Private Signaling

Varun Madathil [*1], Alessandra Scafuro[1], István András Seres[2], Omer Shlomovits[3], and Denis Varlakov[3]

[1]North Carolina State University
[2]Eötvös Loránd University
[3]ZenGo X

June 22, 2021

Abstract

We introduce the problem of *private signaling*. In this problem, a sender posts a message to a certain location of a public bulletin board, and then computes a signal that allows only the intended recipient (and no one else) to learn that it is the recipient of the posted message. Besides privacy, this problem has the following crucial *efficiency* requirements. First, the sender and recipient do not participate in any out-of-band communication, and second, the overhead of the recipient should be asymptotically better than scanning the entire board.

Existing techniques, such as the server-aided fuzzy message detection (Beck et al., CCS'21), could be employed to solve the private signaling problem. However, this solution requires that the computational effort of the recipient grows with the amount of privacy desired, providing no saving over scanning the entire board if the maximum privacy is required.

In this work, we present a server-aided solution to the private signaling problem that guarantees full privacy for all recipients, while requiring only *constant* amount of work for both the recipient and the sender. We provide the following contributions. First, we provide a formal definition of private signaling in the Universal Composability (UC) framework and show that it generalizes several real-world settings where recipient anonymity is desired. Second, we present two protocols that UC-realize our definition: one using a single server equipped with a trusted execution environment, and one based on two servers that employs garbled circuits. Third, we provide an *open-source* implementation of both of our protocols and evaluate their performance and show that they are practical.

1 Introduction

Problem Statement. We focus on the problem of *recipient anonymity* in the context of private signaling. In this abstraction, there are M recipients $R_1, \ldots, R_M$ publicly identified by their public keys $pk_1, \ldots, pk_M$. There is a public venue such as a bulletin board that collects messages $(m_1, m_2, m_3, \ldots)$ from senders and are intended for recipients. The sender who posted message m_j on the board, will also post an auxiliary information c that signals the intended recipient, say R_i, that there is a message for them at a location j of the board. The problem is: how can this sender craft a signal c so that by looking at c and the board, *no one*, except R_i, can detect who the intended recipient is for m_j, with the sender having no communication or prior shared state with R_i?

This abstraction captures various concrete problems such as anonymous messaging [7] and stealth addresses[9]. We describe these specific applications in greater length in Section 3. For the remainder of the introduction we will focus on the general abstraction above.

*Alessandra Scafuro and Varun Madathil are supported by NSF grants #1718074,#1764025

Private Signaling: Efficiency Constraints. A straightforward (though inefficient) solution for the private signaling problem would be as follows. The sender who intends to communicate m_j to R_i can simply encrypt m_j with the public key pk_i using a *key-private* CPA-secure encryption scheme [1] and then only post the ciphertext c on the board. In this case, the signal is the ciphertext itself. Then, each recipient can periodically download all ciphertexts posted on the board, attempt to decrypt each ciphertext to detect the ones directed to them. Thanks to the key-privacy property of the encryption scheme, this solution gives *full* privacy to each recipient, since by looking at the ciphertext, every public key is equally likely to unlock it. Here full means that the anonymity set constitutes the entire set of (honest) recipients. Furthermore, this solution has no overhead on the sender, who simply performs one encryption per signal. However, full privacy comes with a high cost for each recipient since it needs to scan the *entire board* to detect the signal.

Efficient Private Signaling: the need for a Server. Can we do better than a linear scan of the board? First, note that without any external help, such as a server dedicated to filtering messages for each recipient, a recipient must read the entire list of, say N, signals to "see" which one is intended for them. Note that this is true regardless of the anonymity guarantees. Hence, any serverless solutions will lead to complexity $O(N)$ for each recipient. [2]

Thus, for any non-trivial improvement of the complexity cost for the recipient, we need to use an external server to help with the filtering. In a very recent work [2] Beck et al. introduced the concept of Fuzzy Message Detection (FMD), a new cryptographic primitive that allows a third party to perform coarse filtering of messages for each recipient. Coarse means that, for each recipient R_i, the server will detect ciphertexts and maintain a *list* of ciphertexts that *could* be intended for R_i. This list includes a certain fraction p_i of false positive— hence fuzzy detection. The higher the rate p_i of false positive for R_i, the longer the list of ciphertexts detected for R_i, and the higher the anonymity set for R_i. This approach, however, presents major drawbacks for the recipient. First, the work done by the recipient grows proportionally to the amount of anonymity it desires. Specifically, the work done by recipient R_i is $O(p_i \cdot N)$, which translates into $O(1 \cdot N)$ work if the highest privacy is required. Second, even if a recipient R_i chooses the highest false positive rate $p_i = 1$, this would still not guarantee R_i to have full privacy (recall, full privacy means that a signal can be associated to every (honest) recipient with the same probability) if other honest recipients have chosen smaller error rates. A natural question arises:

> *Is there a solution for the private signaling problem that achieves* full *anonymity in the presence of untrusted servers and has only* constant *complexity for the recipient?*

In this paper, we answer affirmatively. We provide two protocols: one with a single untrusted server (equipped with a trusted execution environment) and one with two untrusted (though not colluding) servers. Our contribution and techniques are discussed in Section 2.

2 Our contribution

In this paper we provide three contributions:

1. **Formalization of the Private Signaling Problem.** We introduce the *private signaling problem* and provide a formal definition in the Universal Composability Framework [6]. Thus, we define an ideal functionality $\mathcal{F}_{\mathsf{privSignal}}$ (Figure 7) that captures the correctness and privacy guarantees that we expect from a private signaling system. Previous work on related problems either did not provide any formal definition [19, 28, 15], or provide much weaker security guarantees[2].

[1] Key-private means that by looking at the ciphertext, no one can distinguish which public key was used for encrypting the message [3].

[2] Alternatively, the complexity can be made to be $\mathcal{O}(\log N)$ per message for the recipient assuming that the size of the signal blows up to $\mathcal{O}(M)$. We describe this approach in Appendix C.7

2. **Protocols for private signaling with constant recipient overhead and *provable* UC-security.** We provide two protocols that UC-realize the ideal functionality $\mathcal{F}_{\mathsf{privSignal}}$ while demanding only the minimal overhead to recipients and senders. Both protocols are built around the idea of enabling an untrusted server to *obliviously* update the list of signals for each recipient. We then show two instantiations for obliviously updating the list: one based on garbled circuits that requires two servers, and one leveraging a Trusted Execution Environment (TEE) which requires a single server only. For both protocols we provide formal proofs in the UC-framework.

3. **Open-source Implementations.** We implement both our protocols and evaluate the performance as a function of the number of recipients (M) and the upper bound of signals (ℓ) it receives in each window of time. Our proposed protocols are feasible in practice, and they incur reasonable overhead with either modest numbers of users M or limited number of messages ℓ. Despite their moderate efficiency, our solutions might be viable options even in larger applications considering the privacy benefits of our solutions, i.e., they achieve the highest possible degree of anonymity. This is in contrast with the achieved k-anonymity of [2].

In the remainder of this section, we describe each contribution in more details.

2.1 Definition of Private Signaling in the UC-framework

We define the problem of private signaling in the UC-framework [6]. In this framework, the security properties expected by a system are defined through the description of an ideal functionality. The ideal functionality is an ideal trusted party that performs the task expected by the system in a trustworthy manner. When devising an ideal functionality one describes the ideal properties that the system should achieve, as well as the information that the system will inherently leak.

For the task of private signaling, we want to capture two properties: correctness and privacy. Correctness means that a recipient R_i should be able to learn all signals that are intended for them. Privacy means that by looking at the messages exchanged in the protocol *no one* except R_i (and the senders of the signals) should distinguish which signals are directed to R_i. Furthermore, we want to capture the following inherent leakage. First, an observer of the system can always learn that a signal was posted for "someone" (for instance, just by observing the board). Second, a protocol participant can learn that a certain recipient is trying to retrieve their own signals (for instance, in the serverless case, this can be detected by observing that a node is downloading a big chunk of the board, or in the server-aided case it is just possible to observe that R_i connected to the server). We capture the above security properties and leakage in the ideal functionality $\mathcal{F}_{\mathsf{privSignal}}$, which we describe in detail in Section 6.

2.2 Efficient Protocols for Private Signaling

We present two instantiations of the ideal functionality $\mathcal{F}_{\mathsf{privSignal}}$, that achieve *constant* communication and computation complexity for the recipient. Both instantiations are based on the same high-level approach of obliviously updating the list of signals for the recipient. We explain the general approach first, and then we describe two techniques for implementing it.

Our approach is based on the following natural idea. Assume for a moment that privacy was not a concern, but only performance is, i.e., we want the overhead of the recipient to be minimal and depend only on the number of messages it receives. As illustrated Figure 1 the recipients hire a Server and give their secret keys sk_i to the Server. The sender after posting a message to the board, sends a signal which is the encryption (under the pk_i of the recipient) of the location of the message to the Server. The Server maintains a table $\mathbb{T}$, with one row for each recipient. It decrypts the signal using each recipient's secret key and adds the signal to the row of recipient for which the signal decrypted correctly. When a recipient R_i sends `RECEIVE` to the server, it simply responds with the corresponding row.

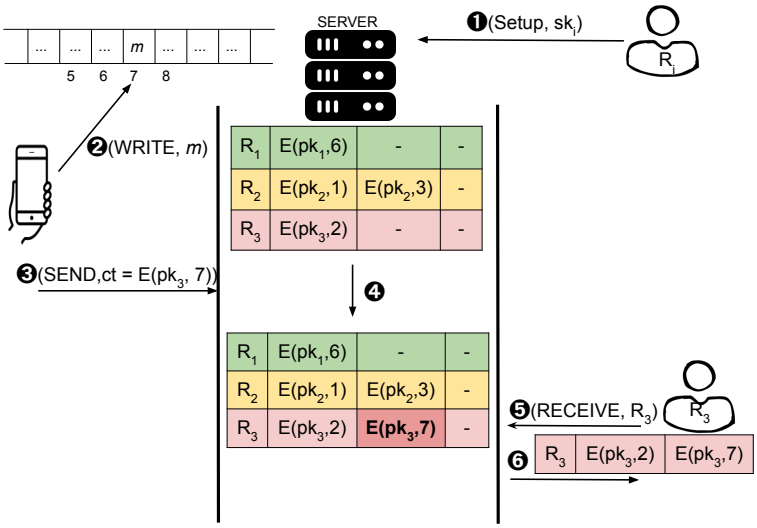

Figure 1: Naive solution example: ❶ Each recipient R_i registers with the Server and sends its sk_i. ❷ A sender writes a message m for recipient R_3 on position 7 of the board. ❸ Sender sends a signal to the Server for the posted message. The signal is an encryption of 7 under the public key of R_3. ❹ The server attempts to decrypt the signal using the secret key of each R_i until it decrypts to a valid message. The Server then adds the encryption to the next available location in R_3's row. ❺ R_3 requests its row from the Server in an authenticated way, and ❻, the Server responds with the encryptions in that row.

Now, to achieve privacy, we "just" need to require the server to update this table *obliviously*. In other words, we need to devise a mechanism by which, on input an encrypted signal for a certain recipient R_i, the server can blindly and correctly update the i-th row without learning anything about the recipient who got the signal.

2.2.1 Single-Server solution

To update the table of signals $\mathbb{T}$ obliviously by employing a single untrusted server, we leverage a trusted execution environment (TEE). Recall that a TEE allows a client to perform a private computation on a secret input, embedded in the TEE, through an untrusted server, called the host. TEEs are used to build virtual enclaves. A client can register with the enclave within the server and is guaranteed that all computations inside the enclave are hidden to the server.

With this tool in hand, the idea is that the recipients will not provide their secret keys directly to the server. Instead, R_i can securely communicate its secret keys sk_i to the enclave.

After this setup phase where recipients register with the enclave, the server will host M enclaves, one for each recipient $R_1, \ldots, R_M$. Furthermore, the server maintains M lists of ciphertexts $\vec{L}_i$ that are originally set as encryptions of 0 under keys SGXkey_i (this is the table of signals, where each $\vec{L}_i$ is a row in $\mathbb{T}$).

Each enclave will implement the following program: on input a signal ciphertext $\mathsf{ct}_{\mathsf{Signal}}$ and encryption of locations $\vec{L}_i$ for recipient R_i, first attempt to decrypt $\mathsf{ct}_{\mathsf{Signal}}$ with the secret key sk_i. If the decryption results in a valid plaintext location loc, then the enclave will update the ciphertext list $\vec{L}_i$ with a new encryption of loc (under SGXkey_i) in the next available position, and re-randomize the other ciphertexts. Else, all ciphertexts are just re-randomized. As in the naive solution, a sender can communicate a signal for location loc to R_i, by simply encrypting the location under the public key pk_i, that is, $\mathsf{ct}_{\mathsf{Signal}} = \mathsf{Enc}(\mathsf{pk}_i, \mathsf{loc})$ and send $\mathsf{ct}_{\mathsf{Signal}}$ to the server. The server will then run each

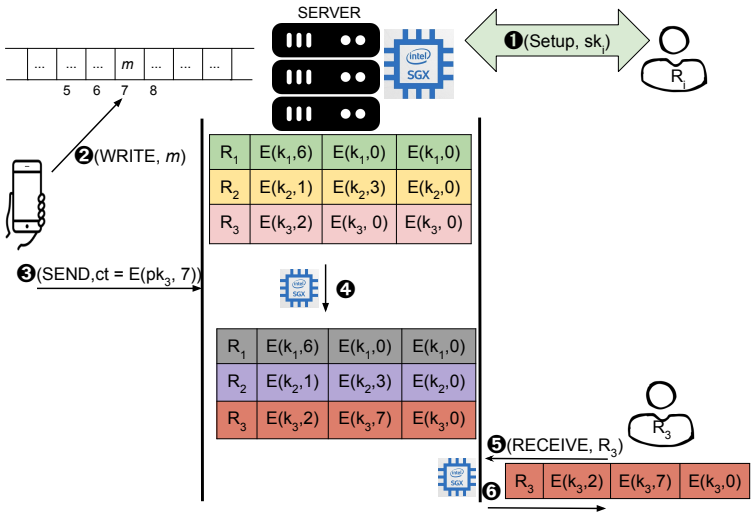

Figure 2: Single-server example: ❶ Each recipient securely and privately communicates its secret key sk_i with the enclave (only). ❷ and ❸ The sender writes a message m for recipient R_3 on position 7 of the board and sends a signal (encryption of 7 under R_3's public key) to the Server for the posted message ❹ The enclave takes as input the signal and updates each row of encryptions. It re-randomizes each encryption except for column 2 in R_3's row, where it adds an encryption of 7. Note that the encryptions of row R_i are done under a key SGXkey_i (here denoted k). ❺ R_3 sends an authenticated request for its row via the server to the enclave. ❻ If the request is valid, the Server responds first decrypts the encryptions in row R_3 and then re-encrypts them under pk_3 and sends them to R_3 via the server.

enclave on input $\mathsf{ct}_{\mathsf{Signal}}, \vec{L}_i$ for each recipient R_i to run the above-described program.

