c[1]^{\mathsf{sk}_R} = c[2]$. Given $(\mathsf{sk}_a, \mathsf{sk}_b)$ as the additive shares of R's secret key, we have

$$c[1]^{\mathsf{sk}_a} = c[2]/c[1]^{\mathsf{sk}_b}.$$

To compute the pertinent indices, two servers first distributively decrypt all the clues, such that S_a holds $c_i[1]^{\mathsf{sk}_a}$ and S_b holds $c_i[2]/c_i[1]^{\mathsf{sk}_b}$. Then two servers jointly compute the permuted indices satisfying $c_i[1]^{\mathsf{sk}_a} = c_i[2]/c_i[1]^{\mathsf{sk}_b}$. Since the result corresponds to the permutation chosen by R, R can easily reconstruct the original pertinent set.

Request a Retrieval. To request a retrieval, R should privately send to S_a: sk_a - one share of its own secret key, and to S_b: sk_b - another share of the secret key, and σ - the permutation to mask the pertinent indices. (sk_b, σ) can be derived using a seed and a random oracle H.

Response to a Retrieval Request. Given $(c_1, c_2, \ldots, c_N)$ as the clue vector obtained from $\mathcal{F}_{\mathsf{Ledger}}$, the goal for two servers is to inform the recipient all the indices i satisfying

$$c_i[1]^{\mathsf{sk}_a} = c_i[2]/c_i[1]^{\mathsf{sk}_b}.$$

Holding sk_a or sk_b, each server can calculate one side of the equation. Therefore, to obliviously return the index, two servers should perform a private equality check.

To mask the input, each server would sample a random exponent: S_a samples $\xi \xleftarrow{\$} \mathbb{Z}_p$, and S_b samples $\omega \xleftarrow{\$} \mathbb{Z}_p$. First, S_a calculates a masked vector

$$\boldsymbol{M} := (\mathsf{H}(c_1[1]^{\mathsf{sk}_a})^{\xi}, \ldots, \mathsf{H}(c_N[1]^{\mathsf{sk}_a})^{\xi}),$$

and sends to S_b. S_b then permutes and blinds both $\boldsymbol{M}$ and its own inputs using ω, resulting the following two vectors $\boldsymbol{P} := \sigma(\boldsymbol{M}^{\omega})$, and $\boldsymbol{P'} := \sigma(\mathsf{H}'(\mathsf{H}(c_1[2]/c_1[1]^{\mathsf{sk}_b})^{\omega}), \ldots, \mathsf{H}'(\mathsf{H}(c_N[2]/c_N[1]^{\mathsf{sk}_b})^{\omega}))$. $\boldsymbol{P}$ and $\boldsymbol{P'}$ are returned to S_a, who can remove its own blinding exponent and output the equality set $\boldsymbol{E} := \{i \,|\, \mathsf{H}'(\boldsymbol{P}[i]^{\xi^{-1}}) = \boldsymbol{P'}[i]\}$ to R.

Assuming the existence of random oracle, and the randomness of ξ, $\boldsymbol{M}$ is computationally indistinguishable from a random vector under the DDH assumption. Therefore, S_b learns nothing from the vector $\boldsymbol{M}$. Likewise, assuming the randomness of σ, S_a gains no information from $\boldsymbol{P}$ and $\boldsymbol{P'}$, other than the number of pertinent indices.

Functionality $\mathcal{F}_{\mathsf{BatchPOE}}$

On public inputs $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) \in \mathbb{G}^{3N}$, and $(x, y) \in \mathbb{Z}_P^2$ from the prover $\mathcal{P}$, return to both parties whether $\boldsymbol{C}[i] = \boldsymbol{A}[i]^x \boldsymbol{B}[i]^y$ for all $i \in [N]$.

Fig. 4. The ideal functionality for batch proof of exponent.

Reconstruct the Pertinent Set. To reconstruct the pertinent indices $\mathcal{P}$, R simply computes $\sigma^{-1}(\boldsymbol{E})$. We argue that assuming two servers act honestly, $\sigma^{-1}(\boldsymbol{E})$ is exactly the pertinent set $\mathcal{P}$.

Lemma 1. *(Correctness) Given $(c_1, \ldots, c_N)$ as the clue vector obtained from $\mathcal{F}_{\mathsf{Ledger}}$, a recipient R participating $\Pi_{\mathsf{privSignal}}.\mathsf{Retrieve}$(Figure 3) learns the set of pertinent signals $\mathcal{P} = \{i \mid \mathsf{Dec}_{\mathsf{sk}_R}(c_i) = 1\}$ in the random oracle model, assuming two servers behave honestly.*

Proof. If $\mathsf{Dec}_{\mathsf{sk}_R}(c_i) = 1$, given $\mathsf{sk}_R = \mathsf{sk}_a + \mathsf{sk}_b$, we have $\mathsf{H}'(\boldsymbol{P}[\sigma(i)]^{\xi^{-1}}) = \mathsf{H}'(\mathsf{H}(c_i[1]^{\mathsf{sk}_a})^{\xi\omega\cdot\xi^{-1}}) = \mathsf{H}'(\mathsf{H}(c_i[2]/c_i[1]^{\mathsf{sk}_b})^{\omega}) = \boldsymbol{P'}[\sigma(i)]$. According to the protocol, $\sigma(i)$ will be added to $\boldsymbol{E}$, and thus $\mathcal{P} \subseteq \sigma^{-1}(\boldsymbol{E})$.

Similarly, for any $\sigma(i) \in \boldsymbol{E}$, we have $\mathsf{H}'(\boldsymbol{P}[\sigma(i)]^{\xi^{-1}}) = \mathsf{H}'(\mathsf{H}(c_i[1]^{\mathsf{sk}_a})^{\xi\omega\cdot\xi^{-1}}) = \mathsf{H}'(\mathsf{H}(c_i[2]/c_i[1]^{\mathsf{sk}_b})^{\omega}) = \boldsymbol{P'}[\sigma(i)]$, which means that $c_i[1]^{\mathsf{sk}_a} = c_i[2]/c_i[1]^{\mathsf{sk}_b}$. Given $\mathsf{sk}_R = \mathsf{sk}_a + \mathsf{sk}_b$, we have $\mathsf{Dec}_{\mathsf{sk}_R}(c_i) = 1$, and thus $\sigma^{-1}(\boldsymbol{E}) \subseteq \mathcal{P}$.

Theorem 1. *$\Pi_{\mathsf{privSignal}}.\mathsf{Retrieve}$ in Figure 3 securely realizes $\mathcal{F}_{\mathsf{Retrieve}}$ in the random oracle model in the semi-honest setting, assuming the hardness of DDH.*

The proof of Theorem 1 can be found in Appendix B.

5 Two-Face: Malicious 2-Server Private Signaling

In this section, we present our malicious protocol for signal retrieval (Figure 6). Assuming the server may act maliciously, the correctness guarantees of the semi-honest protocol no longer hold since: (1) the malicious server may switch inputs that are not consistent with the bulletin board, and (2) after computing the intersection, S_a may maliciously drop certain indices when sending to R.

We address the first problem using a customized Batch Proof of Exponent, and to tackle the second problem, we let two servers perform a dual execution, allowing the recipient to efficiently check consistency.

5.1 Batch Proof of Exponent for Input Authentication

Recall that in the semi-honest protocol, S_a blinds its input with a hash function and a random ξ_a:

$$\boldsymbol{M} := (\mathsf{H}(c_1[1]^{\mathsf{sk}_a})^{\xi_a}, \ldots, \mathsf{H}(c_N[1]^{\mathsf{sk}_a})^{\xi_a}).$$

Protocol Π_{BatchPOE}

Public inputs: $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) \in \mathbb{G}^{3N}$, and a random oracle H.

Private inputs: $x, y \in \mathbb{Z}_P^2$ from the prover $\mathcal{P}$.

Prove:

1. Calculate $\tilde{A}, \tilde{B} \leftarrow \mathsf{H}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$, where $(\tilde{A}, \tilde{B}) \in \mathbb{G}^2$;
2. Compute $\tilde{C} := \tilde{A}^x \tilde{B}^y$;
3. Calculate $\boldsymbol{q} \leftarrow \mathsf{H}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \tilde{A}, \tilde{B}, \tilde{C})$ where $\boldsymbol{q} \in \mathbb{Z}_p^{N+1}$, and compute

$$A := \tilde{A}^{\boldsymbol{q}[N+1]} \cdot \prod_{i \in [N]} \boldsymbol{A}[i]^{\boldsymbol{q}[i]},$$

$$B := \tilde{B}^{\boldsymbol{q}[N+1]} \cdot \prod_{i \in [N]} \boldsymbol{B}[i]^{\boldsymbol{q}[i]},$$

$$C := \tilde{C}^{\boldsymbol{q}[N+1]} \cdot \prod_{i \in [N]} \boldsymbol{C}[i]^{\boldsymbol{q}[i]};$$

4. Choose blinding factors $k_1, k_2 \xleftarrow{\$} \mathbb{Z}_p$ and compute $t = A^{k_1} B^{k_2}$;
5. Calculate a random challenge $e \leftarrow \mathsf{H}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \tilde{A}, \tilde{B}, \tilde{C}, \boldsymbol{q}, t)$ where $e \in \mathbb{Z}_p$;
6. Compute $z_1 = k_1 + xe$, and $z_2 = k_2 + ye$ and send proof $\pi := (\tilde{C}, t, z_1, z_2)$.

Verify: The verifier $\mathcal{V}$ parses π as $(\tilde{C}, t, z_1, z_2)$:

1. Calculate $\tilde{A}, \tilde{B} \leftarrow \mathsf{H}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$, $\boldsymbol{q} \leftarrow \mathsf{H}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \tilde{A}, \tilde{B}, \tilde{C})$, and

$$A := \tilde{A}^{\boldsymbol{q}[N+1]} \cdot \prod_{i \in [N]} \boldsymbol{A}[i]^{\boldsymbol{q}[i]},$$

$$B := \tilde{B}^{\boldsymbol{q}[N+1]} \cdot \prod_{i \in [N]} \boldsymbol{B}[i]^{\boldsymbol{q}[i]},$$

$$C := \tilde{C}^{\boldsymbol{q}[N+1]} \cdot \prod_{i \in [N]} \boldsymbol{C}[i]^{\boldsymbol{q}[i]};$$

2. Compute $e \leftarrow \mathsf{H}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \tilde{A}, \tilde{B}, \tilde{C}, \boldsymbol{q}, t)$;
3. Check whether $A^{z_1} B^{z_2} = t \cdot C^e$.

Fig. 5. Our protocol for batch proof of exponent.

To prove that the input does not deviate from the clues $(c_1, \ldots, c_N)$, S_a should demonstrate that each element of $\boldsymbol{M}$ is calculated correctly. However, since the proof of hash function is generally considered expensive [2], we apply another method to blind S_a's input:

$$\boldsymbol{M} := (R_1^{\xi_a}(c_1[1]^{\mathsf{sk}_a})^{\xi_a}, \ldots, R_N^{\xi_a}(c_N[1]^{\mathsf{sk}_a})^{\xi_a}).$$

Assuming the randomness of $(R_1, \ldots, R_N)$, $\boldsymbol{M}$ is computationally indistinguishable from a random vector under the DDH assumption. To prevent a malicious server from tampering with the randomness, the recipient provides a random seed used to generate $(R_1, \ldots, R_N)$.

The honest construction of $\boldsymbol{M}$ can now be verified using a batch proof of exponent, of which the functionality $\mathcal{F}_{\mathsf{BatchPOE}}$ is presented in Figure 4. Using a batch proof of exponent, given three vectors $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ as public input, the prover can convince the verifier that each element in the vector $\boldsymbol{C}$ is computed from the corresponding element in vector $\boldsymbol{A}$ and $\boldsymbol{B}$ using same exponents x and y, i.e. $\boldsymbol{C}[i] = \boldsymbol{A}[i]^x \boldsymbol{B}[i]^y$, without revealing the value of either x or y.

We present our implementation for this functionality in Figure 5, based on the protocol in [25]. The original protocol is secure in RSA group, but is not sound in a prime group if the prover previously knows the discrete logarithms of all public inputs. We fix this issue by challenging the prover at random group elements $\tilde{A}$ and $\tilde{B}$, based on the verifier's choice.

Lemma 2. *The protocol Π_{BatchPOE} securely realizes $\mathcal{F}_{\mathsf{BatchPOE}}$ with soundness error $|\mathbb{G}|^{-1}$ in the random oracle model, assuming the hardness of the discrete logarithm problem.*

The proof for Lemma 2 can be found in Appendix C.

To convince S_b that the vector is honestly constructed from $(R_1, \ldots, R_N, c_1, \ldots, c_N)$, S_a sends ξ_a and $\mathsf{sk}_a \xi_a$ to Π_{BatchPOE}, which will return to S_b whether $\boldsymbol{M}[i] = R_i^{\xi_a} c_i[1]^{\mathsf{sk}_a \xi_a}$ for $i \in [N]$. To ensure that S_a is using the exact sk_a and ξ_a that R provides, R will public $(g_a, h_a, g_a^{\xi_a} h_a^{\mathsf{sk}_a \xi_a})$, which will also be sent to Π_{BatchPOE}, serving as a proving gadget.

To ensure correctness, the construction of $\boldsymbol{P'}$ changes consequently: $\boldsymbol{P'} := \sigma(\mathsf{H}(R_1^{\omega}(c_1[2]/c_1[1]^{\mathsf{sk}_b})^{\omega}), \ldots, \mathsf{H}(R_N^{\omega}(c_N[2]/c_N[1]^{\mathsf{sk}_b})^{\omega}))$. After receiving $\boldsymbol{P}$ and $\boldsymbol{P'}$ from S_b, S_a computes $\boldsymbol{E} := \{i \mid \mathsf{H}(\boldsymbol{P}[i]^{\xi^{-1}}) = \boldsymbol{P'}[i]\}$.

5.2 Dual Execution for Mutual Integrity Checking

Instead of authenticating vector $\boldsymbol{P}$ and $\boldsymbol{P'}$ (which will involve proof of permutation relative to the size of the bulletin board), we let two servers perform a dual execution, such that the recipient can later compare the results from two servers and check their consistency. In the remainder of this paper, we use $\boldsymbol{M}_a$ to denote a vector generated by S_a, and $\boldsymbol{M}_b$ to denote a vector generated by S_b.

Recall that in the semi-honest protocol, in order to filter out all the indices i such that

$$c_i[1]^{\mathsf{sk}_a} = c_i[2]/c_i[1]^{\mathsf{sk}_b},$$

S_a calculates the LHS of the equation for $i \in [N]$, blinds the vector using a random exponent ξ_a, and sends to S_b (denoted as $\boldsymbol{M}_a$). Both servers then call $\mathcal{F}_{\mathsf{BatchPOE}}$ to authenticate the honest construction of the vector $\boldsymbol{M}_a$. After receiving $\boldsymbol{P}_b$ and $\boldsymbol{P}'_b$, S_a computes $\boldsymbol{E}_a := \{i \mid \mathsf{H}(\boldsymbol{P}_b[i]^{\xi_a^{-1}}) = \boldsymbol{P}'_b[i]\}$ and sends to the recipient R.

To perform a dual execution, two servers switch roles to filter out all the indices i such that

$$c_i[1]^{\mathsf{sk}_b} = c_i[2]/c_i[1]^{\mathsf{sk}_a}.$$

In this dual execution, S_b will calculate the LHS of the equation for $i \in [N]$, blind it using another random exponent ξ_b, and send to S_a:

$$\boldsymbol{M}_b := (R_1^{\xi_b} c_1[1]^{\mathsf{sk}_b \xi_b}, \ldots, R_N^{\xi_b} c_N[1]^{\mathsf{sk}_b \xi_b}).$$

Similarly, both servers call $\mathcal{F}_{\mathsf{BatchPOE}}$ to authenticate the honest construction of the vector $\boldsymbol{M}_b$. After receiving $\boldsymbol{P}_a$ and $\boldsymbol{P}'_a$, S_b computes $\boldsymbol{E}_b := \{i \mid \mathsf{H}(\boldsymbol{P}_a[i]^{\xi_b^{-1}}) = \boldsymbol{P}'_a[i]\}$ and sends to R. R can then reverse the permutation and check whether two results are consistent.

Although both servers can cheat when providing $\boldsymbol{P}_*$ and $\boldsymbol{P}'_*$, it is hard for one server to ensure the consistency between $\boldsymbol{E}_a$ and $\boldsymbol{E}_b$. Without loss of generality, assume that S_a maliciously calculates $\boldsymbol{P}_a$ and $\boldsymbol{P}'_a$ to change a certain index in $\boldsymbol{E}_b$, denoted as i. When providing $\boldsymbol{E}_a$ to R, S_a has to guess $\sigma_b(i)$ without the knowledge of the permutation, otherwise R will abort due to the incosistency of two results.

Repetition and Padding to Mitigate False Negatives. To increase the difficulty to maintain the output consistency, each server applies the permutation independently k times and the other server would output k corresponding results. Additionally, R will pre-pad p values to help detect any malicious behavior by the servers.

More specifically, before retrieving signals, R informs both servers of the padding values $(\gamma_1, \ldots, \gamma_p)$, which are also generated using a public seed. During retrieval, S_a calculates and sends

$$\boldsymbol{M}_a := (R_1^{\xi_a} c_1[1]^{\mathsf{sk}_a \xi_a}, \ldots, R_N^{\xi_a} c_N[1]^{\mathsf{sk}_a \xi_a}, \gamma_1^{\xi_a}, \ldots, \gamma_p^{\xi_a}).$$

After verifying the construction of $\boldsymbol{M}_a$, S_b applies k different permutations and blinding factors on $\boldsymbol{M}_a$ as well as its own inputs, resulting in $2k$ vectors: For $i \in [k]$, S_b samples $\omega_{b,i} \xleftarrow{\$} \mathbb{Z}_p$, and calculates $\boldsymbol{P}_{b,i} := \sigma_{b,i}(\boldsymbol{M}_a^{\omega_{b,i}})$, and $\boldsymbol{P}'_{b,i} := \sigma_{b,i}(\mathsf{H}(R_1^{\omega_{b,i}}(c_1[2]/c_1[1]^{\mathsf{sk}_b})^{\omega_{b,i}}), \ \ldots, \mathsf{H}(R_N^{\omega_{b,i}}(c_N[2]/c_N[1]^{\mathsf{sk}_b})^{\omega_{b,i}}), \mathsf{H}(\gamma_1^{\omega_{b,i}}), \ldots, \mathsf{H}(\gamma_p^{\omega_{b,i}}))$. Then S_b sends all the $2k$ vectors to S_a, who can remove blinding exponent ξ_a and output k results $\{\boldsymbol{E}_{a,i}\}_{i \in [k]}$.

To reconstruct the pertinent indices, R applies the inverse of each permutation to verify consistency and ensure that the padding values are within the set. That is, R checks whether for all $i, j \in [k]$ st. $[N+1, N+p] \subset \sigma_{b,i}^{-1}(\boldsymbol{E}_{a,i}) = \sigma_{a,j}^{-1}(\boldsymbol{E}_{b,j})$. If the equation holds, we argue that $\sigma_{b,1}^{-1}(\boldsymbol{E}_{a,1})$ is the set of pertinent indices with overwhelming probabilities.

Lemma 3. *(False negative probability) Given $(c_1, \ldots, c_N)$ as the clue vector obtained from $\mathcal{F}_{\mathsf{Ledger}}$, an honest recipient R participating $\Pi_{\mathsf{privSignal}}.\mathsf{Retrieve}$ (Figure 6) either learns the set of pertinent signals $\mathcal{P} = \{i \mid \mathsf{Dec}_{\mathsf{sk}_R}(c_i) = 1\}$ or the protocol aborts with probability greater than $1 - \frac{1}{(p+|\mathcal{P}|)^k}$ in the random oracle model, assuming the hardness of DDH and $p > N^{\frac{N}{N-1}} - N$.*

Protocol $\Pi_{\mathsf{privSignal}}$

Setup & Send: Same as the semi-honest version (Figure 3).

Retrieve (Malicious): Given $(c_1, c_2, \ldots, c_N)$ as the clue vector containing N distinct ElGamal ciphertexts, H as a random oracle, to perform a signal retrieval:

1. R samples $\mathsf{seed}, \mathsf{seed}_a, \mathsf{seed}_b \xleftarrow{\$} \{0,1\}^\lambda$, and calculates $(g_a, g_b, h_a, h_b) \leftarrow \mathsf{H}(\mathsf{seed})$, $(\mathsf{sk}_a, \xi_a) \leftarrow \mathsf{H}(\mathsf{seed}_a)$, $\xi_b \leftarrow \mathsf{H}(\mathsf{seed}_b)$, and $\mathsf{sk}_b = \mathsf{sk}_R - \mathsf{sk}_a$;
2. R sends $(\mathsf{seed}, g_a^{\xi_a}, g_b^{\xi_b}, h_a^{\mathsf{sk}_a\xi_a}, h_b^{\mathsf{sk}_b\xi_b})$ to two servers, (seed_a) to S_a, and $(\mathsf{sk}_b, \mathsf{seed}_b)$ to S_b. Two servers compare $\mathsf{H}(\mathsf{seed}, g_a^{\xi_a}, g_b^{\xi_b}, h_a^{\mathsf{sk}_a\xi_a}, h_b^{\mathsf{sk}_b\xi_b})$ and abort if not consistent;
3. Both servers compute $(g_a, g_b, h_a, h_b, R_1, \ldots, R_N, \gamma_1, \ldots, \gamma_p) \leftarrow \mathsf{H}(\mathsf{seed})$. S_a computes $(\mathsf{sk}_a, \xi_a, \{\sigma_{a,i}\}_{i\in[k]}) \leftarrow \mathsf{H}(\mathsf{seed}_a)$, and S_b computes $(\xi_b, \{\sigma_{b,i}\}_{i\in[k]}) \leftarrow \mathsf{H}(\mathsf{seed}_b)$;
4. Two servers proceed as follows to calculate all the indices $i \in [N]$ satisfying $c_i[1]^{\mathsf{sk}_a} = c_i[2]/c_i[1]^{\mathsf{sk}_b}$:
   - S_a calculates and sends $\boldsymbol{M}_a := (R_1^{\xi_a} c_1[1]^{\mathsf{sk}_a\xi_a}, \ldots, R_N^{\xi_a} c_N[1]^{\mathsf{sk}_a\xi_a}, \gamma_1^{\xi_a}, \ldots, \gamma_p^{\xi_a})$ to S_b;
   - Given public value $(R_1, \ldots, R_N, g_a, c_1, \ldots, c_N, h_a, \boldsymbol{M}_a, g_a^{\xi_a} h_a^{\mathsf{sk}_a\xi_a})$, S_a invokes $\mathcal{F}_{\mathsf{BatchPOE}}$ with private input ξ_a and $\mathsf{sk}_a\xi_a$ to prove that for $i \in [N]$, $\boldsymbol{M}_a[i] = R_i^{\xi_a} c_i[1]^{\mathsf{sk}_a\xi_a}$. S_b aborts if check fails;
   - Given public value $(\gamma_1, \ldots, \gamma_p, g_a, \boldsymbol{M}_a, g_a^{\xi_a})$, S_a invokes $\mathcal{F}_{\mathsf{BatchPOE}}$ with private input ξ_a to prove that for $i \in [p]$, $\boldsymbol{M}_a[N+i] = \gamma_i^{\xi_a}$. S_b aborts if check fails;
   - For $i \in [k]$:
     - S_b samples $\omega_{b,i} \xleftarrow{\$} \mathbb{Z}_p$, and sends $\boldsymbol{P}_{b,i} := \sigma_{b,i}(\boldsymbol{M}_a^{\omega_{b,i}})$ and $\boldsymbol{P}'_{b,i} := \sigma_{b,i}(\mathsf{H}(R_1^{\omega_{b,i}}(c_1[2]/c_1[1]^{\mathsf{sk}_b})^{\omega_{b,i}}), \ldots, \mathsf{H}(R_N^{\omega_{b,i}}(c_N[2]/c_N[1]^{\mathsf{sk}_b})^{\omega_{b,i}}), \mathsf{H}(\gamma_1^{\omega_{b,i}}), \ldots, \mathsf{H}(\gamma_p^{\omega_{b,i}}))$ to S_a;
     - S_a aborts if there are repeated elements in $\boldsymbol{P}_{b,i}$. Otherwise, compute $\boldsymbol{E}_{a,i} := \{j \mid \mathsf{H}(\boldsymbol{P}_{b,i}[j]^{\xi_a^{-1}}) = \boldsymbol{P}'_{b,i}[j]\}$;
5. Two servers switch the roles to calculate all the indices $i \in [N]$ satisfying $c_i[1]^{\mathsf{sk}_b} = c_i[2]/c_i[1]^{\mathsf{sk}_a}$:
   - S_b calculates and sends $\boldsymbol{M}_b := (R_1^{\xi_b} c_1[1]^{\mathsf{sk}_b\xi_b}, \ldots, R_N^{\xi_b} c_N[1]^{\mathsf{sk}_b\xi_b}, \gamma_1^{\xi_b}, \ldots, \gamma_p^{\xi_b})$ to S_a;
   - Given public value $(R_1, \ldots, R_N, g_b, c_1, \ldots, c_N, h_b, \boldsymbol{M}_b, g_b^{\xi_b} h_b^{\mathsf{sk}_b\xi_b})$, S_b invokes $\mathcal{F}_{\mathsf{BatchPOE}}$ with private input ξ_b and $\mathsf{sk}_b\xi_b$. S_a aborts if check fails;
   - Given public value $(\gamma_1, \ldots, \gamma_p, g_b, \boldsymbol{M}_b, g_b^{\xi_b})$, S_b invokes $\mathcal{F}_{\mathsf{BatchPOE}}$ with private input ξ_b. S_a aborts if check fails;
   - For $i \in [k]$:
     - S_a samples $\omega_{a,i} \xleftarrow{\$} \mathbb{Z}_p$, and sends $\boldsymbol{P}_{a,i} := \sigma_{a,i}(\boldsymbol{M}_b^{\omega_{a,i}})$ and $\boldsymbol{P}'_{a,i} := \sigma_{a,i}(\mathsf{H}(R_1^{\omega_{a,i}}(c_1[2]/c_1[1]^{\mathsf{sk}_a})^{\omega_{a,i}}), \ldots, \mathsf{H}(R_N^{\omega_{a,i}}(c_N[2]/c_N[1]^{\mathsf{sk}_a})^{\omega_{a,i}}), \mathsf{H}(\gamma_1^{\omega_{a,i}}), \ldots, \mathsf{H}(\gamma_p^{\omega_{a,i}}))$ to S_b;
     - S_b aborts if there are repeated elements in $\boldsymbol{P}_{a,i}$. Otherwise, compute $\boldsymbol{E}_{b,i} := \{j \mid \mathsf{H}(\boldsymbol{P}_{a,i}[j]^{\xi_b^{-1}}) = \boldsymbol{P}'_{a,i}[j]\}$;
6. S_a sends $\{\boldsymbol{E}_{a,i}\}_{i\in[k]}$ to R, and S_b sends $\{\boldsymbol{E}_{b,i}\}_{i\in[k]}$ to R;
7. R aborts unless for all $i, j \in [k]$ s.t. $[N+1, N+p] \subset \sigma_{b,i}^{-1}(\boldsymbol{E}_{a,i}) = \sigma_{a,j}^{-1}(\boldsymbol{E}_{b,j})$.

Fig. 6. Our malicious protocol for private signaling.

Proof. Full proof is included in Appendix D and we provide a proof sketch below.

A malicious S_a can alter certain indices when sending $\boldsymbol{E}_{a,i}$ to R. Denote the set that a malicious S_a drops in $\boldsymbol{E}_{a,i}$ as ϕ_i, and $\{\phi_i\}_{i\in[k]}$ should satisfy that:

1. $\sigma_{b,i}^{-1}(\phi_i) \not\subset [N+1, N+p]$: without the knowledge of $\sigma_{b,i}$ (two servers are non-collude), ϕ_i should not touch the values padded by the recipient R;
2. $\sigma_{b,i}^{-1}(\phi_i) = \sigma_{b,j}^{-1}(\phi_j)$ for $i, j \in [k]$: each ϕ_i should be consistent after reversing the permutation;
3. S_a should correctly guess $\varphi = \sigma_{b,i}^{-1}(\phi_i)$ and modify these entries when constructing $\boldsymbol{P}_{a,i}$ and $\boldsymbol{P}'_{a,i}$, so that entries in φ will not appear in $\boldsymbol{E}_{b,i}$.

The problem can be modeled as a combinatorial problem in which the adversary can succeed in cheating only with probability less than $\frac{1}{(p+|\mathcal{P}|)^k}$. The detailed proof is presented in Appendix D.

Assuming $|\mathcal{P}| \to 0$, the probability that an honest recipient gets the pertinent set is sufficiently large as long as p^k exceeds 2^λ.

Theorem 2. *$\Pi_{\mathsf{privSignal}}.\mathsf{Retrieve}$ in Figure 6 securely realizes $\mathcal{F}_{\mathsf{Retrieve}}$ in the $\mathcal{F}_{\mathsf{BatchPOE}}$-hybrid and random oracle model in the active security setting, assuming the hardness of DDH.*

Proof. Since our protocol is strictly symmetric, we can assume that S_a is the adversary. Additionally, we consider a worst-case scenario in which the adversary colludes with the sender and has complete knowledge of the randomness used in all ciphertexts posted on the bulletin board. This captures a realistic threat model where the adversary may strategically construct and publish clues on the public ledger to serve their own advantage. We show there exists a PPT simulator Sim interacting with functionality $\mathcal{F}_{\mathsf{Retrieve}}$ generates a transcript that is indistinguishable from the transcript generated by the real-world adversary $\mathcal{A}$.

Case I. When one of the server is corrupted with the sender(s).

Given $(c_1, c_2, \ldots, c_N)$ as the clue vector:

1. Upon receiving $(\mathsf{Retrieve}, |\mathcal{P}|)$ from $\mathcal{F}_{\mathsf{retrieve}}$, Sim samples two seeds $\mathsf{seed}, \mathsf{seed}_a \xleftarrow{\$} \{0,1\}^\lambda$, $g_*, h_* \xleftarrow{\$} \mathbb{G}$, and computes $(g_a, g_b, h_a, h_b, R_1, \ldots, R_N, \gamma_1, \ldots, \gamma_p) \leftarrow \mathsf{H}(\mathsf{seed})$ and $(\mathsf{sk}_a, \xi_a, \{\sigma_{a,i}\}_{i\in[k]}) \leftarrow \mathsf{H}(\mathsf{seed}_a)$;
2. Send $(\mathsf{seed}, g_a^{\xi_a}, g_*, h_a^{\mathsf{sk}_a\xi_a}, h_*, \mathsf{seed}_a)$ to $\mathcal{A}$;
3. Compare $\mathsf{H}(\mathsf{seed}, g_a^{\xi_a}, g_*, h_a^{\mathsf{sk}_a\xi_a}, h_*)$ with $\mathcal{A}$ and abort if not consistent;
4. Upon receiving $\boldsymbol{M}_a$ from $\mathcal{A}$, Sim emulates $\mathcal{F}_{\mathsf{BatchPOE}}$ and receives x and y from $\mathcal{A}$. Sim aborts if $x \neq \xi_a$ or $y \neq \mathsf{sk}_a\xi_a$ or $R_i^x c_i[1]^y \neq \boldsymbol{M}_a[i]$ for any $i \in [N]$ or $\gamma_i^x \neq \boldsymbol{M}_a[i+N]$ for any $i \in [p]$;
5. Sample $\mu \xleftarrow{\$} [N+p]^{p+|\mathcal{P}|}$, and for $i \in [k]$: Sim samples $\boldsymbol{P}_{b,i}, \boldsymbol{P}'_{b,i} \xleftarrow{\$} \mathbb{G}^{N+M}$, and programs $\mathsf{H}(\boldsymbol{P}_{b,i}[j]^{\xi_a^{-1}}) = \boldsymbol{P}'_{b,i}[j]$ for $j \in \mu$. Send $(\boldsymbol{P}_{b,i}, \boldsymbol{P}'_{b,i})$ to $\mathcal{A}$;

6. Sample $\xi_b \xleftarrow{\$} \mathbb{Z}_p$, and send $\boldsymbol{M}_b \xleftarrow{\$} \mathbb{G}^{N+M}$ to $\mathcal{A}$ where

$$\boldsymbol{M}_b[j] := \begin{cases} R_j^{\xi_b}(c_j[2]/c_j[1]^{\mathsf{sk}_a})^{\xi_b}, j \in \mu \text{ and } j \in [N], \\ \gamma_{j-N}^{\xi_b}, j \in \mu \text{ and } j > N, \\ \xleftarrow{\$} \mathbb{G}, \text{otherwise}; \end{cases}$$

7. Sim emulates $\mathcal{F}_{\mathsf{BatchPOE}}$ and returns 1 to $\mathcal{A}$ twice;
8. For $i \in [k]$, upon receiving $(\boldsymbol{P}_{a,i}, \boldsymbol{P}'_{a,i})$ from $\mathcal{A}$, aborts if there are repeated elements in $\boldsymbol{P}_{a,i}$. Otherwise, compute $\boldsymbol{E}_{b,i} := \{j \mid \mathsf{H}(\boldsymbol{P}_{a,i}[j]^{\xi_b^{-1}}) = \boldsymbol{P}'_{a,i}[j]\}$;
9. Upon receiving $\{\boldsymbol{E}_{a,i}\}_{i\in[k]}$ from $\mathcal{A}$, abort unless $\mu = \boldsymbol{E}_{a,i} = \sigma_{a,j}^{-1}(\boldsymbol{E}_{b,j})$ for $i, j \in [k]$.