The actual protocol is slightly more complex as it requires a mechanism to prevent replay attacks from the untrusted server against the enclave. For the UC-security proof to go through, we need a mechanism to enforce that even if server and recipient are corrupt and collude, any attack is still simulatable in the ideal world. This requires a mechanism by which, to retrieve its signals, a recipient must first obtain a token from the TEE in every access. The details of the protocol are provided in Section 7 and the protocol is described in Figure 9. We formally prove that our protocol UC-realizes the ideal private signal functionality. For the formal proof, we use the UC-formalization of TEE introduced by Pass and Shi in [21], as the ideal functionality $\mathcal{G}_{\text{att}}$. Our proof is provided in Appendix B and withstands malicious adversaries (for privacy). We present an illustration of this single-server approach in Figure 2. We implement the protocol where we instantiate the TEE using Intel SGX.

Limitations of Intel SGX: The Intel SGX comes with certain limitations.

- **Trust assumptions** Trusted Execution Environments need to rely on a trusted authority (Intel in the case of Intel SGX) to design a secure processor with the correct program. If the design is flawed or Intel is corrupted then attestations from the SGX can be forged and an adversary can claim that the attestation came from a genuine piece of hardware. These assumptions are not suitable for many real-world applications especially cryptocurrencies etc.

- **Side-channel attacks** Intel SGX may be prone to side-channel attacks like cache timing [12] [5] and page table[16][29] side-channel attacks. In [12] they show that using an access-driven cache timing attack on AES that was run inside the SGX, they were able to extract the secret key. In [29] the authors present a technique called Dark-ROP that exploits memory corruption

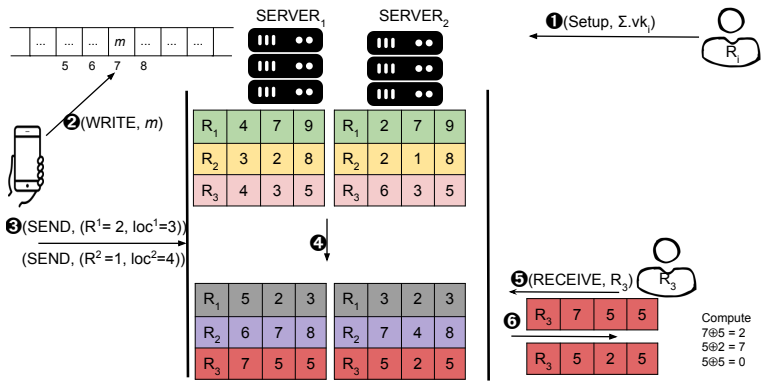

Figure 3: Two-server example: ❶ Each recipient R_i registers with the two servers. ❷ Sender writes message m for recipient R_3 on position 7 of the board. ❸ Create shares of $3 = (2, 1)$ such that $2 \oplus 1 = 3$ and $7 = (3, 4)$ such that $3 \oplus 4 = 7$ and send $(2, 3)$ to Server_1 and $(1, 4)$ to Server_2. ❹ Server_1 and Server_2 run a 2PC with inputs $(\mathbb{T}_1, 2, 3)$ and $(\mathbb{T}_2, 1, 4)$ and some fresh randomness and output new tables $\mathbb{T}_1$ and $\mathbb{T}_2$ such that in the next available of position of R_3's row (column 2), update with shares of $7 = (5$ and $2)$ (as $5 \oplus 2 = 7$) and re-randomize all other indices while maintaining the invariant that $\mathbb{T}_1[i][j] \oplus \mathbb{T}_2[i][j]$ remains the same. ❺ R_3 sends an authentic request for its rows from the two servers. ❻ Server_1 sends [7,5,5] and Server_2 sends [5,2,5]. The recipient recombines the corresponding indices $[7 \oplus 5, 5 \oplus 2, 5 \oplus 5] = [2, 7, 0]$ to compute the locations of its messages.

vulnerability in the enclave software through return-oriented programming (ROP). Thus these known attacks can be used to circumvent the data confidentiality that are guaranteed by the SGX.

- **Memory limitations** The memory of the SGX (Enclave Page Cache size) is only 96MB [8]. Therefore data must be stored in the untrusted memory of the host and be swapped with the SGX memory. This operation is expensive because of the encryption and integrity verification of the data.

2.2.2 Two-server solution

To accomplish the goal of obliviously updating the table of signals $\mathbb{T}$, we can use two servers Server_1 and Server_2 and have the table secret-shared among them. Server_1 (resp., Server_2) holds a table $\mathbb{T}_1$ (resp., $\mathbb{T}_2$) of strings that look random to Server_1(resp., Server_2), but such that $\mathbb{T}_1 \oplus \mathbb{T}_2 = \mathbb{T}$.

Say a sender S posted a message m intended for R on the board that appears in location loc. To prepare a signal for R concerning location loc the sender will perform a simple operation. It will secret-share the input R, loc into random two shares $R^{(1)}, R^{(2)}$ and $\mathsf{loc}^{(1)}, \mathsf{loc}^{(2)}$ such that $R = R^{(1)} \oplus R^{(2)}$ and $\mathsf{loc} = \mathsf{loc}^{(1)} \oplus \mathsf{loc}^{(2)}$.

Next, servers $\mathsf{Server}_1, \mathsf{Server}_2$ will update their tables by running a *secure computation protocol* (e.g., Yao's garbled circuits [30, 4]), participating with their own secret input $R^{(1)}, \mathsf{loc}^{(1)}, \mathbb{T}_1$ (resp., $R^{(2)}, \mathsf{loc}^{(2)}, \mathbb{T}_2$). The function being computed performs the following three elementary operations. (1) Reconstruct R and loc by xoring the shares. (2) Update the R-th row of the table to add loc to the first available index. (3) Re-randomize every other row. Note that, at the end of the secure computation of this function, each server receives a fresh share of the updated table, thus leaking no information about which row and column was actually updated.

Note that each recipient receives a different number of signals over time. To guarantee that no information about the number of messages per recipient is leaked, we introduce a parameter ℓ. Each row in the table is of length ℓ and each index is updated in the table per signal.

	Privacy	Security	Sender	Recipient	Server	#Servers	Setup Assumptions
Naïve scan	fully private	–	$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\emptyset$	0	$\emptyset$
FMD [2]	k-anonymity	game-based,SH	$\mathcal{O}(1)$	$\mathcal{O}(pN)$	$\mathcal{O}(pM)$	1	$\emptyset$
$\Pi_{\mathsf{privSignal1}}$	fully private	UC-secure, M	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\ell M)$	1	TEE
$\Pi_{\mathsf{privSignal2}}$	fully private	UC-secure, SH	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\ell M)$	2	$\emptyset$

Table 1: Comparing privacy-preserving message detection schemes in terms of the achieved privacy guarantees and the computational complexity of the participants. SH and M denotes semi-honest and malicious security, respectively. N denotes the total number of messages in the system and p denotes the false positive rate ($0 \leq p \leq 1$) set individually by recipients in the Fuzzy Message Detection scheme (FMD) [2]. For simplicity, we assume that each recipient has the same false positive rate p. Note that FMD does not allow asymptotically sublinear scans for recipients. The privacy guarantee of [2] is dynamic k-anonymity, where $k \approx pN$. In our constructions, ℓ is a system parameter ($\ell \in \mathbb{N}$) and denotes the maximum number of detectable incoming messages per each recipient. Finally the server computation is based on a single message received by the server(s).

The concrete value of ℓ depends on the application, and we guarantee that the recipient can always retrieve the *last* ℓ signals posted.[3]

When a recipient R_i wishes to retrieve their signals, it will send i (in an authenticating manner) to both servers and receive $\mathbb{T}_1[i], \mathbb{T}_2[i]$ from which it can recover the locations by just performing xor. Upon each retrieve, the recipient's row is flushed.

Our protocol provides full privacy due to the following features: at any point each server only owns only one share of the signals and the table of signals, and upon each update, the server obtains a re-randomization of the entire table, performed with fresh randomness that is sampled by both servers, which leaks no information about the row that was actually updated. Formally, we prove that our protocol UC-realizes the ideal functionality in Appendix C. In our proof, servers can collude with recipients and sender, but (of course) cannot collude with each other. For this protocol, our proofs are in the *semi-honest* setting. Finally we note that we can extend this idea to a multi-server setting, where say n servers participate in an MPC to process a signal and update the shares of the table of signals.

The tradeoff here would be that sender will need to share the location and recipient index among n servers and the recipient would need to recombine the shares received from n servers, but on the positive side one can have weaker assumptions on the trust and non-collusion between the servers.

2.3 Implementation and Evaluation

We implement a proof-of-concept of our protocols $\Pi_{\mathsf{privSignal1}}$ and $\Pi_{\mathsf{privSignal2}}$ and demonstrate their feasibility in Section 9. We present a comparison of our protocols and related work FMD [2] in Table 1.

Recipient and sender computation Our protocols provide *constant* communication and computation complexity for the recipient and the sender. Specifically, a recipient only needs to perform computation that is proportional to the number of signals it receives. Moreover, a sender only needs to compute constant size messages (either an encryption or two shares of a location) and send them to the server(s).

Server computation Note that even in a naive setting, where one does not care about the privacy of the recipients, a server would need to do up to $\mathcal{O}(M)$ (in the worst case) computation to determine the recipient of the encrypted signal. Likewise, in both of our protocols, the overhead of the server(s) is $\mathcal{O}(\ell M)$ (where ℓ is the number of signals a recipient can receive). Depending on the application,

[3]However, note that the scheme can be changed so that past snapshots are given to the recipients. In this case, however, efficiency for the recipient is not guaranteed.

one can choose different values for the parameters M and ℓ. In our experiments we vary both M and ℓ from 10 to 1000. From our implementation, we observe that we can trade off one parameter for the other to get a feasible efficiency for the server(s) (approx. 1 minute to process 100 signals for 100 recipients). We also observe that the single-server solution is more efficient than the two-server, but at the cost of a trusted setup (TEEs).

We also improve the efficiency of the 2-server protocol by having the servers process multiple signals at once. We noticed that we achieve 2.5× improvement in the overall computation time. This can be attributed to the fact that the communication of the garbled tables and OT need to be done only once for a group of signals.

3 Applications of Private Signaling

Private signaling is a powerful abstraction, since many real-world applications can be seen as a special case of it. In the following, we highlight two prominent and timely problems that can be cast as private signaling problems and consequently solved with our proposed solutions.

Stealth addresses and payments. In cryptocurrencies (especially account-based ones [27]) it is common to use static, public identities or addresses. However, sending recurrent payments (e.g., salaries, donations, other regular purchases) to a static address that is publicly linked to an entity is harmful to both sender and recipient anonymity. To avert this issue, senders can generate so-called stealth addresses for their recipients [9]. More specifically, given a recipient's public address, the sender can non-interactively generate new "stealth" addresses for the intended recipient that is unlinkable to the recipient's static, public address [23]. Stealth addresses can only be redeemed by the true recipients. However, the difficulty is that recipients lack an efficient way to detect which stealth address belongs to them and are redeemable by them. Current implementations of stealth address payment systems apply the simple linear scan of the board as described earlier.[4] Private signaling can be seen as a solution to alleviate the computation complexity of the recipient. More specifically, with private signaling, a sender first creates a transaction with a stealth address of recipient R_i and posts it to the board. Once the transaction is confirmed and the location of the transaction is known on the board, the sender sends a *private signal* to the server, who obliviously stores it. Now a recipient only needs to ask the server for its list of signals so it can identify its stealth address transactions directly.

Anonymous messaging. Modern private messaging applications are mostly focused on providing and improving sender anonymity [18, 7], e.g., Signal's sealed sender functionality. In anonymous messaging applications, senders post their messages to one (or more) untrusted store-and-forward server(s) [26] or to a shared public bulletin board, as in Riposte [7], where the servers need to maintain the board. Private signaling easily captures this problem in the following way: A sender first posts encrypted messages on a board. The sender then sends the locations of these messages to the server in a privacy-preserving way, such that only the recipient can retrieve the locations from the servers at a later point of time. Once the recipient has these locations it can simply decrypt the corresponding messages from the board to get their messages. Thus anonymous messaging can be seen as special case of private signaling. Moreover, using our techniques, it is guaranteed that a recipient can retrieve its messages quickly and one can have arbitrary sized messages that can be stored on the public board.

4 Related Work

Fuzzy Message Detection(FMD) [2] The closest work to ours is the fuzzy message detection [2]. We compare the two works in Table 1 and we expand here.

Privacy. In terms of privacy, our protocol achieves the strongest privacy guarantee, where each recipient has an anonymity set that is as large as the number of honest recipients and senders. On

[4] See: Umbra Cash (https://app.umbra.cash)

the other hand, the privacy guaranteed by FMD is more fragile. They achieve k-anonymity. In fact, in their evaluation they present attacks by an independent researcher on their system that show that privacy obtained is indeed dependent on the parameter selection. These attacks allow a curious server to learn that a subset of messages are for a particular recipient. More details of these attacks can be found in [17].

We also note that other anonymous message detection schemes, relying on a wisely chosen false positive rate, are already shown to be challenging to deploy. For example, Bitcoin light clients implementing the Bloom-filter-based anonymous message detection scheme, also known as BIP37 [13], were successfully deanonymized by Gervais et al. [11]. Furthermore, k-anonymity is brittle and prone to attacks, i.e., statistical attacks and intersection attacks. Therefore, in an ideal setting, every recipient should have all the participants of the communication system in their anonymity set. Accordingly, in our protocols we achieve full privacy for the recipients.

Efficiency. Both our solutions offer the best efficiency for both sender and recipients at the cost of the servers doing more work. In FMD, the senders need to compute γ (a constant of the order 10) number of encryptions and send them to the server; whereas in our one-server setting, the sender needs to send only one encryption to the server. If N is the total number of messages that were sent to the server, each recipient will receive pN messages. The recipient then would need to do γ decryptions on each of these messages to test if the message is actually for them or if it's a false positive. In our setting, the server will output ℓ messages to the recipient. The recipient then decrypts these messages until it decrypts to 0 to know it has received all the messages. Finally, in our setting, the server needs to do $O(\ell M)$ computation. That is, for each recipient, it must update ℓ encryptions. In [2], the server attempts to decrypt for each recipient using the $p\gamma$ keys of each recipient. If we assume a common p, then the server needs to do $p\gamma M$ decryptions to determine which recipients might be the recipient of the message. Our protocol is computationally more intensive for the servers since it needs to decrypt ℓM ciphertexts and re-encrypt them for every `SEND` command that is invoked.