We present proof by hybrids to show that the simulated world and the real world are indistinguishable.

Hybrid 0. The real world protocol.

Hybrid 1. This hybrid is the same as the previous hybrid except that we use the internally simulated $\mathcal{F}_{\mathsf{BatchPOE}}$ ideal functionality.

Hybrid 2. This hybrid is the same as the previous hybrid except that Sim emulates the random oracle H and provides $\boldsymbol{P}_b$. Sim samples $\boldsymbol{P}_b^{\mathsf{hybrid}_2} \xleftarrow{\$} \mathbb{G}^{N+M}$ and programs $\mathsf{H}(\boldsymbol{P}_b[j]^{\xi_a^{-1}}) = \boldsymbol{P}'_b[j]$ for $j \in \mathcal{P}$. And we have

$$\mathsf{View}_{\mathsf{hybrid}_1} := \{\boldsymbol{M}, \boldsymbol{P}_b, \boldsymbol{P}'_b, \xi_a\},$$

and

$$\mathsf{View}_{\mathsf{hybrid}_2} := \{\boldsymbol{M}, \boldsymbol{P}_b^{\mathsf{hybrid}_2}, \boldsymbol{P}'_b, \xi_a\}.$$

Since under the random oracle model, $(R_1, \ldots, R_N)$ is uniformly distributed, we can conclude that both $\boldsymbol{M}$ and $\boldsymbol{P}_b'^{\xi_a}$ are also uniformly distributed. Under the DDH assumption, two views are indistinguishable.

Hybrid 3. This hybrid is the same as the previous hybrid except that $\boldsymbol{P}'_b$ is uniformly sampled. Define

$$\boldsymbol{P}_b'^{\mathsf{hybrid}_3} \xleftarrow{\$} \mathbb{G}^{N+p},$$

and accordingly programs the random oracle. Assuming the randomness of $(R_1, \ldots, R_N, \gamma_1, \ldots, \gamma_p)$ and the DDH assumption, the following two views are indistinguishable:

$$\mathsf{View}_{\mathsf{hybrid}_2} := \{R_1, \ldots, R_N, c_1, \ldots, c_N, \gamma_1, \ldots, \gamma_p, \boldsymbol{P}'_b\},$$

$$\mathsf{View}_{\mathsf{hybrid}_3} := \{R_1, \ldots, R_N, c_1, \ldots, c_N, \gamma_1, \ldots, \gamma_p, \boldsymbol{P}_b'^{\mathsf{hybrid}_3}\}.$$

Hybrid 4. This hybrid is the same as the previous hybrid except that $\boldsymbol{M}_b$ is provided by Sim. Define

$$\boldsymbol{M}_b^{\mathsf{hybrid}_4}[j] := \begin{cases} R_j^{\xi_b}(c_j[2]/c_j[1]^{\mathsf{sk}_a})^{\xi_b}, j \in \mathcal{P}, \\ \gamma_{j-N}^{\xi_b}, j > N, \\ \xleftarrow{\$} \mathbb{G}, \text{otherwise}; \end{cases}$$

Assuming the randomness of $(R_1, \ldots, R_N)$ and the DDH assumption, the following two views are indistinguishable:

$$\mathsf{View}_{\mathsf{hybrid}_3} := \{R_1, \ldots, R_N, c_1, \ldots, c_N, \boldsymbol{M}_b\},$$

$$\mathsf{View}_{\mathsf{hybrid}_4} := \{R_1, \ldots, R_N, c_1, \ldots, c_N, \boldsymbol{M}_b^{\mathsf{hybrid}_4}\}.$$

Hybrid 5. This hybrid is the same as the previous hybrid except that $(\mathsf{seed}, g_a^{\xi_a}, g_*, h_a^{\mathsf{sk}_a \xi_a}, h_*, \mathsf{seed}_a)$ are provided by Sim, where g_* and h_* is randomly sampled. Assuming the randomness of (g_b, h_b) and the DDH assumption, two views are indistinguishable.

Hybrid 6. This hybrid is identical to the previous hybrid except that in the previous hybrid, Sim aborts unless for all $i, j \in [k]$,

$$[N+1, N+p] \subset \sigma_{b,i}^{-1}(\boldsymbol{E}_{a,i}) = \sigma_{a,j}^{-1}(\boldsymbol{E}_{b,j}).$$

While in this hybrid, Sim aborts unless for all $i, j \in [k]$,

$$\mathcal{P} \cup [N+1, N+p] = \sigma_{b,i}^{-1}(\boldsymbol{E}_{a,i}) = \sigma_{a,j}^{-1}(\boldsymbol{E}_{b,j}).$$

By Lemma 3, two hybrids are at least $\frac{1}{(p+|\mathcal{P}|)^k}$-indistinguishable.

Hybrid 7. This hybrid is the same as the previous hybrid except that the equality set is uniformly sampled. Sample $\mu \xleftarrow{\$} [N+p]^{|\mathcal{P}|+p}$, and define

$$\boldsymbol{M}_b^{\mathsf{hybrid}_7}[j] := \begin{cases} R_j^{\xi_b}(c_j[2]/c_j[1]^{\mathsf{sk}_a})^{\xi_b}, j \in \mu \text{ and } j \in [N], \\ \gamma_{j-N}^{\xi_b}, j \in \mu \text{ and } j > N, \\ \xleftarrow{\$} \mathbb{G}, \text{otherwise}; \end{cases}$$

$$\mathsf{H}(\boldsymbol{P}_b^{\mathsf{hybrid}_7}[j]^{\xi_a^{-1}}) = \boldsymbol{P}_b'^{\mathsf{hybrid}_3}[j], \text{if } j \in \mu.$$

Also in this hybrid, Sim aborts unless for all $i, j \in [k]$,

$$\mu = \boldsymbol{E}_{a,i} = \sigma_{a,j}^{-1}(\boldsymbol{E}_{b,j}).$$

And we have

$$\mathsf{View}_{\mathsf{hybrid}_6} := \{\sigma_b(\mathcal{P} \cup [N+1, N+p]), R_1, \ldots, R_N, c_1, \ldots,$$

$$c_N, \gamma_1, \ldots, \gamma_p, \boldsymbol{M}_b^{\mathsf{hybrid}_4}, \boldsymbol{P}_b^{\mathsf{hybrid}_2}, \boldsymbol{P}_b'^{\mathsf{hybrid}_3}, \xi_a\},$$

and

$$\mathsf{View}_{\mathsf{hybrid}_7} := \{\mu, R_1, \ldots, R_N, c_1, \ldots, c_N, \gamma_1, \ldots, \gamma_p,$$
$$\boldsymbol{M}_b^{\mathsf{hybrid}_7}, \boldsymbol{P}_b^{\mathsf{hybrid}_7}, \boldsymbol{P}_b'^{\mathsf{hybrid}_3}, \xi_a\}.$$

Because of the randomness of $(\sigma_b, R_1, \ldots, R_N, \gamma_1, \ldots, \gamma_p)$ and the DDH assumption, two views are indistinguishable.

Since this hybrid is identical to the ideal world, we show that the real and ideal worlds are indistinguishable.

Case II. When one of the server is corrupted with both the recipient and the sender(s). Since the servers do not have any private input, the protocol does not provide a malicious receiver any information to extract. However, we still present the whole simulation for this case.

	Two-Face					One-Face
	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	N/A
#padding values p	2^{20}	10322	1024	256	102	N/A
Server running time (sec)	409.52	197.46	212.87	265.19	301.00	25.37
Reconstruction time (sec)	5.18	0.27	0.33	0.40	0.49	0.046
Server $\leftrightarrow$ Server (MB)	393	183.54	229.45	278.14	327.06	41
Digest size (KB)	16384	243.12	33.57	11.95	7.13	0.21

Table 1. Cost under different choice of k, using board size $N = 2^{19}$, 4 threads. The parameter k denotes how many times the servers repeat the permutation and blinding of input vectors, while p represents the number of dummy values the recipient pads to the input. The two parameters should satisfy the condition that $p^k \geq 2^{40}$. Reconstruction time refers to the time the receiver spends recovering the relevant signals after receiving the response from the servers. Digest size represents the amount of data communicated from the server to the receiver. We also include our semi-honest protocol (One-Face) in the rightmost column of the table for reference.

1. Upon receiving $(\mathsf{seed}, g_{\mathcal{A}}, g_b^{\xi_b}, h_{\mathcal{A}}, h_b^{\mathsf{sk}_b\xi_b}, \mathsf{sk}_b, \mathsf{seed}_b)$, compare $\mathsf{H}(\mathsf{seed}, g_{\mathcal{A}}, g_b^{\xi_b}, h_{\mathcal{A}}, h_b^{\mathsf{sk}_b\xi_b})$ with $\mathcal{A}$ and abort if not consistent. Compute $(g_a, g_b, h_a, h_b, R_1, \ldots, R_N, \gamma_1, \ldots, \gamma_p) \leftarrow \mathsf{PRG}(\mathsf{seed})$, and $(\xi_b, \{\sigma_{b,i}\}_{i\in[k]}) \leftarrow \mathsf{PRG}(\mathsf{seed}_b)$;
2. Upon receiving $\boldsymbol{M}_a$ from $\mathcal{A}$, Sim emulates $\mathcal{F}_{\mathsf{BatchPOE}}$ and receives x and y from $\mathcal{A}$. Sim aborts if $g_a^x \neq g_{\mathcal{A}}$ or $h_a^y \neq h_{\mathcal{A}}$ or $R_i^x c_i[1]^y \neq \boldsymbol{M}_a[i]$ for any $i \in [N]$ or $\gamma_i^x \neq \boldsymbol{M}_a[i+N]$ for any $i \in [p]$;
3. For $i \in [k]$, Sim samples $\omega_{b,i} \xleftarrow{\$} \mathbb{Z}_p$, and sends $\boldsymbol{P}_{b,i} := \sigma_{b,i}(\boldsymbol{M}_a^{\omega_{b,i}})$ and $\boldsymbol{P}'_{b,i} := \sigma'_{b,i}(\mathsf{H}(R_1^{\omega_{b,i}}(c_1[2]/c_1[1]^{\mathsf{sk}_b})^{\omega_{b,i}}), \ldots, \mathsf{H}(R_N^{\omega_{b,i}}(c_N[2]/c_N[1]^{\mathsf{sk}_b})^{\omega_{b,i}}), \mathsf{H}(\gamma_1^{\omega_{b,i}}), \ldots, \mathsf{H}(\gamma_p^{\omega_{b,i}}))$ to $\mathcal{A}$;
4. Sim sends $\boldsymbol{M}_b := (R_1^{\xi_b} c_1[1]^{\mathsf{sk}_b\xi_b}, \ldots, R_N^{\xi_b} c_N[1]^{\mathsf{sk}_b\xi_b}, \gamma_1^{\xi_b}, \ldots, \gamma_p^{\xi_b})$ to $\mathcal{A}$;
5. Sim emulates $\mathcal{F}_{\mathsf{BatchPOE}}$ and returns 1 to $\mathcal{A}$ twice;
6. For $i \in [k]$, upon receiving $(\boldsymbol{P}_{a,i}, \boldsymbol{P}'_{a,i})$ from $\mathcal{A}$, aborts if there are repeated elements in $\boldsymbol{P}_{a,i}$. Otherwise, compute $\boldsymbol{E}_{b,i} := \{j \mid \mathsf{H}(\boldsymbol{P}_{a,i}[j]^{\xi_b^{-1}}) = \boldsymbol{P}'_{a,i}[j]\}$. Send $\{\boldsymbol{E}_{b,i}\}_{i\in[k]}$ to $\mathcal{A}$.

The only difference between the ideal world and the real protocol is that the $\mathcal{F}_{\mathsf{BatchPOE}}$ ideal functionality is internally simulated, and thus two worlds are indistinguishable.

6 Performance Evaluation

We highlight the main takeaways from our performance evaluation:

– One-Face is highly competitive compared to state-of-the-art semi-honest protocols. Its digest size is about 30× smaller than the best prior work (PS2) while at the same time maintaining a competitive running time compared to the fastest prior work (HomeRun).

– Two-Face introduces about 10× overhead of running time compared to its semi-honest counterpart. Its performance is still competitive compared to prior semi-honest protocols. For example, the digest size of Two-Face is only 5× bigger than the best digest size of all prior semi-honest protocols.
– Finally, we experimentally verified the scalability of our protocols. It is friendly to multi-threading, and the running time scales linearly with the board size, processing one million messages in 4 minutes under moderate hardware.

6.1 Experiment setup

Our protocol is implemented in C++. We compared the performance of different elliptic curves across various libraries and decided to instantiate our cyclic group $\mathbb{G}$ using the secp256r1 elliptic curve from OpenSSL [23], due to its exceptional performance in elliptic curve multiplication. We set the computational security parameter κ to 128, and the statistical security parameter λ to 40.

SophOMD [16] and our protocols are executed using two AWS c4.8xlarge instances, with 36v CPUs (2.9GHz) and 60GB RAM. The results for PS1, PS2 [20], and HomeRun [14] is obtained from [14]. We use the same network configuration as prior works (10Gbps bandwidth). However, we use significantly less amount of communication and our speed would not be impacted much even under 1Gbps network.

6.2 Microbenchmark

According to Lemma 3, we set p and k such that $p^k \geq 2^{40}$ for 40 bits of statistical security. Below, we analyze how the parameters k and p influence the performance of our protocol.

Figure 7 illustrates the impact of two parameters on server running time, as well as the concrete cost breakdown, assuming board size $N = 2^{19}$ (approximately the number of Bitcoin transactions per day, as noted in [20]). As illustrated in the bar chart, elliptic curve operations constitute the majority of the server-side computation. In our protocol, each server performs a total of $8N + 5p + k(4N + 3p)$ elliptic curve multiplications. Given $N = 2^{19}, p = \lceil 2^{40/k} \rceil$, the number of EC multiplication is minimized to 1.06×10^7 when $k = 3$, with the second-lowest value of 1.26×10^7 when $k = 4$. This analysis of k aligns with the experimental results in Figure 7, where the server running time reaches its minimum at $k = 3$ and $k = 4$.

The remaining part of the server running time includes: (1) Data transfer, which contains both the communication latency between the two servers and the conversion of raw data into valid elliptic curve points; (2) Digest generation, where each server produces a digest; and (3) Other overhead, such as the time needed to expand the random seed.

Table 1 presents number of padding values, server running time, reconstruction time (recipient running time), server to server communication and digest size (server to recipient communication) across different values of k. As shown in

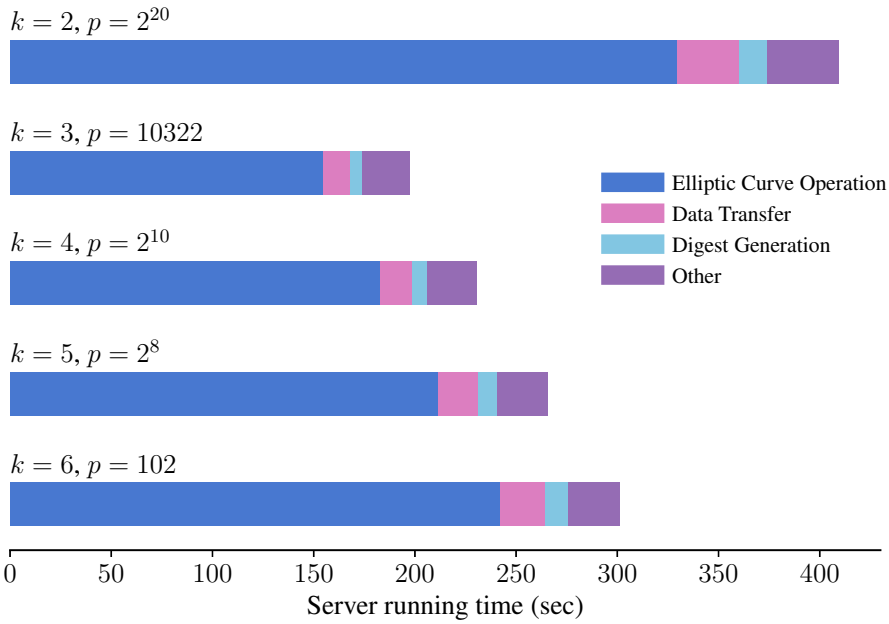

Fig. 7. Cost breakdown of server running time under different k and p, using board size $N = 2^{19}$, 4 threads. The parameter k denotes how many times the servers repeat the permutation and blinding of input vectors, while p represents the number of dummy values the recipient pads to the input.

the table, besides server running time, the reconstruction time and server communication are also minimized at $k = 3$, with the second-lowest value occurring at $k = 4$. The digest size decreases significantly as k increases, due to the exponential decline of p. Notably, in the configuration of $k = 4$, a digest size of 33.57 KB is already comparable to that of other semi-honest protocols. Given this, along with reasonable server running time and communication cost, we propose our protocol with $k = 4$.

6.3 Protocol Comparison

In this section, we provide a thorough performance comparison of our protocol against prior works, leveraging the experiment results in [14]. In the realm of FHE-based constructions [16, 17, 18], we replace OMDp1 [17] to SophOMD [16] as the latter is the most recent work and offers the best performance in this category. To ensure a fair comparison, we manually limit the number of threads in SophOMD via setting OMP_NUM_THREADS=1 (or 4, or 16) prior to execution. Assuming that there are 2^{19} messages on the bulletin board, we compare the computation and communication costs and summarize the results in Table 2.

		PS1 [20]	PS2 [20]	SophOMD [16]	HomeRun [14]	One-Face	Two-Face
Server(s) running time (sec)	1 thread	4.48	7425.94	1475.88	19.16	97.6	775.69
	4 threads			660.13	7.42	25.37	212.87
	16 threads			549.31	4.60	11.86	120.29
Reconstruction time (sec)		1.52	0.00014	0.0078	0.0065	0.046	0.33
Recipient → Server(s) (KB)		0.5	2.09	145408	0.22	0.048	0.35
Number of server		1	2	1	2	2	2
Server ↔ Server (MB)		N/A	∼3750	N/A	582.84	41	229.45
Digest size for 50 pertinent signals (KB)		25	6.25	132	128	0.21	33.57
Limit of $\|\mathcal{P}\|$		$0 \sim 50$			$0 \sim N$		
Assumption		TEE	$\mathcal{F}_{\mathsf{OT}}$	RLWE	DDH, $\mathcal{F}_{\mathsf{OT}}$	DDH, RO	
Malicious Security		✓	✗	✗	✗	✗	✓

Table 2. Comparisons of running time, communication cost, assumption used, and security level in LAN settings, assuming board size $N = 2^{19}$. The running time for PS1, PS2, and HomeRun are sourced from [14]. Reconstruction time refers to the time the receiver spends recovering the relevant signals after receiving the response from the servers. Digest size represents the amount of data communicated from the server to the receiver. $|\mathcal{P}|$ refers to the number of pertinent signals for a single recipient. $\mathcal{F}_{\mathsf{OT}}$ refers to the functionality of Oblivious Transfer [24]; RLWE refers to Ring Learning With Errors [19]; RO refers to Random Oracle [1].

Server(s) Running Time. The server(s) is responsible for helping the recipient to identify pertinent signals. As shown in Table 2, One-Face incurs 2.6× overhead compared to HomeRun, while achieving a 46.3× speedup over SophOMD. Two-Face, our maliciously secure protocol, is also competitive with semi-honest protocols, showing 26.2× overhead relative to HomeRun and running 4.6× faster than SophOMD. The comparison is obtained using 16 threads, a typical configuration in modern servers. Our performance is highly competitive because the protocol scales well with multithreading; when using fewer threads (e.g., 1 or 4), the overhead increases slightly. We include these results in the table for completeness.

Recipient Reconstruction Time. After receiving the response from the server(s), the recipient needs to reconstruct the pertinent indexes (or messages). From the table, we can tell that the reconstruction in all the semi-honest schemes are much less than 1 second. The reconstruction time of Two-Face is 4.6× faster than the other malicious protocol - PS1. The recipient in PS1 needs to decrypt 50 ciphertexts. While in Two-Face, the recipient only needs to reverse $2k$ permuted sets of size $p + |\mathcal{P}|$ and check whether they are all equal (The results is based on the configuration $|\mathcal{P}| = 50$, $p = 2^{10}$, $k = 4$).