Assumptions and Threat-model. The work of FMD relies on a much weaker assumption, which is a single untrusted server, while for our single-server solution requires the use of a trusted execution environment (TEE) [21, 8]. In FMD, the authors present game-based proofs, whereas we define an ideal functionality for private signaling and prove its security in the UC-model and achieve privacy against a malicious adversary in the single-server setting.

Privacy-preserving Light Clients A related problem to private signaling is that of devising privacy-preserving light clients for cryptocurrencies. A light client does not store the full blockchain. However, it wants to learn information about certain addresses owned by the client from full nodes that store the entire blockchain. Additionally, they wish to learn their balances, incoming transactions and other details of their addresses in a privacy-preserving way. Current approaches are either inefficient [25] (full scan of the blockchain) or transparent (a trusted node learns all the relevant balances and transactions of the light client[5]). Several privacy-preserving cryptocurrency light client proposals have been offered in the literature [22, 28, 15, 19]. In all of these works, a recipient asks one (or more) powerful server(s) to learn its balance and other relevant information about addresses the light client is interested in. To preserve the privacy of the light client, Qin et al. apply private information retrieval [22], TEEs have been proposed by Wüst et al.[28] and Matetic et al. [19], while applying ORAM in this context has been suggested by Le et al. [15]. However, crucially, these works assume that the light clients (the recipient in our setting) *already know* the addresses they want to obtain information about. This is equivalent to knowing the locations of the transactions in our setting. Whereas, in the private signaling problem we are interested in communicating the locations of the signal to the recipient. Thus, our problem and techniques are complementary to the problem of light clients, and can be used in addition to the systems proposed in [28, 19, 15] so that a recipient privately learns these addresses (without having to communicate with the sender).

[5] See: https://github.com/vtnerd/monero-lws

5 Preliminaries and Definitions

5.1 Notation

Let λ be the security parameter, $\mathsf{poly}(\cdot)$ be a polynomial function and let $\mathsf{negl}(\lambda)$ be a negligible function. M denotes the total number of recipients, and N denotes the total number of messages on the a public bulletin **board**. Finally, loc denotes a location on the **board**

In our protocols all messages are posted on a public bulletin board, that all entities have read-and-write oracle access. We define these oracle accesses below:

1. $\mathsf{ReadBoard}(\mathbf{board}, \mathsf{loc}) \rightarrow \mathbf{board}[\mathsf{loc}]$ returns the message at location loc of **board**.
2. $\mathsf{WriteBoard}(\mathbf{board}, m) \rightarrow (\mathbf{board}', \mathsf{loc})$ returns a new $\mathbf{board}'$ where the message m is appended to the end of **board**. That is $\mathbf{board}' = \mathbf{board} \| m$. Additionally, the location loc is also returned where m was written.

In this section we present the crucial definitions and security guarantees of the primitives used in our protocols. We present the rest of the primitives more formally in Appendix A.

5.2 Oblivious transfer

Oblivious transfer (OT) is a two-party protocol in which a sender S has two input strings $s_0, s_1 \in \{0,1\}^\lambda$, and a receiver R has a choice bit $b \in \{0,1\}$. An OT protocol is called non-trivial if for any pair of strings $s_0, s_1 \in \{0,1\}^\lambda$, and for any $b \in \{0,1\}$, after participating in the interactive protocol, S outputs nothing and R learns s_b. We capture this definition formally as an ideal functionality $\mathcal{F}_{\mathsf{ot}}$ in Figure 4.

Ideal Functionality $\mathcal{F}_{\mathsf{ot}}$:

- Upon receiving message $(\texttt{SEND}, s_0, s_1, S, R)$ from S, where $s_0, s_1 \in \{0,1\}^\lambda$, store s_0, s_1 and answer `SEND` to R and $\mathcal{S}$.
- Upon receiving message $(\texttt{RECEIVE}, b)$ from R, where $b \in \{0,1\}$, send s_b to R and `RECEIVE` to S and $\mathcal{S}$, and halt. If no message $(\texttt{SEND}, \cdot)$ was previously sent, do nothing.

Figure 4: Ideal functionality for oblivious transfer

5.3 Garbled circuits

We present a formal definition for garbled circuits. We present the definitions of [4].

Definition 1. *A garbling scheme $\mathcal{G}$ consists of five polynomial time algorithms* $(\mathsf{Garble}, \mathsf{Encode}, \mathsf{Eval}, \mathsf{Decode}, \mathsf{evaluate})$.

1. $\mathsf{Garble}(1^\lambda, f) \rightarrow (F, e, d)$. *The garbling algorithm* Garble *takes in the security parameter λ and a circuit f, and returns a garbled circuit F, encoding information e, and decoding information d.*
2. $\mathsf{Encode}(e, x) \rightarrow X$. *The encoding algorithm* Encode *takes in the encoding information e and an input x, and returns a garbled input X.*
3. $\mathsf{Eval}(F, X) \rightarrow Y$. *The evaluation algorithm* Eval *takes in the garbled circuit F and the garbled input X, and returns a garbled output Y.*
4. $\mathsf{Decode}(d, Y) \rightarrow y$. *The decoding algorithm* Decode *takes in the decoding information d and the garbled output Y, and returns the plaintext output y.*

procedure INITIALIZE
  Pick $b \leftarrow \{0,1\}$
procedure GARBLE$((f,x))$
  if $x \notin \{0,1\}^{f.n}$ **then**
    return $\perp$
  if $b = 1$ **then**
    $(F,e,d) \leftarrow \mathsf{Garble}(1^k, f)$
    $X \leftarrow \mathsf{Encode}(e,x)$
  else
    $y \leftarrow \mathsf{evaluate}(f,x)$
    $(F,X,d) \leftarrow \mathcal{S}(1^k, y, \phi(f))$
procedure FINALIZE
  return $b = b'$

Figure 5: The $\mathsf{PrvSim}_{\mathcal{G},\phi,\mathcal{S}}$ game

5. $\mathsf{evaluate}(f,x) \to y$. *The algorithms takes as input the description of the original function f and the initial input x and outputs the final output y.*

Correctness if $f \in \{0,1\}^*, k \in \mathbb{N}, x \in \{0,1\}^{f.n}$ and $(F,e,d) \in [\mathsf{Garble}(1^k, f)]$, then

$$\mathsf{Decode}(d, \mathsf{Eval}(F, \mathsf{Encode}(e,x))) = \mathsf{evaluate}(f,x)$$

Privacy Let $\mathcal{G} = (\mathsf{Garble}, \mathsf{Encode}, \mathsf{Decode}, \mathsf{Eval}, \mathsf{evaluate})$ be a garbling scheme, $k \in \mathbb{N}$ a security parameter and ϕ a side-information function. We present below the simulation-based notion of privacy via game $\mathsf{PrvSim}_{\mathcal{G},\phi,\mathcal{S}}$, see the definition of the game in Figure 5.

The adversary wins the game if it guesses b correctly. The advantage of the adversary is defined as

$$\mathbf{Adv}_{\mathcal{G}}^{\text{prv.sim},\phi,\mathcal{S}}(\mathcal{A},k) = 2Pr[\mathsf{PrvSim}_{\mathcal{G},\phi,\mathcal{S}}^{\mathcal{A}}(\lambda)] - 1$$

and protocol $\mathcal{G}$ is prv.sim secure over ϕ if for every polynomial time adversary $\mathcal{A}$ there is a polynomial time algorithm $\mathcal{S}$ such that $\mathbf{Adv}_{\mathcal{G}}^{\text{prv.sim},\phi,\mathcal{S}}(\mathcal{A},k)$ is negligible.

Projective scheme In our schemes we consider a *projective* garbling scheme. Thus e consists of $2n$ wire labels, where n is the number of input bits. We denote these wire labels as $(X_i^0, X_i^1)_{i \in \mathsf{indices}}$. $\mathsf{Encode}(e, x = (v_i)_{i \in \mathsf{indices}})$ returns $X = (X_i^{v_i})_{i \in \mathsf{indices}}$.

5.4 Attested Execution Processers

In this section we present details on the formalization of attested execution processers as described in [21] and presented in Figure 6.

Initialization Upon initialization, a manufacturer chooses a public verification key and signing key pair denoted $(\mathsf{mpk}, \mathsf{msk})$, for the signature scheme Σ. All attestations later will be done using msk.

The registry $\mathcal{G}_{\text{att}}$ is parameterized by a signature scheme Σ and a global registry reg which contains the list of all parties that are equipped with an attested execution processor. In our setting, only the Server is in the registry reg.

Public interface $\mathcal{G}_{\text{att}}$ provides a public interface such that any party is allowed to query and obtain the public key mpk.

$\mathcal{G}_{\text{att}}[\Sigma, \mathsf{reg}]$

//initialization
On initialize: $(\mathsf{mpk}, \mathsf{msk}) := \Sigma.\mathsf{KeyGen}(1^\lambda), T = \emptyset$
// public query interface:
On receive* getpk() for some $\mathcal{P}$: send mpk to $\mathcal{P}$

Enclave operations

//local interface – install an enclave:
On receive* install(idx, Prog) from some $\mathcal{P} \in \mathsf{reg}$:

- if $\mathcal{P}$ is honest, assert $idx = sid$
- generate nonce $eid \in \{0,1\}^\lambda$, store $T[eid, \mathcal{P}] := (idx, \mathsf{Prog}, \vec{0})$, send eid to $\mathcal{P}$.

//local interface – resume an enclave:
On receive* resume(eid, inp) from some $\mathcal{P} \in \mathsf{reg}$:

- let $(idx, \mathsf{Prog}, \mathsf{mem}) := T(eid, \mathcal{P})$, abort if not found
- let $(\mathsf{outp}, \mathsf{mem}) := \mathsf{Prog}(\mathsf{inp}, \mathsf{mem})$, update $T[eid, \mathcal{P}] := (idx, \mathsf{Prog}, \mathsf{mem})$
- let $\sigma := \Sigma.\mathsf{Sig}_{\mathsf{msk}}(idx, eid, \mathsf{Prog}, \mathsf{outp})$ and send (outp, σ) to $\mathcal{P}$.

Figure 6: Global functionality modeling SGX-like secure processor [21]

Local interface When a machine $\mathcal{P}$ calls an install instruction to $\mathcal{G}_{\text{att}}$, it asserts that $\mathcal{P}$ is in reg. This models the fact that for a remote party to interact with $\mathcal{P}$'s trusted processor, all commands have to be passed through the intermediary $\mathcal{P}$. They formalize two types of invocations to the trusted hardware.

- *Installation* Enclave installation establishes a software enclave with program Prog, linked to some identifier idx . The functionality enforces that honest hosts provide the session identifier of the current protocol instance as idx. $\mathcal{G}_{\text{att}}$ further generates a random identifier (or nonce) eid for each installed enclave, which can later be used to identify the enclave upon resume. Finally, $\mathcal{G}_{\text{att}}$ returns the generated enclave identifier eid to the caller.
- *Stateful resume* An installed enclave can be resumed multiple times carrying state across these invocations. Each invocation identifies the enclave to be resumed by its unique eid. The enclave program Prog is then run over the given input, to produce some output (together with an updated memory mem). The enclave then signs an attestation, attesting to the fact that the enclave with session identifier idx and enclave identifier eid was installed with a program Prog, which was then executed on some input to produce outp.

6 Private Signaling

In this section, we present a formal definition for private signaling over a public bulletin board.

6.1 Threat model

In protocol $\Pi_{\mathsf{privSignal1}}$ (Section 7) we consider a single-server protocol that relies on a TEE. In this setting, the adversary has full control over the operations of the TEE. Even though the adversary controls the TEE, we assume it cannot break the hardware enforcement of the TEE. That is, the adversary cannot access the keys that are processer-specific msk and nor can it access the keys that are stored inside the TEE. This is in spite of the known attacks on the Intel SGX. Moreover, the adversary cannot access any runtime specific memory that the TEE uses since they are encrypted by the TEE. We also assume that the Server can collude with a recipient R and a sender S. The adversary is allowed to delay and discard messages and can choose to stop the execution of the TEE and therefore we prove that only privacy is guaranteed against such a malicious adversary.

In protocol $\Pi_{\mathsf{privSignal2}}$ (Section 8) we consider a two-server protocol where we do not allow the two servers to collude. We allow any other collusion between a single server and recipients and senders. Therefore in this setting we consider a semi-honest adversary that follows the protocol.

In both settings when we consider a corrupt sender, we only consider an adversary that aims to break the privacy of the system. For example, a corrupt sender could send a malformed location or overload the system with many signals for a particular recipient. We do not prevent these attacks in our protocols.

6.2 Communication model

In our protocols we do not allow any out-of-band communication between the senders and the recipients. All entities have read and write access to the **board**. We assume that senders and recipients have direct channels with the Server(s) and all messages are delivered with with a maximum delay under the bounded synchronous communication setting. In particular, in protocol $\Pi_{\mathsf{privSignal2}}$ we assume signals received by the two servers are in the same order.

Functionality $\mathcal{F}_{\mathsf{privSignal}}$
The functionality maintains a table denoted $\mathbb{T}$ indexed by recipient R_j, that contains information on the locations of signals for the corresponding recipient. The functionality maintains a list **board** to which all parties have read and write access.

- Upon receiving $(\texttt{WRITE}, m)$ from a sender S_i, send $(\texttt{WRITE}, (S_i, m))$ to $\mathcal{A}$. Upon receiving $\texttt{ok}$ from $\mathcal{A}$, update $\mathbf{board} = \mathbf{board} \| m$.
- Upon receiving $(\texttt{SEND}, R_j, \mathsf{loc})$ from a sender S_i, send $(\texttt{SEND}, S_i)$ to the adversary. Upon receiving $(\texttt{SEND}, \texttt{ok})$ from the adversary, append loc to $\mathbb{T}[R_j]$.
- Upon receiving $(\texttt{RECEIVE})$ from some R_j, send $(\texttt{RECEIVE}, R_j)$ to the adversary. Upon receiving $(\texttt{RECEIVE}, \texttt{ok})$ from the adversary, send $(\texttt{RECEIVE}, \mathbb{T}[R_j])$ to the recipient R_j and update $\mathbb{T}[R_j] = []$
- Upon receiving $(\texttt{READ}, \mathsf{loc})$ from any party (or the adversary) return $(\texttt{READ}, \mathbf{board}[\mathsf{loc}])$ to the party.

Figure 7: Private Signaling functionality

6.3 Private Signaling Ideal Functionality

The functionality $\mathcal{F}_{\mathsf{privSignal}}$ (Figure 7) provides the following interface - $\texttt{WRITE}$, $\texttt{SEND}$, $\texttt{RECEIVE}$ and $\texttt{READ}$. The ideal functionality allows parties to post messages on the **board** using the $\texttt{WRITE}$ command. To

add a signal for a recipient R_j, a sender simply sends `SEND` command with the pair (R_j, loc) to the ideal functionality. The latter will store this information for R_i in a table denoted $\mathbb{T}$, and will send to the adversary this information that a signal has been posted. This leakage captures the fact that in real life it is easy for an observer to detect that some sender is trying to send a message to some recipient. However this is the only information that anyone (except the sender, of course) will ever learn.

A recipient R_j can later query the ideal functionality to retrieve the signals that were sent to them. This is done using the `RECEIVE` command. This command also instructs the functionality to flush the row $\mathbb{T}[R_j]$. The ideal functionality will return the list to R_j and will inform the adversary that R_j has downloaded its private list of signals. Again, this captures the fact that in a real-world system a global observer can detect the fact that a certain device is trying to retrieve their signals (e.g., by observing the traffic).

Since the only information leaked to the adversary is that a sender has posted a signal and that a recipient has retrieved its signals we capture the privacy requirement of private signaling.

7 Private Signaling with one server

In this section, we present a protocol for private signaling with one server. In this setting, we assume that the server runs a trusted hardware processor.

Protocol Overview The Server runs a trusted hardware processor which we capture using the $\mathcal{G}_{\text{att}}$ functionality. This hardware processor runs the program Prog that is described in Figure 8. Note that only the Prog is presented in this section, the hardware processor $\mathcal{G}_{\text{att}}$ attests (see Figure 6) to any computation that is done inside the processor and outputs a signed message to the server.

A recipient must first register with the server. This includes a key exchange with $\mathcal{G}_{\text{att}}$, so that the recipient and $\mathcal{G}_{\text{att}}$ are guaranteed private communication. Once the private channel has been set up, the recipient R_i then sends an encryption secret key (sk_i) and a signature verification key vk_i to the Server who forwards it to $\mathcal{G}_{\text{att}}$. The processor then creates a vector $\vec{L}_i$ of encryptions of 0. This vector will store the encryptions of the locations on the board for the recipient. The vector is encrypted under a new symmetric key that is generated by the $\mathcal{G}_{\text{att}}$ functionality and is denoted as SGXkey_i. Moreover, these encryptions ($\vec{L}$) are stored on the Server and only the keys are maintained inside the trusted processor.

Now, when a sender S creates a signal (which is just an encryption under the recipient's pk of the location of the message) and sends it to the Server, the Server runs the trusted processor on this input along with each $\vec{L}_i$ that was stored previously. Note that the server does not know which R_i the signal corresponds to and therefore must run the processor with each $\vec{L}_i$.

Now for the processor to recognize the correct recipient, the processor decrypts $\mathsf{ct}_{\mathsf{Signal}}$ with the secret key (sk_i) corresponding too each $\vec{L}_i$ and checks if the first λ bits of the decrytion are zeros.

$\mathcal{G}_{\text{att}}$	Secure processor functionality Figure 6
$\vec{L}_i$	Encrypted locations for R_i
SGXkey_i	Symmetric key to encrypt $\vec{L}_i$ in persistent storage
$\mathsf{ct}_{\mathsf{keys}}$	Encryption of decryption key and verification key
index_i	Next available index in $\vec{L}_i$
ctr_i	Counter to prevent replayability of signatures
$\mathsf{ct}_{\mathsf{Signal}}$	Encryption of loc
$\mathsf{ct}_{\mathsf{loc}}$	Encrypted locations returned on `RECEIVE`
$\mathsf{mpk}, \mathsf{msk}$	Attestation keys of $\mathcal{G}_{\text{att}}$ functionality

Table 2: Notations for $\Pi_{\mathsf{privSignal1}}$

Such properties are guaranteed by encryption schemes such as RSA-OAEP [10]. For the purpose of presentation, we are explicit about the padding for the encryption as well as the decryption.

The processor then outputs a new $\vec{L}$ such that if the signal was meant for a recipient R_i then its $\vec{L}_i$ would have an encryption of loc in the next available index and all other indices re-randomized, and if the signal is not meant for the recipient R_i then all the indices of $\vec{L}_i$ are simply re-randomized. Note that since $\vec{L}$ of every recipient is updated and no adversary can tell the difference between an actual encryption and a re-randomization of an existing encryption, privacy of the signals are maintained. Note that this encryption is done under the symmetric key SGXkey_i that is known only to the processor.

On input $(\text{“keyex”}, g^a)$: $b \leftarrow_\$ \mathbb{Z}_p$, store $k = (g^a)^b$, return (g^a, g^b). Generate $\mathsf{SGXkey} \leftarrow \mathsf{PrivEnc.KeyGen}(1^\lambda)$ and store SGXkey.

On input $(\text{“setup”}, \mathsf{ct_{keys}})$

Compute $(\mathsf{sk}, \Sigma.\mathsf{vk}) = \mathsf{Dec}(k, \mathsf{ct_{keys}})$ and let corresponding public key of sk be pk.
Compute $\vec{L} = \{\mathsf{Enc}(\mathsf{SGXkey}, 0)\}_{j=0}^{\ell}$
Set $\mathsf{index} = 0$ and $\mathsf{ctr} = 0$
return $(\mathsf{pk}, \vec{L})$ and store $(\mathsf{pk}, \mathsf{sk}, \Sigma.\mathsf{vk}, \mathsf{index}, \mathsf{ctr})$.

On input* $(\text{“send”}, \vec{L}, \mathsf{ct_{Signal}})$

Read $(\mathsf{pk}, \mathsf{sk}, \Sigma.\mathsf{vk}, \mathsf{index}, \mathsf{ctr})$ from internal memory
Let $\mathsf{msg} = \mathsf{Dec}(\mathsf{sk}, \mathsf{ct_{Signal}})$.
if $\mathsf{msg}[0:\lambda] = 0^\lambda$ **then**
  Update $\mathsf{index} = (\mathsf{index} + 1) \mod \ell$
  for i in $[1, \ell]$ **do**
    Let $\mathsf{curr} = \mathsf{Dec}(\mathsf{SGXkey}, \vec{L}[i])$
    if $i = \mathsf{index}$ **then**
      $\vec{L}[i] = \mathsf{Enc}(\mathsf{SGXkey}, \mathsf{msg})$
    else
      $\vec{L}[i] = \mathsf{Enc}(\mathsf{SGXkey}, \mathsf{curr})$
else
  for i in $[1, \ell]$ **do**
    $\vec{L}[i] = \mathsf{Rerandomize}(\vec{L}[i])$
return $\vec{L}$

On input* $(\text{“receive”}, \mathsf{ctr}, \sigma, \vec{L})$

if $\Sigma.\mathsf{Ver}(\Sigma.\mathsf{vk}, \mathsf{ctr}', \sigma) = 1$ and $\mathsf{ctr} = \mathsf{ctr}'$ **then**
  Compute $\mathsf{loc}_i = \mathsf{Dec}(\mathsf{SGXkey}, \vec{L}[i])$ for $i \in [1, \ell]$
  Compute $\vec{\mathsf{ct}}_{\mathsf{loc}} = \mathsf{Enc_{pk}}(\mathsf{loc}_1), \ldots, \mathsf{Enc_{pk}}(\mathsf{loc}_\ell)$
  Update $\mathsf{ctr} = \mathsf{ctr} + 1$
  Update $\vec{L} = \{\mathsf{Enc}(\mathsf{SGXkey}, 0)\}_{j=0}^{\ell}$
  return $(\vec{L}, \vec{\mathsf{ct}}_{\mathsf{loc}})$
else
  return $\perp$

Figure 8: Program run by $\mathcal{G}_{\text{att}}$

At a later point in time, when a recipient R_i requests from the server for its signals, the server sends the corresponding $\vec{L}_i$ to the secure processor. The processor first authenticates that the request is valid, then it decrypts the $\vec{L}_i$ to compute the list of locations for R_i. Finally, using the recipient's public key pk_i, the processor computes the encryptions of the locations: $(\mathsf{ct_{loc}})$ and sends it to the

Setup

recipient R_i:

1. Let $a \leftarrow_\$ \mathbb{Z}_p$, $\mathsf{mpk} := \mathcal{G}_{\mathrm{att}}.\mathsf{getpk}()$
2. Send $(\text{“keyex”}, g^a)$ to Server, await $(eid_i, \mathsf{Prog}, (g^a, g^b)), \sigma)$ from Server
3. Assert $\Sigma.\mathsf{Ver}_{\mathsf{mpk}}((eid_i, \mathsf{Prog}, (g^a, g^b)), \sigma) = 1$. Let $k_i := (g^b)^a$.
4. Compute $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Enc.KeyGen}(1^\lambda)$, $(\Sigma.\mathsf{sk}_i, \Sigma.\mathsf{vk}_i) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$. Set $\mathsf{ct}_{\mathsf{keys},i} = \mathsf{Enc}(k_i, (\mathsf{sk}_i, \Sigma.\mathsf{vk}_i))$ and send $(\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i})$ to Server, and await $((eid_i, \mathsf{pk}_i, \vec{L}_i), \sigma)$ from Server. Assert $\Sigma.\mathsf{Ver}_{\mathsf{mpk}}((eid_i, \mathsf{pk}_i, \vec{L}_i), \sigma) = 1$ and publish pk_i. Initialize $\mathsf{ctr}_i = 0$.

Server:

1. Upon receiving $(\text{“keyex”}, g^a)$ from R_i. Let $eid_i := \mathcal{G}_{\mathrm{att}}.\mathsf{install}(\mathsf{Prog})$. Let $((g^a, g^b), \sigma) := \mathcal{G}_{\mathrm{att}}.\mathsf{resume}(eid_i, (\text{“keyex”}, g^a))$ and send $(eid_i, \mathsf{Prog}, (g^a, g^b)), \sigma)$ to R_i
2. Upon receiving $(\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i})$ from R_i, let $((\mathsf{pk}_i, \vec{L}_i), \sigma) = \mathcal{G}_{\mathrm{att}}.\mathsf{resume}(eid_i, (\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i}))$. Send $((eid_i, \mathsf{pk}_i, \vec{L}_i), \sigma)$ to R_i.

Procedure $(\texttt{WRITE}, m)$

1. Sender S: Call $\mathsf{WriteBoard}(m)$ and receive $(\mathsf{loc}, \mathbf{board}')$

Procedure $(\texttt{SEND}, R_i, \mathsf{loc})$

1. Sender S, computes $\mathsf{ct}_{\mathsf{Signal}} = \mathsf{Enc}(\mathsf{pk}_i, 0^\lambda \| \mathsf{loc})$ and send $(\texttt{SEND}, \mathsf{ct}_{\mathsf{Signal}})$ to Server.
2. Server: Upon receiving $(\texttt{SEND}, \mathsf{ct}_{\mathsf{Signal}})$ from some S: For $j \in [1, M]$, call $\mathcal{G}_{\mathrm{att}}.\mathsf{resume}(eid_j, (\text{“send”}, \vec{L}_j, \mathsf{ct}_{\mathsf{Signal}}))$ and receive an updated $\vec{L}_j$.

Procedure RECEIVE

Recipient R_i:

1. Compute $\sigma_i = \mathsf{Sig}(\Sigma.\mathsf{sk}_i, \mathsf{ctr}_i)$ and send $(\texttt{RECEIVE}, \mathsf{ctr}_i, \sigma_i)$ to Server. Await $((eid_i, \vec{L}_i, \vec{\mathsf{ct}}_{\mathsf{loc},i}), \sigma_T)$ from Server
2. Assert $\Sigma.\mathsf{Ver}_{\mathsf{mpk}}((eid_i, \vec{L}_i, \vec{\mathsf{ct}}_{\mathsf{loc},i}), \sigma_T) = 1$
3. Initialize $\mathsf{locns} = [], j = 0$
   while $(\mathsf{loc}_j = \mathsf{Dec}(\mathsf{sk}_i, \vec{\mathsf{ct}}_{\mathsf{loc}}[j])) \neq 0$ **do**
   $\mathsf{locns}.add(\mathsf{loc}_j)$
   $j = j + 1$
   return locns.

Server:

1. Upon receiving $(\texttt{RECEIVE}, \mathsf{ctr}_i, \sigma_i)$ from R_i, let $((eid_i, \vec{L}_i, \vec{\mathsf{ct}}_{\mathsf{loc},i}), \sigma_T) = \mathcal{G}_{\mathrm{att}}.\mathsf{resume}(eid_i, (\text{“receive”}, \mathsf{ctr}_i, \sigma_i, \vec{L}_i))$.
2. Send $((eid_i, \vec{L}_i, \vec{\mathsf{ct}}_{\mathsf{loc},i}), \sigma_T)$ to R_i and update $\vec{L}_i$

Procedure $(\texttt{READ}, \mathsf{loc})$ Call $\mathsf{ReadBoard}(\mathbf{board}, \mathsf{loc})$ and receive $\mathbf{board}[\mathsf{loc}]$

Figure 9: The protocol for private signaling in the $\mathcal{G}_{\mathrm{att}}$ hybrid world

Server who forwards it to R_i. The recipient decrypts these to receive locations of its signals. Meanwhile the secure processor will also have sent a new $\vec{L}$ which are encryptions of 0 (under SGXkey_i) to the server for storage.

On the parameter ℓ We note that the parameter ℓ denotes how many signals can be received by each recipient. If a recipient gets more than ℓ signals then $\vec{L}$ is overwritten and the recipient will lose some of the signals. Moreover, we can allow different ℓs for different recipients, where a recipient R_i can specify its ℓ value. This can be a service that the Server provides depending on a price the recipient pays. Note that different ℓs for different recipients does not affect the privacy guarantees of the protocol.