Recipient → Server(s). The recipient needs to send a request to the server(s). In One-Face, the recipient should send one share of the secret key, and one seed to expand permutations, resulting in 49 bytes in total. In Two-Face, the recipient needs to send four seeds, one share of secret key and 8 group elements

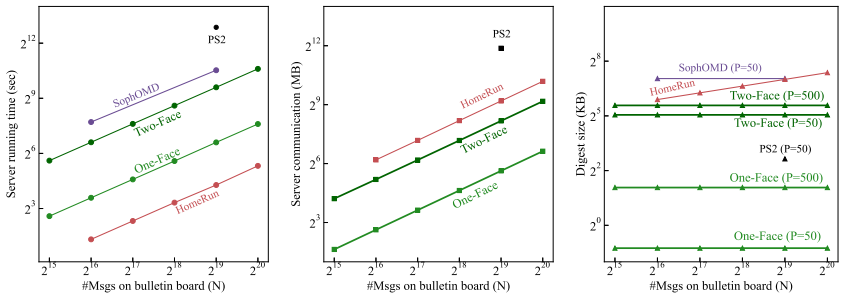

Fig. 8. Comparison of our protocols: One-Face (semi-honest) and Two-Face (malicious), with SophOMD, HomeRun, and PS2 in server running time, server-server communication and digest size under different bulletin board size N, using single thread, LAN setting.

for Π_{BatchPOE}, which is 328 bytes in total. Both request size is much smaller than that in SophOMD and comparable with that in the other designs.

Server $\leftrightarrow$ Server. Besides our protocols, only HomeRun and PS2 rely on two servers. In One-Face, servers exchange 82 bytes per message on the board, resulting in a total server communication cost that is 14.2× smaller than HomeRun and 91.5× smaller than PS2. In Two-Face, servers exchange 458 bytes per message, achieving a 2.5× reduction compared to HomeRun and 16.3× reduction compared to PS2.

Digest Size. The server(s) responds with a digest to the recipient, such that the recipient can recover the pertinent indexes or messages. In One-Face, one server sends $|\mathcal{P}| = 50$ permuted indices to the recipient, resulting in a digest size of only 0.21KB, which achieves a 29.8× reduction in digest size compared to the best prior result. In Two-Face, each server should send $k(p + |\mathcal{P}|) =$ indices to the recipient. When $k = 4, p = 1024$, the digest size is 33.57KB, which is smaller than that in the designs of SophOMD and HomeRun.

Limit of $|\mathcal{P}|$. Like HomeRun, we also enjoy the feature that the number of pertinent signals for a recipient is unlimited. In contrast, other schemes only allow a recipient to receive up to a pre-determined number of pertinent signals. Additionally, as stated in Lemma 3, more pertinent signals will lead to a lower false negative probability in our scheme. For the sake of fair comparisons, we set the size of pertinent indices as 50 for all schemes.

Assumption & Malicious Security. As observed, Two-Face is the only one that achieves malicious security without relying on the setup assumption of TEE. Both the PS2 and HomeRun schemes cannot be straightforwardly adapted to malicious security, as both protocols require each server to maintain a private table, which serves as input for some general 2PC. To achieve malicious security, besides replacing the underlying semi-honest 2PC with a maliciously secure one, the servers must also prove that their inputs are honestly derived from the

board and the recipient's request. SophOMD provides similar functionality to our protocol when the retrieval process is executed twice by different servers, assuming at least one server is honest. However, under these conditions, our protocol achieves lower server running time and a detection key size that is 415451× smaller.

6.4 Protocol Scalability

Figure 8 presents the experimental results as the number of messages N increases from 2^{15} to 2^{20}. We compare our malicious protocol with previous works do not rely on the TEE setup assumption. As shown in the figure, both One-Face and Two-Face achieve faster server running time than PS2 for $N = 2^{19}$, and SophOMD for $N = 2^{16}$ and $N = 2^{19}$. We only report two results from SophOMD because their implementation provides fine-tuned parameters solely for these two configurations. In comparison with HomeRun, our protocols introduce a notable overhead due to the extensive use of elliptic curve multiplications in order to achieve malicious security. It is worth noting that our implementation does not incorporate any optimization techniques for accelerating elliptic curve multiplication, despite repeatedly performing N multiplications with the same exponent. We leave the exploration of such optimizations to future work.

For server communication, we compare our protocol with two other multi-server schemes: HomeRun and PS2. The communication cost reported for HomeRun includes both its online and offline phases, since its offline phase cannot be reused across different bulletin board configurations. As illustrated in the figure, the server communication in both One-Face and Two-Face are consistently lower than HomeRun and PS2 with N varying from 2^{15} to 2^{20}.

In terms of digest size, only HomeRun scales with varying board sizes, while both SophOMD and PS2 have a fixed maximum number of 50 pertinent messages. When the board size exceeds 2^{15}, the digest size of HomeRun and SophOMD are consistently larger than that of Two-Face (with 500 pertinent signals). The digest size of One-Face is remarkably small — its digest for 500 pertinent signals is even smaller than the digest size for just 50 signals in PS2, which previously held the most compact digest among existing schemes.

References

1. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.
2. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018.
3. Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

4. Dan Boneh. The decision Diffie-Hellman problem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *LNCS*. Springer, 1998. Invited paper.
5. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.
6. Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: SGX cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC, August 2017. USENIX Association.
7. Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE Computer Society Press, May 2015.
8. Victor Costan and Srinivas Devadas. Intel SGX explained. Cryptology ePrint Archive, Report 2016/086, 2016.
9. Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Fast and private computation of cardinality of set intersection and union. In Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis, editors, *CANS 12*, volume 7712 of *LNCS*, pages 218–231. Springer, Berlin, Heidelberg, December 2012.
10. Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache attacks on intel SGX. In Cristiano Giuffrida and Angelos Stavrou, editors, *Proceedings of the 10th European Workshop on Systems Security, EUROSEC 2017*, pages 2:1–2:6. ACM, 2017.
11. Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 3889–3905. USENIX Association, August 2023.
12. Yan Huang, Jonathan Katz, and David Evans. Quid-Pro-Quo-tocols: Strengthening semi-honest protocols with dual execution. In *2012 IEEE Symposium on Security and Privacy*, pages 272–284. IEEE Computer Society Press, May 2012.
13. Sashidhar Jakkamsetti, Zeyu Liu, and Varun Madathil. Scalable private signaling. Cryptology ePrint Archive, Report 2023/572, 2023.
14. Yanxue Jia, Varun Madathil, and Aniket Kate. HomeRun: High-efficiency oblivious message retrieval, unrestricted. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 2012–2026. ACM Press, October 2024.
15. Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 705–734. Springer, Berlin, Heidelberg, May 2016.
16. Keewoo Lee and Yongdong Yeo. SophOMR: Improved oblivious message retrieval from SIMD-aware homomorphic compression. Cryptology ePrint Archive, Report 2024/1814, 2024.
17. Zeyu Liu and Eran Tromer. Oblivious message retrieval. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 753–783. Springer, Cham, August 2022.
18. Zeyu Liu, Eran Tromer, and Yunhao Wang. PerfOMR: Oblivious message retrieval with reduced communication and computation. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.

19. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Berlin, Heidelberg, May / June 2010.
20. Varun Madathil, Alessandra Scafuro, István András Seres, Omer Shlomovits, and Denis Varlakov. Private signaling. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 3309–3326. USENIX Association, August 2022.
21. Payman Mohassel and Matthew Franklin. Efficiency tradeoffs for malicious two-party computation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 458–473. Springer, Berlin, Heidelberg, April 2006.
22. Shen Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015.
23. OpenSSL Project. OpenSSL: The Open Source Toolkit for SSL/TLS, 2024. Available at `https://www.openssl.org/`.
24. Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005.
25. Lior Rotem. Simple and efficient batch verification techniques for verifiable delay functions. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 382–414. Springer, Cham, November 2021.
26. Sajin Sasy and Ian Goldberg. Sok: Metadata-protecting communication systems. *Proceedings on Privacy Enhancing Technologies*, 2024.
27. David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 179–182, 2012.

A Volume-Hiding Private Signaling

A.1 Linear-Digest in Exchange of Volume-Hiding

As discussed in Section 3.2, imposing a strict 50-message cap per client is impractical for two reasons. First, it would make the protocol highly vulnerable to denial-of-service attacks, since an adversary could easily exhaust a client's quota and block legitimate traffic. Second, no practical mechanism is known to enforce a per-client limit of 50 messages.

Accordingly, the only practical way to conceal message volume is to let the digest grow linearly with the size of the public ledger while keeping client computation efficient. In our protocol, rather than having servers perform the comparisons (which would leak volume), the two servers send three elements per entry to the client, who then carries out the comparisons locally. Full protocol is presented in Figure 9.

A.2 Concrete performance

In our Volume-Hiding protocol, since the adversary does not learn the volume of the retrieved signals, the false negative probability becomes $(\frac{1}{p+N})^k$ (according to Lemma 3) where p is the number of padding values, N is the size of the

Protocol $\Pi_{\mathsf{privSignal}}$

Participants: Two servers S_a and S_b, the recipient R, and multiple anonymous senders.

Setup & Send: Same as the semi-honest version (Figure 3).

Retrieve (Malicious, Volume-Hiding): Given $(c_1, c_2, \ldots, c_N)$ as the clue vector containing N distinct ElGamal ciphertexts, H as a random oracle, to perform a signal retrieval:

1. R samples $\mathsf{seed}, \mathsf{seed}_a, \mathsf{seed}_b \xleftarrow{\$} \{0,1\}^\lambda$, and calculates $(g_a, g_b, h_a, h_b) \leftarrow \mathsf{H}(\mathsf{seed})$, $(\mathsf{sk}_a, \xi_a) \leftarrow \mathsf{H}(\mathsf{seed}_a)$, $\xi_b \leftarrow \mathsf{H}(\mathsf{seed}_b)$, and $\mathsf{sk}_b = \mathsf{sk}_R - \mathsf{sk}_a$;
2. R sends $(\mathsf{seed}, g_a^{\xi_a}, g_b^{\xi_b}, h_a^{\mathsf{sk}_a\xi_a}, h_b^{\mathsf{sk}_b\xi_b})$ to two servers, (seed_a) to S_a, and $(\mathsf{sk}_b, \mathsf{seed}_b)$ to S_b. Two servers compare $\mathsf{H}(\mathsf{seed}, g_a^{\xi_a}, g_b^{\xi_b}, h_a^{\mathsf{sk}_a\xi_a}, h_b^{\mathsf{sk}_b\xi_b})$ and abort if not consistent;
3. Both servers compute $(g_a, g_b, h_a, h_b, R_1, \ldots, R_N) \leftarrow \mathsf{H}(\mathsf{seed})$. S_a computes $(\mathsf{sk}_a, \xi_a, \{\sigma_{a,i}\}_{i\in[k]}) \leftarrow \mathsf{H}(\mathsf{seed}_a)$, and S_b computes $(\xi_b, \{\sigma_{b,i}\}_{i\in[k]}) \leftarrow \mathsf{H}(\mathsf{seed}_b)$;
4. Two servers proceed as follows to calculate all the indices $i \in [N]$ satisfying $c_i[1]^{\mathsf{sk}_a} = c_i[2]/c_i[1]^{\mathsf{sk}_b}$:
   - S_a calculates and sends $\boldsymbol{M}_a := (R_1^{\xi_a} c_1[1]^{\mathsf{sk}_a\xi_a}, \ldots, R_N^{\xi_a} c_N[1]^{\mathsf{sk}_a\xi_a})$ to S_b;
   - Given public value $(R_1, \ldots, R_N, g_a, c_1, \ldots, c_N, h_a, \boldsymbol{M}_a, g_a^{\xi_a} h_a^{\mathsf{sk}_a\xi_a})$, S_a invokes $\mathcal{F}_{\mathsf{BatchPOE}}$ with private input ξ_a and $\mathsf{sk}_a\xi_a$ to prove that for $i \in [N]$, $\boldsymbol{M}_a[i] = R_i^{\xi_a} c_i[1]^{\mathsf{sk}_a\xi_a}$. S_b aborts if check fails;
   - For $i \in [k]$:
     - S_b samples $\omega_{b,i} \xleftarrow{\$} \mathbb{Z}_p$, and sends $\boldsymbol{P}_{b,i} := \sigma_{b,i}(\boldsymbol{M}_a^{\omega_{b,i}})$ to S_a;
     - S_a aborts if there are repeated elements in $\boldsymbol{P}_{b,i}$. Otherwise, compute $\boldsymbol{A}_i := (\mathsf{H}(\boldsymbol{P}_{b,i}[1]^{\xi_a^{-1}}), \ldots, \mathsf{H}(\boldsymbol{P}_{b,i}[N]^{\xi_a^{-1}}))$;
     - S_a sends $\boldsymbol{A}_i$ to R, and S_b sends $\boldsymbol{B}_i := (\mathsf{H}(R_1^{\omega_{b,i}}(c_1[2]/c_1[1]^{\mathsf{sk}_b})^{\omega_{b,i}}), \ldots, \mathsf{H}(R_N^{\omega_{b,i}}(c_N[2]/c_N[1]^{\mathsf{sk}_b})^{\omega_{b,i}}))$ to R;
     - R computes $\boldsymbol{E}_{a,i} := \{j \mid \boldsymbol{A}_i[\sigma_{b,i}(j)] = \boldsymbol{B}_i[j]\}$;
5. Two servers switch the roles and repeat the above procedure so that R obtains $\{\boldsymbol{E}_{b,i}\}_{i\in[k]}$;
6. R aborts unless for all $i, j \in [k]$ s.t. $\boldsymbol{E}_{a,i} = \boldsymbol{E}_{b,j}$.

Fig. 9. Our volume-hiding malicious protocol for private signaling.

board, and k is number of times applying permutations. To reach the security requirement for $\lambda = 40$ while minimizing the digest size, we set $p = 0$ and $k = 3$, and the digest size becomes $2kN * \{\text{output size of the random oracle}\} = 2 * 3 * 2^{19} * 16\text{bytes} = 48\text{MB}$.

However, despite the linear blowup of the digest size, the client can still process the digest and recover the relevant indices in under half a second. Comparing the equality of 3 millions of group elements takes less than 10ms, resulting in a reconstruction time of 0.34 second.

B Proof of Theorem 1

Theorem 3. *(Theory 1, restated)* $\Pi_{\mathsf{privSignal}}.\mathsf{Retrieve}$ *in Figure 3 securely realizes* $\mathcal{F}_{\mathsf{Retrieve}}$ *in the random oracle model in the semi-honest setting, assuming the hardness of DDH.*

Proof. We consider a worst-case scenario in which the adversary colludes with the sender and has complete knowledge of the randomness used in all ciphertexts posted on the bulletin board. We show there exists a PPT simulator Sim interacting with functionality $\mathcal{F}_{\mathsf{Retrieve}}$ generates a transcript that is indistinguishable from the transcript generated by the real-world adversary $\mathcal{A}$ in each of the following two settings.

Case I. When S_a is corrupted with the sender(s).
Given $(c_1, c_2, \ldots, c_N)$ as the clue vector:

1. Upon receiving $(\mathsf{Retrieve}, |\mathcal{P}|)$ from $\mathcal{F}_{\mathsf{retrieve}}$, Sim samples $\mathsf{sk}_a \xleftarrow{\$} \mathbb{Z}_p$ and sends to $\mathcal{A}$;
2. Upon receiving $\boldsymbol{M}$ from $\mathcal{A}$, sample $\mu \in [N]^{|\mathcal{P}|}$, $\sigma \xleftarrow{\$} \mathbb{Z}_p^N$, and $\omega \xleftarrow{\$} \mathbb{Z}_p$. Sim computes $\boldsymbol{P} := \sigma(\boldsymbol{M}^\omega)$, $\boldsymbol{P'} \xleftarrow{\$} \mathbb{G}^N$, and programs $\mathsf{H}'(\mathsf{H}(c[i]^{\mathsf{sk}_a})^\omega) = \boldsymbol{P'}[i]$ for $i \in \mu$;
3. Sim sends $\boldsymbol{P}$ and $\boldsymbol{P'}$ to $\mathcal{A}$ and receives $\boldsymbol{E}$.

By construction, the view only differs in the way $\boldsymbol{P'}$ is constructed. Denote $a_i = c_i[1]^{\mathsf{sk}_a}$ and $b_i = c_i[2]/c_i[1]^{\mathsf{sk}_b}$ where sk_a and sk_b are the secret keys in the real world, then we demonstrate the following two views are indistinguishable:

$$\begin{aligned}\mathsf{View}_{\mathsf{real}} := \{&\mathsf{H}(b_1), \ldots, \mathsf{H}(b_N), \boldsymbol{P'} := \sigma(\mathsf{H}'(\mathsf{H}(b_1)^\omega),\\ &\ldots, \mathsf{H}'(\mathsf{H}(b_N)^\omega)), \boldsymbol{M}, \boldsymbol{P}\},\\ \mathsf{View}_{\mathsf{ideal}} := \{&\mathsf{H}(b_1), \ldots, \mathsf{H}(b_N), \boldsymbol{P'} := \sigma(\mathsf{H}'(\mathsf{H}(a_1)^\omega),\\ &\ldots, \mathsf{H}'(\mathsf{H}(a_{|\mathcal{P}|})^\omega), \mathsf{H}'(r_1), \ldots, \mathsf{H}'(r_{N-|\mathcal{P}|})), \boldsymbol{M}, \boldsymbol{P}\},\end{aligned}$$

where $r_1, \ldots, r_{N-|\mathcal{P}|} \xleftarrow{\$} \mathbb{G}$.

Constructs $(\mathsf{H}(b_1), \ldots, \mathsf{H}(b_N)) := (g, g^x, g^{r_3}, \ldots, g^{r_N})$, and

$$\begin{aligned}\mathcal{D} := (&g, g^x, g^{r_3}, \ldots, g^{r_N}, \sigma(\mathsf{H}'(g^y), \mathsf{H}'(g^z), \mathsf{H}'(g^{r_3 y}), \ldots,\\ &\mathsf{H}'(g^{r_N y})), \boldsymbol{M}, \boldsymbol{P}).\end{aligned}$$

Note that $\mathcal{D}$ belongs to $\mathsf{View}_{\mathsf{real}}$ if and only if (g, g^x, g^y, g^z) is a proper Diffie-Hellman tuple, i.e., $z = xy$ and to $\mathsf{View}_{\mathsf{ideal}}$ otherwise. Therefore, the adversary has only negligible advantage in distinguishing the two views.

Case II. When S_b is corrupted with the sender(s). In this case, given $(c_1, c_2, \ldots, c_N)$ as the clue vector, Sim randomly samples $\boldsymbol{M} \xleftarrow{\$} \mathbb{G}^N$ and sends to $\mathcal{A}$. As we explicitly require that c_i to be pairwise distinct, $\boldsymbol{M}$ is intinguishable from a vector of random group elements assuming the hardness of DDH.

C Proof of Lemma 2

Lemma 4. *(Lemma 2, restated) The protocol Π_{BatchPOE} securely realizes $\mathcal{F}_{\mathsf{BatchPOE}}$ with soundness error $|\mathbb{G}|^{-1}$ in the random oracle model, assuming the hardness of the discrete logarithm problem.*

Proof. We first consider the case of a corrupted prover and then the case of a corrupted verifier.

Corrupted prover $\mathcal{P}^*$ (Knowledge Soundness). We argue that there exists an extractor $\mathcal{E}$ such that for any probabilistic polynomial-time (PPT) prover $\mathcal{P}^*$, if $\mathcal{P}^*$ convinces $\mathcal{V}$ on input $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$, then $\mathcal{E}^{\mathcal{P}^*}$ extracts a witness (x, y) such that $\boldsymbol{C}[i] = \boldsymbol{A}[i]^x \boldsymbol{B}[i]^y$ for $i \in [N]$ with probability $1 - |\mathbb{G}|^{-1}$.

The extractor $\mathcal{E}$ programs H and runs $\mathcal{P}^*$ twice with the same t but different challenges e and e'. With P^* returning (z_1, z_2) and (z_1', z_2'), $\mathcal{E}$ computes

$$x = \frac{z_1 - z_1'}{e - e'}, y = \frac{z_2 - z_2'}{e - e'}.$$

Denote $\boldsymbol{A}[N+1] = \tilde{A}, \boldsymbol{B}[N+1] = \tilde{B}, \boldsymbol{C}[N+1] = \tilde{C}$. Then we argue that if $\exists i \in [N+1], \boldsymbol{A}[i]^x \boldsymbol{B}[i]^y \neq \boldsymbol{C}[i]$, the probability that $C = A^x B^y$ is small. Let $\mathcal{S}$ be the set for $i \in \mathcal{S}, \boldsymbol{A}[i]^x \boldsymbol{B}[i]^y = \boldsymbol{C}[i] \cdot D_i$ where $D_i \neq 1$. We have

$$\begin{aligned}
&\Pr_{\boldsymbol{q} \xleftarrow{\$} \mathbb{Z}_p^{N+1}} \Big[\Big(\prod_{i \in [N+1]} \boldsymbol{A}[i]^{\boldsymbol{q}[i]} \Big)^x \Big(\prod_{i \in [N+1]} \boldsymbol{B}[i]^{\boldsymbol{q}[i]} \Big)^y \\
&\qquad\qquad = \prod_{i \in [N+1]} \boldsymbol{C}[i]^{\boldsymbol{q}[i]} \Big] \\
= &\Pr_{\boldsymbol{q} \xleftarrow{\$} \mathbb{Z}_p^{N+1}} \Big[\Big(\prod_{i \in \mathcal{S}} \boldsymbol{A}[i]^{\boldsymbol{q}[i]} \Big)^x \Big(\prod_{i \in \mathcal{S}} \boldsymbol{B}[i]^{\boldsymbol{q}[i]} \Big)^y = \prod_{i \in \mathcal{S}} \boldsymbol{C}[i]^{\boldsymbol{q}[i]} \Big] \\
= &\Pr_{\boldsymbol{q} \xleftarrow{\$} \mathbb{Z}_p^{N+1}} \Big[\prod_{i \in \mathcal{S}} (\boldsymbol{C}[i] \cdot D_i)^{\boldsymbol{q}[i]} = \prod_{i \in \mathcal{S}} \boldsymbol{C}[j]^{\boldsymbol{q}[i]} \Big] \\
= &\Pr_{\boldsymbol{q} \xleftarrow{\$} \mathbb{Z}_p^{N+1}} \Big[\prod_{i \in \mathcal{S}} D_i^{\boldsymbol{q}[i]} = 1 \Big] \\
= &\Pr_{\boldsymbol{q} \xleftarrow{\$} \mathbb{Z}_p^{N+1}} \Big[\sum_{i \in \mathcal{S}} \Big(\boldsymbol{q}[i] \cdot \mathsf{Dlog}(D_i) \Big) = 0 \Big].
\end{aligned}$$