Theorem 1. *Assume that the signature scheme* Σ *is existentially unforgeable under chosen message attacks, the DDH assumption holds in the chosen algebraic group, the encryption scheme* Enc *is CPA secure and key-private. Then the* $\mathcal{G}_{att}$*-hybrid protocol* $\Pi_{\mathsf{privSignal1}}$ *UC realizes* $\mathcal{F}_{\mathsf{privSignal}}$ *against PPT adversaries.*

Proof. In our proofs we consider different cases one of which is when a corrupt sender may collude with corrupt server and recipient and tries to learn the recipients an locations of honest signals. We present below the crucial strategies of the simulation:

- At all times each $\vec{L}_i$ stored in the persistent storage will be encryptions of 0s. The simulator internally simulates the $\mathcal{G}_{\text{att}}$ functionality.
- *Send*: When the simulator receives a "send" command from $\mathcal{A}$ (on behalf of Server) the it can decrypt $\mathsf{ct}_{\mathsf{Signal}}$ to learn the location and the recipient of the signal. The simulator then sends the `SEND` command to the $\mathcal{F}_{\mathsf{privSignal}}$ functionality to simulate the transcript in the ideal world.
- *Receive*: When a corrupt recipient R_i requests the server for its signals, the server first runs the enclave ($\mathcal{G}_{\text{att}}$) on $\vec{L}_i$ and the request from the recipient. Since the simulator simulates the $\mathcal{G}_{\text{att}}$ functionality it learns of this request from the corrupt recipient. The simulator then sends `RECEIVE` to the $\mathcal{F}_{\mathsf{privSignal}}$ functionality on behalf of the corrupt recipient and receives back the correct locations. The simulator then encrypts these locations under the public key of the recipient R_i and sends it back to the Server.

For the full proofs we refer the reader to Appendix B. □

8 Private signaling with two servers

In this section we present the protocol $\Pi_{\mathsf{privSignal2}}$ where we assume two non-colluding servers. We first present an overview of the protocol below. The protocol is formally described in Figure 10.

Overview of the protocol We first present an overview of $\Pi_{\mathsf{privSignal2}}$. As described earlier in Section 5 all parties can access a public **board** using the ReadBoard and WriteBoard commands. Local to each Server_i is a table that stores information on the signals and is denoted as $\mathbb{T}^{(i)}$. The tables are $M \times \ell$ matrices where each row is associated with a recipient and ℓ is the maximum number of signals that can be received by each recipient. Moreover, while writing to the tables, the servers must know which index of the row to write the location to for a recipient. To this end, the servers maintain another table denoted $\mathbb{L}^{(i)}$, such that $\mathbb{L}^{(1)}[R] \oplus \mathbb{L}^{(2)}[R]$ stores the next available index for recipient R.

As a warm-up, we first consider a simpler case when a party can receive only one signal. We will then describe how to handle multiple signals. The tables are initialized such that the same cell in the two tables contain shares of zero, i.e. $\mathbb{T}^{(1)}[i] \oplus \mathbb{T}^{(2)}[i] = 0$.

Recall that a signal informs a recipient R that a message exists for the recipient at a location loc on the **board**. To this end, input to Server_a is $\mathsf{loc}^{(a)}$ and $R^{(a)}$, such that $\mathsf{loc}^{(1)} \oplus \mathsf{loc}^{(2)} = \mathsf{loc}$ and

$\mathbb{T}^{(i)}$	Table ($M \times \ell$) of locations maintained by Server_i
$\mathbb{L}^{(i)}$	Table of available indices denoted index
$R^{(i)}$	Share of R received by Server_i
$\mathsf{loc}^{(i)}$	Share of loc received by Server_i
$r^1_{(i,j)}, r^2_{(i,j)}$	Randomness used for both $\mathbb{T}^{(1)}[i][j]$ and $\mathbb{T}^{(2)}[i][j]$
$r^1_{(i)}, r^2_{(i)}$	Randomness used for both $\mathbb{L}^{(1)}[i]$ and $\mathbb{L}^{(2)}[i]$

Table 3: Notations for $\Pi_{\mathsf{privSignal2}}$

$R^{(1)} \oplus R^{(2)} = R$. The two servers then run a MPC protocol such that $\mathbb{T}^{(1)}[R] \oplus \mathbb{T}^{(2)}[R] = \mathsf{loc}$ and all other cells (for other R') are updated such that $\mathbb{T}^{(1)}[R'] \oplus \mathbb{T}^{(1)}[R']$ remains the same.

Now we proceed to describe the extension such that the servers can store more than one loc. Note that the idea described above does not work directly since the same cell may be overwritten in case R receives more than one signal. To this end, each row of the table will have ℓ cells, where ℓ is a system parameter and determines the maximum number of signals that can be retrieved by a recipient. At the end of a `RECEIVE` command, the recipient's row is *flushed* so that it can receive new signals.

To ensure that a loc is not written to a cell that already contains information from a previous signal, we have the two server maintain tables $\mathbb{L}^{(1)}$ and $\mathbb{L}^{(2)}$ such that $\mathbb{L}^{(1)}[R] \oplus \mathbb{L}^{(2)}[R]$ determines the last index in row R that was updated with shares of a loc. In details, the two servers now run an MPC protocol with inputs $\mathbb{T}^{(1)}, \mathbb{L}^{(1)}\ \mathsf{loc}^{(1)}, R^{(1)}$ and $\mathbb{T}^{(2)}, \mathbb{L}^{(2)}, \mathsf{loc}^{(2)}, R^{(2)}$ (where $\mathbb{L}^{(1)}[R] \oplus \mathbb{L}^{(2)}[R]$ is some index, $R^{(1)} \oplus R^{(2)} = R$ and $\mathsf{loc}^{(1)} \oplus \mathsf{loc}^{(2)} = \mathsf{loc}$) and outputs a new $\mathbb{T}^{(1)}, \mathbb{T}^{(2)}, \mathbb{L}^{(1)}, \mathbb{L}^{(2)}$ such that $\mathbb{T}^{(1)}[R][\mathsf{index}] \oplus \mathbb{T}^{(2)}[R][\mathsf{index}] = \mathsf{loc}$ and $\mathbb{L}^{(1)}[R] \oplus \mathbb{L}^{(2)}[R] = \mathsf{index}+1$. All other cells in $\mathbb{T}^{(1)}, \mathbb{T}^{(2)}$, $\mathbb{L}^{(1)}, \mathbb{L}^{(2)}$ are also updated while maintaining the invariant that $\mathbb{T}^{(1)} \oplus \mathbb{T}^{(2)}$ and $\mathbb{L}^{(1)} \oplus \mathbb{L}^{(2)}$ remain the same. Note that this update is necessary so that no information on which party received a signal is leaked to either of the servers. Finally, we note that if a recipient were to receive more than ℓ messages before it sends a `RECEIVE` command, then the new locations are overwritten to the corresponding row the recipient.

Our protocol makes use standard cryptographic primitives namely - an ideal oblivious transfer functionality: $\mathcal{F}_{\mathsf{ot}}$ defined in Appendix 5.2, garbled circuits as defined in Appendix 5.3 and finally EUF-CMA signatures (defined in Appendix A.2).

Setup In our protocol, each recipient R_i registers with the two servers by computing shares to a vector of $\ell + 1$ zeros. We denote these shares as $r_0 \dots r_\ell$. The servers use these shares and update the tables: Add a row to $\mathbb{T}^{(a)}$ and $\mathbb{L}^{(a)}$ - $\mathbb{T}^{(a)}[R] = [r_1 \dots r_\ell]$ an $\mathbb{L}^{(a)}[R] = r_0$. The recipient R_i also sets a counter denoted ctr_i. The ctr_i is updated each time, the recipient invokes a `RECEIVE` command. The ctr_i along with the vectors are signed by the recipient and sent to the servers. We will describe the use of ctr_i ahead.

Sending a signal To send a signal to recipient R_i that a message exists for R_i in location loc on the **board**, the sender creates shares of R_i denoted $R^{(1)}$ and $R^{(2)}$ such that $R^{(1)} \oplus R^{(2)} = R_i$ and creates shares of loc denoted $\mathsf{loc}^{(1)}$ and $\mathsf{loc}^{(2)}$ such that $\mathsf{loc}^{(1)} \oplus \mathsf{loc}^{(2)} = \mathsf{loc}$. The sender then sends $(R^{(1)}, \mathsf{loc}^{(1)})$ and $(R^{(2)}, \mathsf{loc}^{(2)})$ to Server_1 and Server_2 respectively.

Upon receiving the shares of R_i and loc the two servers run a 2PC protocol so that the $\mathbb{T}^{(1)}$ and $\mathbb{T}^{(2)}$ are updated such that in the next available index on $\mathbb{T}^{(a)}[R_i]$, the share $\mathsf{loc}^{(a)}$ is added. We implement this 2PC using garbled circuits, where the function implemented by the circuit is defined in Figure 12. After the 2PC is complete each cell in the table would have been updated. One cell in R_i's row in either table would be updated with a share of the loc and all other cells in the table would just have been re-randomized, such that for any shares in the same cell in both tables combine to result in the same value before.

Receiving a signal A recipient R_i receives a signal when it requests its row from the two servers.

Setup
Recipient R_i:

1. $(\Sigma.\mathsf{sk}_i, \Sigma.\mathsf{vk}_i) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$ and publish $\Sigma.\mathsf{vk}_i$.
2. Randomly sample $(r_0, \ldots, r_\ell) \leftarrow \mathbb{Z}_q^{\ell+1}$, and initialize $\mathsf{ctr}_i = 0$.
3. Compute $\sigma_i = \Sigma.\mathsf{Sig}(\Sigma.\mathsf{sk}_i, ((r_0 \ldots, r_\ell), \mathsf{ctr}_i))$ and send $(\mathbf{Setup}, (r_0 \ldots, r_\ell), \mathsf{ctr}_i), \sigma_i)$ to Server_1 and Server_2.

Server_a, for $a \in \{1, 2\}$:

1. If $\Sigma.\mathsf{Ver}(\Sigma.\mathsf{vk}_i, (r_0 \ldots, r_\ell), \mathsf{ctr}_i), \sigma_i) \neq 1$, ignore.
2. Else store ctr_i and set $\mathbb{T}^{(a)}[R_i] = (r_1, \ldots, r_\ell)$ and $\mathbb{L}^{(a)}[R_i] = r_0$.

Procedure $(\texttt{WRITE}, m)$

1. Sender S: Call $\mathsf{WriteBoard}(m)$ and receive $(\mathsf{loc}, \mathbf{board}')$

Procedure $(\texttt{SEND}, R, \mathsf{loc})$
Sender S:

1. Compute $R^{(1)}$ and $R^{(2)}$ such that $R = R^{(1)} \oplus R^{(2)}$.
2. Compute $\mathsf{loc}^{(1)}$ and $\mathsf{loc}^{(2)}$ such that $\mathsf{loc} = \mathsf{loc}^{(1)} \oplus \mathsf{loc}^{(2)}$
3. Let $\mathsf{Signal}_1 = (R^{(1)}, \mathsf{loc}^{(1)})$ and $\mathsf{Signal}_2 = (R^{(2)}, \mathsf{loc}^{(2)})$. Send Signal_a to Server_a.

Server_1 and Server_2 participate in protocol $\mathsf{processSignal}$ and update $(\mathbb{T}^{(1)}, \mathbb{L}^{(1)})$ and $(\mathbb{T}^{(2)}\ \mathbb{L}^{(2)})$ respectively.
Procedure RECEIVE
Recipient R_i:

1. Randomly sample $(r_0, \ldots, r_\ell) \leftarrow \mathbb{Z}_q^{\ell+1}$.
2. Send $\sigma_i = \Sigma.\mathsf{Sig}(\Sigma.\mathsf{sk}_i, (r_0, \ldots, r_\ell), a)$ to Server_a
3. Upon receiving $\mathbb{T}^{(a)}[R_i]$, for $j \in [1, \ell]$ compute $\mathbb{T}^{(1)}[R][j] \oplus \mathbb{T}^{(2)}[R][j]$ until $\mathbb{T}^{(1)}[R][j] \oplus \mathbb{T}^{(2)}[R][j] = 0$.
4. Update $\mathsf{ctr}_i = \mathsf{ctr}_i + 1$

Server_a:
Check if $\Sigma.\mathsf{Ver}(\Sigma.\mathsf{vk}_i, (\mathsf{ctr}', (r_0, \ldots, r_\ell)), \sigma_i) = 1$ and $\mathsf{ctr}' = \mathsf{ctr}_i$, if yes, send $\mathbb{T}^{(a)}[R]$ to R. And update $\mathsf{ctr}_i = \mathsf{ctr}_i + 1$
Procedure READ

1. Call $\mathsf{ReadBoard}(\mathbf{board}, \mathsf{loc})$ and receive $\mathbf{board}[\mathsf{loc}]$

Figure 10: Private signaling protocol with 2 servers

The recipient can simply recombine the two shares at the corresponding indices and get a vector of locations where it has received messages on the **board**. Note that the recipient needs to recombine the shares only until the shares recombine to 0, since it knows that there are no more signals after

Protocol processSignal

Server_a (where $a \in \{1,2\}$), upon receiving σ_a from S:

1. Parse $\mathsf{Signal}_a = (R^{(a)}, \mathsf{loc}^{(\mathsf{a})})$
2. Sample $r^{(a)}_{(i)} \leftarrow \{0,1\}^\lambda$ for $i \in [1, M], j \in [1, \ell]$.
3. (As garbler of GC) Compute $\mathsf{Garble}(1^\lambda, (\mathsf{UpdateTable}, \mathsf{UpdIndex})) \rightarrow (F, e, d)$, where F is the garbled circuit, and e encodes both possible bits of $|\mathbb{T}^{(\cdot)}|, |\mathsf{loc}^{(\cdot)}|, |R^{(1)}|, |R^{(2)}|, |\mathbb{L}^{(1)}|, |\mathbb{L}^{(2)}|$, $|r^{(a)}_{(i,j)}|$ for $i \in [1, M], j \in [1, \ell], a \in \{1,2\}$ and $|r^{(a)}_{(i)}|$ for $i \in [1, M], a \in \{1,2\}$
4. Send $(\mathtt{SEND}, (s_0, s_1))$ to $\mathcal{F}_{\mathsf{ot}}$, for each pair of encoded keys of bits in $|\mathbb{T}^{(\cdot)}|, |\mathsf{loc}^{(\cdot)}|, |R^{(\cdot)}|, |\mathbb{L}^{(\cdot)}|, |r_{(i,j)}|$ for $i \in [1, M], j \in [1, \ell]$, $|r_{(i)}|$ for $i \in [1, M]$
5. Send (F, d) to the other server , where F includes the keys for its own inputs, i.e. $r_{(i,j)}$ for $i \in [1, M], j \in [1, \ell]$, $\mathsf{loc}^{(\mathsf{a})}$, $R^{(a)}$.

Server_a, upon receiving (F, d) from the other server:

1. (As evaluator of GC) Upon receiving SEND from $\mathcal{F}_{\mathsf{ot}}$, send $(\mathtt{RECEIVE}, b)$ to $\mathcal{F}_{\mathsf{ot}}$ for each bit b in $\mathbb{T}^{(a)}, \mathsf{loc}^{(\mathsf{a})}, R^{(a)}, \mathbb{L}^{(a)}, r^{(a)}_{(i)}$ for $i \in [1, M], j \in [1, \ell]$ and denote these strings as X_a
2. Compute $\mathsf{Eval}(F, (X_a))$ to get Y
3. Compute $\mathsf{Decode}(d, Y)$ to get a new $\mathbb{T}^{(a)}$ and $\mathbb{L}^{(a)}$

Figure 11: GC protocol to update two tables

index.

When the recipient requests its row, it also sends a vector $r_0 \ldots r_\ell$ to the two servers. With these two vectors, the servers update their tables. In this way, the rows for the recipient are flushed so that new signals can be received. The request also includes a ctr_i and a signature on the vector and ctr_i. The servers check that the ctr_i match with their locally stored ctr_i. This check ensures that a malicious user cannot simply replay an old request and learn the signals. The protocol is presented formally in Figure 10.

Theorem 2. *The protocol $\Pi_{\mathsf{privSignal}}$ (Figure 10) UC-realizes the $\mathcal{F}_{\mathsf{privSignal}}$ functionality (Figure 7) in the $\mathcal{F}_{\mathsf{ot}}$-hybrid model assuming secure garbled circuits (Definition 1) and existential unforgeable signature schemes (Definition 4) .*

Proof. In our proofs we consider different cases one of which is when a corrupt sender may collude with a corrupt Server_1 and recipient and tries to learn the recipients and locations of honest signals. We present below the crucial strategies of the simulation:

- At all times each $\mathbb{T}^{(1)}$ and $\mathbb{T}^{(2)}$ only store shares of 0s.
- *Send*: Note that the simulator simulates Server_2 and will therefore receive $R^{(2)}$ and $\mathsf{loc}^{(2)}$ from the corrupt sender. Moreover the simulator also simulates the $\mathcal{F}_{\mathsf{ot}}$ functionality towards the corrupt Server_1. This way when Server_1 requests the labels of $\mathsf{loc}^{(1)}$ and $R^{(1)}$ via the $\mathcal{F}_{\mathsf{ot}}$ functionality, the simulator learns the exact bits of $\mathsf{loc}^{(1)}$ and $R^{(1)}$. The simulator then computes $R = R^{(1)} \oplus R^2$ and $\mathsf{loc} = \mathsf{loc}^{(1)} \oplus \mathsf{loc}^{(2)}$ and send $(\mathtt{SEND}, R, \mathsf{loc})$ to $\mathcal{F}_{\mathsf{privSignal}}$ on behalf of the corrupt sender.
- *Receive*: Note that for a corrupt R_i to request its row, it must request both Server_1 and Server_2. Therefore the simulator learns when a corrupt R_i makes a RECEIVE request. The simulator then

The UpdateTable function

Input: $\mathbb{T}^{(1)}$, $\mathbb{L}^{(1)}$, $\mathsf{loc}^{(1)}$, $R^{(1)}$, $R^{(2)}$, $\{r^1_{(i,j)}\}_{[i\in[1,M],j\in[1,\ell]]}$, $\{r^{(2)}_{(i,j)}\}_{[i\in[1,M],j\in[1,\ell]]}$, $\{r^{(1)}_{(i)}\}_{i\in[1,M]}$ and $\{r^{(2)}_{(i)}\}_{i\in[1,M]}$

Output: Updated $\mathbb{T}^{(1)}, \mathbb{L}^{(1)}$

Algorithm

1: Compute $R = R^{(1)} \oplus R^{(2)}$
2: Compute $\mathsf{index} = (\mathbb{L}^{(1)}[R] \oplus \mathbb{L}^{(2)}[R]) \mod \ell$
3: Update $\mathbb{T}^{(1)}[R][\mathsf{index}] = \mathsf{loc}^{(1)}$
4: Update $\mathbb{L}^{(1)}[R] = (\mathsf{index} + 1)$
5: **for** i in $[1, M]$ **do**
6: $\quad \mathbb{L}^{(1)}[i] = \mathbb{L}^{(1)}[i] \oplus r^{(1)}_{(i)} \oplus r^{(2)}_{(i)}$
7: $\quad$ **for** j in $[1, \ell]$ **do**
8: $\quad\quad \mathbb{T}^{(1)}[i][j] = \mathbb{T}^{(1)}[i][j] \oplus r^{(1)}_{(i,j)} \oplus r^{(2)}_{(i,j)}$
9: **return** $\mathbb{T}^{(1)}, \mathbb{L}^{(1)}$

Figure 12: The function to update the tables $\mathbb{T}^{(1)}$ and $\mathbb{L}^{(1)}$. The same algorithm updates the tables for Server_2, except in step 4: the circuit updates $\mathbb{L}^{(2)}[R] = 0$

sends the `RECEIVE` command to the $\mathcal{F}_{\mathsf{privSignal}}$ ideal functionality on behalf of the corrupt R_i and then learns the locations that R_i would receive. Note that the two servers maintained shares of zero in every index. The simulator now updates the Server_2 row by XORing those shares with the locations it received from the functionality and sends this updated row to the recipient. The recipient learns the correct locations after receiving the rest of the shares from Server_1.

For the full proofs we refer the reader to Appendix C. □

9 Implementation and Evaluation

We implemented both of our proposed protocols, i.e., our TEE-assisted solution from Section 7 as well as our garbled circuit-based construction in the two server setting, cf. Section 8.

9.1 Implementation

We implemented Protocol $\Pi_{\mathsf{privSignal1}}$ using an Intel SGX [20]. In the implementation we use RSA-OAEP[10] as the public key encryption algorithm, since we require no ambiguity in the decryption for the TEE. Moreover, the RSA-OAEP encryption can be modified to make it key-private as was noted by Bellare et al. [3]. Additionally, the encryption scheme used to encrypt the vectors $\vec{L}$ in the persistent storage is an authenticated encryption scheme with associated data (AEAD) instantiated with AES-256 in GCM mode [24].

We implemented our garbled circuit-based protocol, cf. Figure 11, in Rust. We applied the garbled circuit compiler of Ball et al. [1], that has an open-source implementation we used.[6] Our implementation applies AES as symmetric cipher, while as the hash function SHA-256 was used. We take advantage of the point-and-permute and the Free-XOR optimization of Kolesnikov and Schneider [14].

[6] https://github.com/GaloisInc/fancy-garbling

9.2 Evaluation

In the single-server setting, our TEE-based protocol yields a performant solution for the private signaling problem. For parameters $l = 20$ and $M = 100$ recipients, the `SEND` function was executed in 6.26 seconds, while for $l = 20, M = 1000$, it took 31.86 seconds to register a signal.

The implementation of our protocol in the two-server setting was evaluated on a AWS t3.medium instance. It had 4 GB RAM and 1 core Intel(R) Xeon(R) Platinum 8259CL CPU at 2.50GHz and it was running on the Amazon Linux 2 operating system.

l	10^3	10^4	10^5
XOR	32,580,496	325,800,496	3,258,000,496
AND	15,390,000	153,900,000	1,539,000,000
PROJ	1,020,000	10,200,000	102,000,000

Table 4: Garbled circuit size in terms of the number of AND, XOR and PROJ gates for $M = 30$. Note that the number of gates are symmetric in M and l.

First, we observe the size of the garbled-circuits used in our implementations, cf. Table 4. Even though the majority of the gates are XOR-gates, considerable amount of AND gates are necessary to compute privately our $\mathcal{F}_{\mathsf{privSignal}}$ functionality. The number of AND gates grows linearly in the number of messages and also in the number users. Hence, this limits our protocol to be applied in scenarios where there are modest numbers of users or messages.

l \ M	10^1	10^2	10^3	10^1	10^2	10^3
10^1	.75s	5.8s	57s	5.7MB	57MB	0.6GB
10^2	5.7s	58s	582s	57.5MB	0.6GB	5GB
10^3	58s	582s	97m	0.6GB	5GB	50GB

Table 5: Left: Running times for evaluating the garbled circuit implementing $\Pi_{\mathsf{privSignal2}}$ Right: Communication costs of the two servers by varying M and ℓ

Computation and communication costs of our implementation of the $\Pi_{\mathsf{privSignal2}}$ shows that the garbled-circuit-based approach can support modest numbers of users or messages in practice. However, private signaling in applications where both the number of users and messages are plentiful requires considerable bandwidth and computing power from the servers, cf. Table 5.

Our experiments indicate that the TEE based solution is much more efficient than the GC based solution. Note that in both protocols, the computation done is to update an $M \times \ell$ table. In Protocol $\Pi_{\mathsf{privSignal1}}$ the overall latency is the network time for the host to supply the SGX with a row, the time taken by the SGX to encrypt and decrypt each element in the row. Whereas for $\Pi_{\mathsf{privSignal2}}$ the overall latency is in the computation of the garbled circuits, the network time in communicating the garbled truth tables and oblivious transfer and finally the evaluation of the circuits. As can be seen from Table 4 the number of non-linear gates (AND gates) in $\Pi_{\mathsf{privSignal2}}$ is very high (approx. 15 million for $M = 30, \ell = 1000$). This is one of the main bottlenecks in the overall latency for the Protocol $\Pi_{\mathsf{privSignal2}}$. The other major bottleneck in our implementation is in the communication of large garbled truth tables between the two servers. As can be seen from Table 5 the size of the tables increases linearly with M and ℓ and is approximately 0.6GB for $\ell = 10$ and $M = 1000$. We note, however, that our implementation is not optimized. For instance, the garbled circuit evaluation could be parallelized due to the circuit's structure. This and other garbled circuit optimizations (such as half-gates etc.) can help decreasing the communication and computation costs of our protocols. We leave these optimizations for future work.

Unfortunately, we cannot directly compare our performance evaluation with that of [2]. Beck et al. solely measured the computation costs of their FMD protocol for recipients. We do not provide recipient running time measurements, since in our protocols recipients only do negligible computation. However, Beck et al. unlike us, do not provide performance measurements for the server performing fuzzy message detection. Even though we provide stronger notions of privacy for recipients, we expect that the server's communication and computation costs in our protocols are competitive in comparison with the fuzzy message detection protocol.

Optimizing by processing a batch of signals We also improve the circuit evaluation for Protocol $\Pi_{\mathsf{privSignal2}}$ by processing K signals at a time. We observed that for $K = 5$, the overall computation took $2\times$ less time than linearly processing 5 signals sequentially. And for $K = 25$, the overall computation improved by $2.7\times$. This gain in overall processing time can be attributed to lesser number of garbled tables that need to be computed and communicated.

10 Conclusion and Open Problems

We have introduced the problem of *private signaling* that abstracts and generalizes several real-world recipient-anonymous applications. We have provided a formal definition in the UC-framework, two server-aided protocols that achieve this definition (in the semi-honest and malicious setting), and open source implementations. Our protocols achieves the best efficiency for the sender and recipients, requiring only minimal overhead.

The workload of the servers however is proportional to $O(M\ell)$ *per signal*, which limits the choice of the parameters of M and ℓ. We leave it as future work to explore techniques such as ORAM to improve the workload of the servers.

Availability

We release the source code of our implementations of private signaling protocols to facilitate further improvements and experiments. It can be found at https://github.com/ZenGo-X/pps-gc.

References

[1] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for boolean and arithmetic circuits. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 565–577, 2016.

[2] Gabrielle Beck, Julia Len, Ian Miers, and Matthew Green. Fuzzy message detection. *IACR eprint (Accepted at CCS 2021)*, 2021.

[3] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 566–582. Springer, 2001.

[4] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796, 2012.

[5] 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)*, 2017.

[6] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[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, 2015.

[8] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

[9] Nicolas T Courtois and Rebekah Mercer. Stealth address and key management techniques in blockchain systems. *ICISSP*, 2017:559–566, 2017.

[10] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. Rsa-oaep is secure under the rsa assumption. In *Annual International Cryptology Conference*, pages 260–274. Springer, 2001.

[11] Arthur Gervais, Srdjan Capkun, Ghassan O Karame, and Damian Gruber. On the privacy provisions of bloom filters in lightweight bitcoin clients. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 326–335, 2014.

[12] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache attacks on intel sgx. In *Proceedings of the 10th European Workshop on Systems Security*, pages 1–6, 2017.

[13] Mike Hearn and Matt Corallo. Bip 37: Connection bloom filtering. *URL https://github.com/bitcoin/bips/blob/master/bip-0037. mediawiki*, 2012.

[14] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 486–498. Springer, 2008.

[15] Duc V Le, Lizzy Tengana Hurtado, Adil Ahmad, Mohsen Minaei, Byoungyoung Lee, and Aniket Kate. A tale of two trees: one writes, and other reads: Optimized oblivious accesses to bitcoin and other utxo-based blockchains. *Proceedings on Privacy Enhancing Technologies*, 2020(2), 2020.

[16] Jaehyuk Lee, Jinsoo Jang, Yeongjin Jang, Nohyun Kwak, Yeseul Choi, Changho Choi, Taesoo Kim, Marcus Peinado, and Brent ByungHoon Kang. Hacking in darkness: Return-oriented programming against secure enclaves. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 523–539, 2017.

[17] Sarah Lewis. Discreet log #1: Anonymity, bandwidth and fuzzytags.

[18] Ian Martiny, Gabriel Kaptchuk, Adam Aviv, Dan Roche, and Eric Wustrow. Improving signal's sealed sender. 2021.

[19] Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiainen, Ghassan Karame, and Srdjan Capkun. {BITE}: Bitcoin lightweight client privacy using trusted execution. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 783–800, 2019.

[20] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. *Hasp@ isca*, 10(1), 2013.

[21] Rafael Pass, Elaine Shi, and Florian Tramer. Formal abstractions for attested execution secure processors. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 260–289. Springer, 2017.

[22] Kaihua Qin, Henryk Hadass, Arthur Gervais, and Joel Reardon. Applying private information retrieval to lightweight bitcoin clients. In *2019 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 60–72. IEEE, 2019.

[23] Justus Ranvier. Reusable payment codes for hierarchical deterministic wallets.

[24] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 98–107, 2002.

[25] Antoine Rondelet and Michal Zajac. Zeth: On integrating zerocash on ethereum. *arXiv preprint arXiv:1904.00905*, 2019.

[26] 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.

[27] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[28] Karl Wüst, Sinisa Matetic, Moritz Schneider, Ian Miers, Kari Kostiainen, and Srdjan Čapkun. Zlite: Lightweight clients for shielded zcash transactions using trusted execution. In *International Conference on Financial Cryptography and Data Security*, pages 179–198. Springer, 2019.

[29] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*, pages 640–656. IEEE, 2015.

[30] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.

A Preliminaries (contd.)