If $N + 1 \notin \mathcal{S}$, then x and y are publicly defined given the randomness of $\tilde{A}$ and $\tilde{B}$, and thus $\{D_i\}_{i \in \mathcal{S}}$ is fixed before the chosen of $\boldsymbol{q}$, and we have

$$\begin{aligned}
&\Pr_{\boldsymbol{q} \xleftarrow{\$} \mathbb{Z}_p^{N+1}} \Big[\sum_{i \in \mathcal{S}} \Big(\boldsymbol{q}[i] \cdot \mathsf{Dlog}(D_i) \Big) = 0 \Big] \\
= &\Pr_{\boldsymbol{q} \xleftarrow{\$} \mathbb{Z}_p^{N+1}} \Big[\boldsymbol{q}[i^*] = -\mathsf{Dlog}^{-1}(D_{i^*}) \cdot \sum_{i \in \mathcal{S} \setminus i^*} \Big(\boldsymbol{q}[i] \mathsf{Dlog}(D_i) \Big) \Big] \\
= &|\mathbb{G}|^{-1}.
\end{aligned}$$

If $N+1 \in \mathcal{S}$, then we argue that given the knowledge of every discrete log in the protocol except $\tilde{A}$, $\mathcal{P}^*$ can calculate $\mathsf{Dlog}(\tilde{A})$ as follows, which violates the discrete log assumption. Since

$$\mathsf{Dlog}(\tilde{A}) = x^{-1}\Big(\mathsf{Dlog}(D_{N+1}) + \mathsf{Dlog}(\tilde{C}) - y \cdot \mathsf{Dlog}(\tilde{B})\Big),$$

where

$$\begin{aligned}\mathsf{Dlog}(D_{N+1}) &= -\sum_{i\in\mathcal{S}}\Big(\boldsymbol{q}[i]\cdot\mathsf{Dlog}(D_i)\Big)\\ &= -\sum_{i\in\mathcal{S}}\Big(\boldsymbol{q}[i]\Big(x\mathsf{Dlog}(\boldsymbol{A}[i]) + y\mathsf{Dlog}(\boldsymbol{B}[i])\\ &\qquad\qquad - \mathsf{Dlog}(\boldsymbol{C}[i])\Big)\Big).\end{aligned}$$

Corrupted Verifier $\mathcal{V}^*$ (Zero-Knowledge). We argue that there exists a PPT simulator Sim such that for any PPT verifier $\mathcal{V}^*$, the transcript generated by Sim on input $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ is computationally indistinguishable from a real execution of the protocol between an honest prover and $\mathcal{V}^*$.

The simulator Sim simulates the transcript $(\tilde{C}, t, z_1, z_2)$ as follows:

1. Sim samples $\boldsymbol{q} \xleftarrow{\$} \mathbb{Z}_p^{N+1}$ and computes

$$A := \tilde{A}^{\boldsymbol{q}[N+1]} \prod_{i\in[N]} \boldsymbol{A}[i]^{\boldsymbol{q}[i]}, B := \tilde{B}^{\boldsymbol{q}[N+1]} \prod_{i\in[N]} \boldsymbol{B}[i]^{\boldsymbol{q}[i]};$$

2. Sim samples $t, z_1, z_2, e \xleftarrow{\$} \mathbb{Z}_p$ and computes

$$\tilde{C} = \left(\frac{(A^{z_1}B^{z_2}\cdot t^{-1})^{e^{-1}}}{\prod_{i\in[N]}\boldsymbol{C}[i]^{\boldsymbol{q}[i]}}\right)^{\boldsymbol{q}[N+1]^{-1}},$$

   which ensures that $A^{z_1}B^{z_2} = t \cdot C^e$;
3. Sim programs $\mathsf{H}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \tilde{A}, \tilde{B}, \tilde{C}) = \boldsymbol{q}$ and $\mathsf{H}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \tilde{A}, \tilde{B}, \tilde{C}, \boldsymbol{q}, t) = e$.

D Proof of Lemma 3

Lemma 5. *(Lemma 3, restated) Given $(c_1, \ldots, c_N)$ as the clue vector obtained from $\mathcal{F}_{\mathsf{Ledger}}$, an honest recipient R participating $\Pi_{\mathsf{privSignal}}.\mathsf{Retrieve}$ (Figure 6) either learns the set of pertinent signals $\mathcal{P} = \{i \mid \mathsf{Dec}_{\mathsf{sk}_R}(c_i) = 1\}$ or the protocol aborts with probability greater than $1 - \frac{1}{(p+|\mathcal{P}|)^k}$ in the random oracle model, assuming the hardness of DDH and $p > N^{\frac{N}{N-1}} - N$.*

Proof. Assuming the existence of $\mathcal{F}_{\mathsf{BatchPOE}}$, the vector $\boldsymbol{M}_a$ is well defined by public values, and the adversary thus can not cheat in calculating $\boldsymbol{M}_a$. As a result, if the adversary honestly sends $\boldsymbol{E}_{a,1}$ to the recipient, then the honest

recipient R either gets the set of pertinent signals $\mathcal{P} = \sigma_{b,1}^{-1}(\boldsymbol{E}_{a,1})$, or aborts if any pertinent set received later is not consistent with $\mathcal{P}$.

Below, we analyze the probability that the adversary can drop certain indices from the sets $\boldsymbol{E}_{a,1}$ while the protocol does not abort.

As stated in the proof Theorem 2, the real-world protocol is indistinguishable from the one in Hybrid 5 of Case I. Equivalently, we present the probability that the adversary drops certain indices from $\boldsymbol{E}_{a,i}$ and Sim does not abort in this hybrid. Denote the set that the adversary drops in $\boldsymbol{E}_{a,i}$ as ϕ_i, and Sim will not abort if and only if:

1. $\sigma_{b,i}^{-1}(\phi_i) \not\subset [N+1, N+p]$: without the knowledge of $\sigma_{b,i}$, ϕ_i should not touch the values in $[N+1, N+p]$;
2. $\sigma_{b,i}^{-1}(\phi_i) = \sigma_{b,j}^{-1}(\phi_j)$ for $i, j \in [k]$: each ϕ_i should be consistent after reversing the permutation;
3. The adversary should also correctly guess $\varphi = \sigma_{b,i}^{-1}(\phi_i)$ and modify these entries when constructing $\boldsymbol{P}_{a,i}$ and $\boldsymbol{P}'_{a,i}$, so that entries in φ will not appear in $\boldsymbol{E}_{b,i}$.

We assume a worst-case scenario in which the adversary already knows the set $\mathcal{P}$ (e.g., by excluding invalid clues they themselves have posted), which enables the adversary directly guess $\varphi \subset \mathcal{P}$. As observed in the Hybrid 5 of Case I, the adversary calculates $\boldsymbol{E}_{a,i}$ by computing the intersection of two randomly sampled vectors. Therefore there is no input to help $\mathcal{A}$ guess $(\varphi, \{\phi_i\}_{i\in[k]})$ other than $(\mathcal{P}, \{\boldsymbol{E}_{a,i}\}_{i\in[k]})$. Alternatively, the problem can be viewed as a game of picking balls:

1. Randomly pick $|\varphi|$ balls from $|\mathcal{P}|$ black balls and p white balls. Fail if any chosen ball is white;
2. Repeat $k-1$ times: Mix the $|\varphi|$ black balls with another $p + |\mathcal{P}| - |\varphi|$ white balls and then blindly pick $|\varphi|$ balls again. Fail if any picked ball is white;
3. Mix the $|\varphi|$ black balls with another $|\mathcal{P}| - |\varphi|$ white balls and then blindly pick $|\varphi|$ balls again. Fail if any picked ball is white.

Thus we have

$$
\begin{aligned}
&\Pr\left[\begin{array}{c}\varphi, \{\phi_i\}_{i\in[k]} \leftarrow \mathcal{A}(\mathcal{P}, \{\boldsymbol{E}_{a,i}\}_{i\in[k]}) \\ \text{st. } \varphi = \sigma_{b,j}^{-1}(\phi_j) = \sigma_{b,i}^{-1}(\phi_i) \subset \mathcal{P}\end{array}\right] \\
=&\frac{\binom{|\mathcal{P}|}{|\varphi|}}{\binom{p+|\mathcal{P}|}{|\varphi|}} \cdot \frac{1}{\binom{p+|\mathcal{P}|}{|\varphi|}^{k-1}} \cdot \frac{1}{\binom{|\mathcal{P}|}{|\varphi|}} \\
\le&\Big(\frac{|\mathcal{P}|}{p+|\mathcal{P}|}\Big)^{|\varphi|} \cdot \Big(\frac{|\varphi|}{p+|\mathcal{P}|}\Big)^{(k-1)|\varphi|} \cdot \Big(\frac{|\varphi|}{|\mathcal{P}|}\Big)^{|\varphi|} \\
=&\Big(\frac{|\varphi|}{p+|\mathcal{P}|}\Big)^{k|\varphi|} \\
\le&\max\left(\frac{1}{(p+|\mathcal{P}|)^k}, \Big(\frac{|\mathcal{P}|}{p+|\mathcal{P}|}\Big)^{k|\mathcal{P}|}\right).
\end{aligned}
$$

Because

$$\begin{aligned}
&\ln \frac{1}{p+|\mathcal{P}|} - \ln\left(\left(\frac{|\mathcal{P}|}{p+|\mathcal{P}|}\right)^{|\mathcal{P}|}\right) \\
=&\ln \frac{1}{p+|\mathcal{P}|} - |\mathcal{P}| \cdot \ln \frac{|\mathcal{P}|}{p+|\mathcal{P}|} \\
=&(|\mathcal{P}|-1)\ln(p+|\mathcal{P}|) - |\mathcal{P}|\ln(|\mathcal{P}|)
\end{aligned}$$

is larger than 0 when

$$p > |\mathcal{P}|^{\frac{|\mathcal{P}|}{|\mathcal{P}|-1}} - |\mathcal{P}|.$$

Given $p > N^{\frac{N}{N-1}} - N > |\mathcal{P}|^{\frac{|\mathcal{P}|}{|\mathcal{P}|-1}} - |\mathcal{P}|$, the above equation is greater than 0, and thus the probability is less than $\frac{1}{(p+|\mathcal{P}|)^k}$.