A.1 Indistinguishability under chosen plaintext attacks

We present a definition for CPA security for private key encryption in Def 2 and for public key encryption in Def 3.

Definition 2. *Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme. Let $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$. Let $\mathcal{A}$ be an adversary and consider the following experiment.*

Experiment $\boldsymbol{Exp}^{\mathsf{priv-cpa-b}}_{\mathcal{E},\mathcal{A}}(\lambda)$*:*

$k \leftarrow \mathsf{KeyGen}(\lambda)$
$m_0, m_1 \leftarrow \mathcal{A}^{\mathcal{O}_{Enc_k(\cdot)}}(\lambda)$
$c \leftarrow \mathsf{Enc}(k, m_b)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{Enc_k(\cdot)}}(\lambda)$
return b'

The advantage of the adversary is given as

$$\mathbf{Adv}^{\mathsf{priv-cpa}}_{\mathcal{E},\mathcal{A}} = Pr[\boldsymbol{Exp}^{\mathsf{priv-cpa-1}}_{\mathcal{E},\mathcal{A}}(\lambda) = 1] - Pr[\boldsymbol{Exp}^{\mathsf{priv-cpa-0}}_{\mathcal{E},\mathcal{A}}(\lambda) = 1]$$

The scheme $\mathcal{E}$ is said to be CPA secure if the function $\mathbf{Adv}^{\mathsf{priv-cpa}}_{\mathcal{E},\mathcal{A}}(\cdot)$ is negligble for any adversay $\mathcal{A}$ whose time complexity is polynomial in λ.

Lemma 1. *Any private-key encryption scheme that has indistinguishable encryptions under a chosen-plaintext attack also has indistinguishable multiple encryptions under a chosen-plaintext attack.*

Definition 3. *Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme. Let $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$. Let $\mathcal{A}$ be an adversary and consider the following experiment.*

Experiment $\boldsymbol{Exp}^{\mathsf{pub-cpa-b}}_{\mathcal{E},\mathcal{A}}(\lambda)$*:*

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\lambda)$
$m_0, m_1 \leftarrow \mathcal{A}^{\mathcal{O}_{Enc_{\mathsf{pk}}(\cdot)}}(\lambda)$
$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{Enc_{\mathsf{pk}}(\cdot)}}(\lambda)$
return b'

The advantage of the adversary is given as

$$\mathbf{Adv}^{\mathsf{pub-cpa}}_{\mathcal{E},\mathcal{A}} = Pr[\boldsymbol{Exp}^{\mathsf{pub-cpa-1}}_{\mathcal{E},\mathcal{A}}(\lambda) = 1] - Pr[\boldsymbol{Exp}^{\mathsf{pub-cpa-0}}_{\mathcal{E},\mathcal{A}}(\lambda) = 1]$$

The scheme $\mathcal{E}$ is said to be CPA secure if the function $\mathbf{Adv}^{\mathsf{pub-cpa}}_{\mathcal{E},\mathcal{A}}(\cdot)$ is negligble for any adversay $\mathcal{A}$ whose time complexity is polynomial in λ.

A.2 Existential Unforgeability under Chosen Message Attacks

Definition 4. *A digital signature scheme consists of three algorithms $(\Sigma.\mathsf{KeyGen}.\Sigma.\mathsf{Sig}, \Sigma.\mathsf{Ver})$. Let $\mathcal{A}$ be the adversary and consider the following experiment: Experiment* $\boldsymbol{Exp}^{\mathsf{euf-cma}}_{\Sigma,\mathcal{A}}(\lambda)$*:*

$(\Sigma.\mathsf{vk}, \Sigma.\mathsf{sk}) \leftarrow \Sigma.\mathsf{KeyGen}(\lambda)$
$(m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sig}_\Sigma.\mathsf{sk}(\cdot)}}(\lambda)$
Let $\mathcal{Q}$ be the set of oracle queries to $\mathsf{Sig}_\Sigma.\mathsf{sk}(\cdot)$.
If $m \notin \mathcal{Q}$ and $\Sigma.\mathsf{Ver}(\Sigma.\mathsf{vk}, m, \sigma) = 1$, return 1.

The advantage of the adversary is given as

$$\mathbf{Adv}^{\mathsf{euf-cma}}_{\Sigma,\mathcal{A}} = Pr[\boldsymbol{Exp}^{\mathsf{euf-cma}}_{\Sigma,\mathcal{A}}(\lambda) = 1]$$

A.3 Key privacy under chosen plaintext attacks

We present a notion of key-privacy under chosen plaintext attacks as defined in [3]. The adversary runs in two stages: in the find stage it takes two public keys pk_0 and pk_1 and outputs a message x with some state information s. In the guess stage the adversary gets a challenge ciphertext y, which is encrypted under one of the two keys at random. The adversary then tries to guess which key was used to compute the ciphertext y.

Definition 5. *Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme. Let $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$. Let $\mathcal{A}$ be an adversary. Consider the following experiment:*

Experiment $\boldsymbol{Exp}^{\mathsf{ik-cpa-b}}_{\mathcal{E},\mathcal{A}}(\lambda)$*:*

$(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KeyGen}(1^k)$ *and* $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
$(x, s) \leftarrow \mathcal{A}(\mathsf{find}, \mathsf{pk}_0, \mathsf{pk}_1)$
$y \leftarrow \mathsf{Enc}(\mathsf{pk}_b, x)$
$d \leftarrow \mathcal{A}(\mathsf{guess}, y, s)$
return d

The advantage of the adversary is given as

$$\mathbf{Adv}^{\mathsf{ik-cpa}}_{\mathcal{E},\mathcal{A}} = Pr[\boldsymbol{Exp}^{\mathsf{ik-cpa-1}}_{\mathcal{E},\mathcal{A}}(\lambda) = 1] - Pr[\boldsymbol{Exp}^{\mathsf{ik-cpa-0}}_{\mathcal{E},\mathcal{A}}(\lambda) = 1]$$

.

The scheme $\mathcal{E}$ is said to be $IK - CPA$ secure, if the function $\mathbf{Adv}^{\mathsf{ik-cpa}}_{\mathcal{E},\mathcal{A}}(\cdot)$ is negligible for any adversary $\mathcal{A}$ whose time complexity is polynomial in k.

A.4 Universal Composability

In UC security we consider the execution of the protocol in a special setting involving an environment machine $\mathcal{Z}$, in addition to the honest parties and adversary. In UC, ideal and real models are considered where a trusted party carries out the computation in the ideal model while the actual protocol runs in the real model. The trusted party is also called the ideal functionality. For example the ideal functionality $\mathcal{F}_{\mathsf{privSignal}}$ is a trusted party that provides the functionality of private signaling. In the UC setting, there is a global environment (the distinguisher) that chooses the inputs for the honest parties, and interacts with an adversary who is the party that participates in the protocol on behalf of dishonest parties. At the end of the protocol execution, the environment receives the output of the honest parties as well as the output of the adversary which one can assume to contain the entire transcript of the protocol. When the environment activates the honest parties and the adversary, it does not know whether the parties and the adversary are running the real protocol –they are in the real world, or they are simply interacting with the trusted ideal functionality, in which case the adversary is not interacting with any honest party, but is simply "simulating" to engage in the protocol. In the ideal world the adversary is therefore called simulator, that we denote by $\mathcal{S}$.

In the UC-setting, we say that a protocol securely realizes an ideal functionality, if there exist no environment that can distinguish whether the output he received comes from a real execution of the protocol between the honest parties and a real adversary $\mathcal{A}$, or from a simulated execution of the protocol produced by the simulator, where the honest parties only forward date to and from the ideal functionality.

The transcript of the ideal world execution is denoted $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda, z)$ and the transcript of the real world execution is denoted ${}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)$. A protocol is secure if the ideal world transcript and the real world transcripts are indistinguishable. That is, $\{\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda, z\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \equiv \{{}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$.

B Proof of Theorem 1

B.1 Correctness

The correctness of this scheme is pretty straightforward conditioned on the fact that there are no two secret keys for which the decryption of $\mathsf{ct}_{\mathsf{Signal}}$ gives the first λ bits as 0^{λ}. Since we use encryption scheme (RSA-OAEP) that gives us this in the implementation

B.2 Protocol $\Pi_{\mathsf{privSignal1}}$ realizes the $\mathcal{F}_{\mathsf{privSignal}}$ functionality

Proof. To prove that $\Pi_{\mathsf{privSignal1}}$ UC-realizes $\mathcal{F}_{\mathsf{privSignal}}$ we show that there exists a simulator $\mathcal{S}$ that interacts with $\mathcal{F}_{\mathsf{privSignal}}$ and the adversary $\mathcal{A}$ to generate a transcript that is indistinguishable from the real world protocol. We consider the following cases of corruption:

- Simulator $\mathcal{S}_N$ for the case when only the server is corrupt.
- Simulator $\mathcal{S}_s$ for the case when a subset of the senders and the server are corrupt.
- Simulator $\mathcal{S}_r$ for the case when a subset of the recipients and the server are corrupt/
- Simulator $\mathcal{S}_{rs}$ for the case when a subset of the recipients, a subset of the senders and the server is corrupt.

We discuss these simulators in more detail in the next subsections. □

B.3 Case 1: Neither S nor R is corrupt

Simulator overview When neither the sender nor the recipients are corrupt, then the only corrupt entity is the server. In this case, the simulator interacts with the Server and the $\mathcal{F}_{\mathsf{privSignal}}$ to simulate a transacript that is indistinguishable from the real world. Note that the simulator also simulates $\mathcal{G}_{\text{att}}$ towards $\mathcal{A}$.

Proof by hybrids We prove security via a sequence of hybrids where we start from the real world and move to the ideal world.

- $\mathbf{Hyb}_0$ The real world protocol.
- $\mathbf{Hyb}_1$ is the same as $\mathbf{Hyb}_0$ except that upon receiving a `SEND` command, the $\mathsf{ct}_{\mathsf{Signal}}$ is replaced with an encryption to 0 instead of the actual location. By the CPA security (Def 3) of the underlying encryption scheme we prove in Lemma 2 that the two hybrids are indistinguishable.
- $\mathbf{Hyb}_2$ is the same as $\mathbf{Hyb}_1$ except that for each `SEND` command, the loc is now encrypted under a fresh public key as in the simulation. By the key privacy (Def 5) of the underlying encryption scheme we prove in Lemma 3 that the two hybrids are indistinguishable.
- $\mathbf{Hyb}_3$ is the same as $\mathbf{Hyb}_2$ except that in the `SEND` command, the $\mathcal{G}_{\text{att}}$ functionality returns encryptions of 0 under SGXkey_j instead of actual locations. By the CPA security (Def 2) of the encryption scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_4$ is the same as $\mathbf{Hyb}_3$ except that in the `RECEIVE` command, the simulator returns encryptions of 0 as $\mathsf{ct}_{\mathsf{loc},i}$ to the server on behalf of $\mathcal{G}_{\text{att}}$. By the CPA security (Def 3) of the underlying encryption scheme, we prove in Lemma 5 that the two hybrids are indistinguishable.
- $\mathbf{Hyb}_5$ is the same as $\mathbf{Hyb}_4$ except that in the **Setup** procedure, the simulator aborts with $\mathsf{sigFailure}_1$. We prove in Lemma 6 that this occurs with negligible probability.

The simulator $\mathcal{S}$ maintains a public **board** and internally simulates $\mathcal{G}_{\text{att}}$ towards the adversary $\mathcal{A}$

Setup For each recipient R_i:

1. Sample random a and send $(\text{“keyex”}, g^a)$ to $\mathcal{A}$
2. Upon receiving $(eid_i, \mathsf{Prog}, (g^a, g^b)), \sigma)$ from $\mathcal{A}$, abort with $\mathsf{sigFailure}_1$ if σ would be validated by a honest R_i, yet the following $\mathcal{A} \Leftrightarrow \mathcal{G}_{\text{att}}$ communication was not recorded:
   - $eid := \mathcal{G}_{\text{att}}.\mathsf{install}(\mathsf{Prog})$
   - $((g^a, g^b), \sigma) := \mathcal{G}_{\text{att}}.\mathsf{resume}(eid_i, (\text{“keyex”}, g^a))$

   Else compute $k_i = g^r$, where $r \leftarrow_\$ \mathbb{Z}_q$ and $\mathsf{SGXkey}_i \leftarrow \mathsf{PrivEnc.KeyGen}(1^\lambda)$
3. Compute $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Enc.KeyGen}(1^\lambda)$, $(\Sigma.\mathsf{sk}_i, \Sigma.\mathsf{vk}_i) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$. Set $\mathsf{ct}_{\mathsf{keys},i} = \mathsf{Enc}(k_i, (\mathsf{sk}_i, \Sigma.\mathsf{vk}_i))$ and send $(\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i})$ to Server. Upon receiving $((eid_i, \mathsf{pk}_i, \vec{L}_i), \sigma)$ abort with $\mathsf{sigFailure}_2$ if σ would be validated but the following communication was not recorded: $((\mathsf{pk}_i, \vec{L}_i), \sigma) = \mathcal{G}_{\text{att}}.\mathsf{resume}(eid_i, (\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i})$, where $\vec{L}_i = \{\mathsf{Enc}(\mathsf{SGXkey}_i, 0)\}_{j=0}^{\ell}$. Else publish pk_i and set $\mathsf{index}_i = 0, \mathsf{ctr}_i = 0$.

`WRITE` : Upon receiving $(\texttt{WRITE}, (S_i, m))$ from $\mathcal{F}_{\mathsf{privSignal}}$, update $\mathbf{board} = \mathbf{board} \| m$ and send `ok` to $\mathcal{F}_{\mathsf{privSignal}}$.

`SEND`: Upon receiving $(\texttt{SEND}, S_i)$ from $\mathcal{F}_{\mathsf{privSignal}}$

1. Create keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Enc.KeyGen}(1^\lambda)$ and compute $\mathsf{ct}_{\mathsf{Signal}} = \mathsf{Enc}(\mathsf{pk}, 0)$ and send $(\texttt{SEND}, \mathsf{ct}_{\mathsf{Signal}})$ to $\mathcal{A}$.
2. Upon receiving $(\text{“send”}, \vec{L}_j, \mathsf{ct}_{\mathsf{Signal}})$ on behalf of $\mathcal{G}_{\text{att}}$ from $\mathcal{A}$, update $\vec{L}_j = \{\mathsf{Enc}(\mathsf{SGXkey}_j, 0)\}_0^{\ell}$ for $s \in [1, M]$ and return the updated $\vec{L}_j$ to $\mathcal{A}$.

`RECEIVE` : Upon receiving $(\texttt{RECEIVE}, R_j)$ from $\mathcal{F}_{\mathsf{privSignal}}$

1. Compute $\sigma_i = \mathsf{Sig}(\Sigma.\mathsf{sk}_i, \mathsf{ctr}_i)$ and send $(\texttt{RECEIVE}, \mathsf{ctr}_i, \sigma_i)$ to $\mathcal{A}$.
2. Upon receiving $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid_i, (\text{“receive”}, \mathsf{ctr}_i, \sigma_i, \vec{L}_i))$ on behalf of $\mathcal{G}_{\text{att}}$:
   (a) Compute $\vec{\mathsf{ct}}_{\mathsf{loc},i} = \mathsf{Enc}(\mathsf{pk}_i, 0) \ldots \mathsf{Enc}(\mathsf{pk}_i, 0)$
   (b) Update $\vec{L} = \{\mathsf{Enc}(\mathsf{SGXkey}, 0)\}_{j=0}^{\ell}$
   (c) Update $\mathsf{ctr}_i = \mathsf{ctr}_i + 1$
   (d) Compute $\sigma_T = \mathsf{Sig}(\mathsf{msk}, (eid_i, \vec{L}_i, \vec{\mathsf{ct}}_{\mathsf{loc},i}))$
   (e) Return $(\vec{L}_i, \vec{\mathsf{ct}}_{\mathsf{loc},i}, \sigma_T)$ to $\mathcal{A}$
3. Receive $(eid_i, \vec{L}_i, \vec{\mathsf{ct}}_{\mathsf{loc},i}, \sigma_T)$ from $\mathcal{A}$. If this was received without the communication with $\mathcal{G}_{\text{att}}$, abort with $\mathsf{sigFailure}_3$.

Figure 13: Simulator $\mathcal{S}_N$ for the case of only one corrupt server

- $\mathbf{Hyb}_6$ is the same as $\mathbf{Hyb}_5$ except that in the **Setup** procedure, the simulator aborts with $\mathsf{sigFailure}_2$. We prove in Lemma 7 that this occurs with negligible probability.
- $\mathbf{Hyb}_7$ is the same as $\mathbf{Hyb}_6$ except that in the `RECEIVE` command, the simulator may abort with $\mathsf{sigFailure}_3$. We prove in Lemma 8 that this occurs with negligible probability.
- $\mathbf{Hyb}_8$ is the same as $\mathbf{Hyb}_7$ except that in the **Setup** procedure, the key k_i is computed as $k_i \leftarrow g^c$ where $c \leftarrow_\$ \mathbb{Z}_q$. We prove in Lemma 9 that this occurs with negligible probability.

Lemma 2. *Assuming CPA secure encryption scheme (Def 3),* $\mathbf{Hyb}_1$ *and* $\mathbf{Hyb}_0$ *are indistinguishable against a PPT adversary.*

Proof. Note that the difference between $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ is that in $\mathbf{Hyb}_1$ the encryption $\mathsf{ct}_{\mathsf{Signal}}$ is replaced by an encryption to 0, under the same pk.

Assume a distinguisher D can distinguish between $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$, i,e. $Pr[D(\mathbf{Hyb}_1) = 1] - Pr[D(\mathbf{Hyb}_0) = 1] > \mathsf{negl}(\lambda)$

Using this distinguisher D we construct a reduction B that can break the CPA security of encryption scheme.

1. Activate the distinguisher D
2. Run the **Setup** procedure for an honest recipient R, and let the encryption public key be pk.
3. Upon receiving a $(\texttt{SEND}, R, \mathsf{loc})$ command from the environment, set $m_0 = \mathsf{loc}$ and $m_1 = 0^\lambda$.
4. Send m_0, m_1 to the challenger and receive back c.
5. Set $\mathsf{ct}_{\mathsf{Signal}} = c$ and send the transcript to the distinguisher D
6. Output whatever D outputs.

Note that in the case $\mathsf{ct}_{\mathsf{Signal}}$ was the encryption of m_0 the distinguisher sees the hybrid world - $\mathbf{Hyb}_0$ and on the other hand when encryption of m_1 is returned the distinguisher sees the hybrid world $\mathbf{Hyb}_1$.

Thus
$Pr[D(\mathbf{Hyb}_1) = 1] = Pr[\mathbf{Exp}^{\mathsf{pub-cpa}-1}_{\mathcal{E},\mathcal{A}}(\lambda) = 1$
and
$Pr[D(\mathbf{Hyb}_0) = 1] = Pr[\mathbf{Exp}^{\mathsf{pub-cpa}-0}_{\mathcal{E},\mathcal{A}}(\lambda) = 1$
This implies

$$\mathbf{Adv}^{\mathsf{pub-cpa}}_{\mathcal{E},\mathcal{A}} = Pr[\mathbf{Exp}^{\mathsf{pub-cpa}-1}_{\mathcal{E},\mathcal{A}}(\lambda) = 1] - Pr[\mathbf{Exp}^{\mathsf{pub-cpa}-0}_{\mathcal{E},\mathcal{A}}(\lambda) = 1]$$

$= Pr[D(\mathbf{Hyb}_1) = 1] - Pr[D(\mathbf{Hyb}_0) = 1] > \mathsf{negl}(\lambda)$

But this is a contradiction, since we assume CPA secure encryption schemes and therefore $\mathbf{Adv}^{\mathsf{pub-cpa}}_{\mathcal{E},\mathcal{A}} < \mathsf{negl}(\lambda)$.

Hence $Pr[D(\mathbf{Hyb}_1) = 1] - Pr[D(\mathbf{Hyb}_0) = 1] < \mathsf{negl}(\lambda)$ making the two hybrids indistinguishable. □

Lemma 3. *Assuming key-privacy (Def 5) of the underlying PKE scheme,* $\mathbf{Hyb}_2$ *and* $\mathbf{Hyb}_1$ *are indistinguishable to a PPT adversary.*

Proof. Note that the difference between $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ is that in $\mathbf{Hyb}_2$ the encryption $\mathsf{ct}_{\mathsf{Signal}}$ is done using a freshly generated pk.

Assume a distinguisher D can distinguish between $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$, i,e. $Pr[D(\mathbf{Hyb}_2) = 1] - Pr[D(\mathbf{Hyb}_1) = 1] > \mathsf{negl}(\lambda)$

Using this distinguisher D we construct a reduction B that can break the IK-CPA security (Def 5) of encryption scheme.

1. Activate the distinguisher D and receive $\mathsf{pk}_0, \mathsf{pk}_1$ from the challenger.
2. Run the **Setup** procedure for an honest recipient R, and let the encryption public key be pk_0.
3. Upon receiving a $(\texttt{SEND}, R, \mathsf{loc})$ command from the environment, send $x = \mathsf{loc}$ and $s = \perp$ to the challenger.
4. Receive y. Set $\mathsf{ct}_{\mathsf{Signal}} = y$ and send the transcript to the distinguisher D
5. Output whatever D outputs.

Note that in the case $\mathsf{ct}_{\mathsf{Signal}}$ was encrypted under pk_0 the distinguisher sees the hybrid world - $\mathbf{Hyb}_1$ and on the other hand when encryption is done using pk_1 is returned the distinguisher sees the hybrid world $\mathbf{Hyb}_2$.

Thus
$Pr[D(\mathbf{Hyb}_2) = 1] = Pr[\mathbf{Exp}^{\mathsf{ik-cpa-1}}_{\mathcal{E},\mathcal{A}}(\lambda) = 1$
and
$Pr[D(\mathbf{Hyb}_1) = 1] = Pr[\mathbf{Exp}^{\mathsf{ik-cpa-0}}_{\mathcal{E},\mathcal{A}}(\lambda) = 1$
This implies

$$\mathbf{Adv}^{\mathsf{ik-cpa}}_{\mathcal{E},\mathcal{A}} = Pr[\mathbf{Exp}^{\mathsf{ik-cpa-1}}_{\mathcal{E},\mathcal{A}}(\lambda) = 1] - Pr[\mathbf{Exp}^{\mathsf{ik-cpa-0}}_{\mathcal{E},\mathcal{A}}(\lambda) = 1]$$

$= Pr[D(\mathbf{Hyb}_2) = 1] - Pr[D(\mathbf{Hyb}_1) = 1] > \mathsf{negl}(\lambda)$
But this is a contradiction, since we assume CPA secure encryption schemes and therefore $\mathbf{Adv}^{\mathsf{ik-cpa}}_{\mathcal{E},\mathcal{A}} < \mathsf{negl}(\lambda)$.
Hence $Pr[D(\mathbf{Hyb}_2) = 1] - Pr[D(\mathbf{Hyb}_1) = 1] < \mathsf{negl}(\lambda)$ making the two hybrids indistinguishable. □

Lemma 4. *Assuming the CPA security (Def 2) of the private key encryption scheme used by $\mathcal{G}_{att}$, $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ are indistinguishable.*

Proof. Note that the difference between $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ is that in $\mathbf{Hyb}_3$ the encryption of the elements in $\vec{L}$ is replaced by an encryptions to 0.

Assume a distinguisher D can distinguish between $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$, i,e. $Pr[D(\mathbf{Hyb}_3) = 1] - Pr[D(\mathbf{Hyb}_2) = 1] > \mathsf{negl}(\lambda)$

Using this distinguisher D we construct a reduction B that can break the CPA security of encryption scheme.

1. Activate the distinguisher D
2. Run the **Setup** procedure for an honest recipient R, and let the key for enccrypting $\vec{L}$ be SGXkey.
3. Upon receiving a $(\texttt{SEND}, R, \mathsf{loc})$ command from the environment, set $\vec{m_0} = \mathsf{loc}_1 \ldots \mathsf{loc}_\ell$ and $\vec{m_1} = 0^\lambda, \ldots 0^\lambda$.
4. Send $\vec{m_0}, \vec{m_1}$ to the challenger and receive back c.
5. Set $\vec{L} = c$ and send the transcript to the distinguisher D
6. Output whatever D outputs.

Note that in the case $\vec{L}$ was the encryption of $\vec{m_0}$ the distinguisher sees the hybrid world - $\mathbf{Hyb}_2$ and on the other hand when encryption of $\vec{m_1}$ is returned the distinguisher sees the hybrid world $\mathbf{Hyb}_3$.

Thus by lemma 1,
$Pr[D(\mathbf{Hyb}_3) = 1] = Pr[\mathbf{Exp}^{\mathsf{priv-cpa-1}}_{\mathcal{E},\mathcal{A}}(\lambda) = 1$

and
$Pr[D(\mathbf{Hyb}_2) = 1] = Pr[\mathbf{Exp}^{\mathsf{priv-cpa}-0}_{\mathcal{E},\mathcal{A}}(\lambda) = 1$
This implies

$$\mathbf{Adv}^{\mathsf{priv-cpa}}_{\mathcal{E},\mathcal{A}} = Pr[\mathbf{Exp}^{\mathsf{priv-cpa}-1}_{\mathcal{E},\mathcal{A}}(\lambda) = 1] - Pr[\mathbf{Exp}^{\mathsf{priv-cpa}-0}_{\mathcal{E},\mathcal{A}}(\lambda) = 1]$$

$= Pr[D(\mathbf{Hyb}_3) = 1] - Pr[D(\mathbf{Hyb}_2) = 1] > \mathsf{negl}(\lambda)$
But this is a contradiction, since we assume CPA secure encryption schemes and therefore $\mathbf{Adv}^{\mathsf{priv-cpa}}_{\mathcal{E},\mathcal{A}} < \mathsf{negl}(\lambda)$.
Hence $Pr[D(\mathbf{Hyb}_3) = 1] - Pr[D(\mathbf{Hyb}_2) = 1] < \mathsf{negl}(\lambda)$ making the two hybrids indistinguishable. □

Lemma 5. *Assuming CPA secure encryption scheme (Def 3) and by Lemma 1,* $\mathbf{Hyb}_4$ *and* $\mathbf{Hyb}_3$ *are indistinguishable against a PPT adversary.*

Proof. Note that the difference between $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$ is that in $\mathbf{Hyb}_4$ the encryption $\mathsf{ct}_{\mathsf{loc}}$ is replaced by encryptions to 0, under the same pk.
Assume a distinguisher D can distinguish between $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_3$, i,e. $Pr[D(\mathbf{Hyb}_4) = 1] - Pr[D(\mathbf{Hyb}_3) = 1] > \mathsf{negl}(\lambda)$
Using this distinguisher D we construct a reduction B that can break the CPA security of encryption scheme.

1. Activate the distinguisher D
2. Run the **Setup** procedure for an honest recipient R, and let the encryption public key be pk.
3. Upon receiving a `RECEIVE` command, simulate the protocol such that $\vec{\mathsf{locns}} = \{\mathsf{loc}_1 \dots \mathsf{loc}_\ell\}$ and set $\vec{m_0} = \vec{\mathsf{locns}}$ and $\vec{m_1} = 0^\lambda$.
4. Send $\vec{m_0}, \vec{m_1}$ to the challenger and receive back $\vec{c}$.
5. Set $\vec{\mathsf{ct}}_{\mathsf{loc}} = \vec{c}$ and send the transcript to the distinguisher D
6. Output whatever D outputs.

Note that in the case $\vec{\mathsf{ct}}_{\mathsf{loc}}$ was the encryption of $\vec{m_0}$ the distinguisher sees the hybrid world - $\mathbf{Hyb}_3$ and on the other hand when encryption of $\vec{m_1}$ is returned the distinguisher sees the hybrid world $\mathbf{Hyb}_4$.
Thus
$Pr[D(\mathbf{Hyb}_4) = 1] = Pr[\mathbf{Exp}^{\mathsf{pub-cpa}-1}_{\mathcal{E},\mathcal{A}}(\lambda) = 1$
and
$Pr[D(\mathbf{Hyb}_3) = 1] = Pr[\mathbf{Exp}^{\mathsf{pub-cpa}-0}_{\mathcal{E},\mathcal{A}}(\lambda) = 1$
This implies

$$\mathbf{Adv}^{\mathsf{pub-cpa}}_{\mathcal{E},\mathcal{A}} = Pr[\mathbf{Exp}^{\mathsf{pub-cpa}-1}_{\mathcal{E},\mathcal{A}}(\lambda) = 1] - Pr[\mathbf{Exp}^{\mathsf{pub-cpa}-0}_{\mathcal{E},\mathcal{A}}(\lambda) = 1]$$

$= Pr[D(\mathbf{Hyb}_4) = 1] - Pr[D(\mathbf{Hyb}_3) = 1] > \mathsf{negl}(\lambda)$
But this is a contradiction, since we assume CPA secure encryption schemes and therefore $\mathbf{Adv}^{\mathsf{pub-cpa}}_{\mathcal{E},\mathcal{A}} < \mathsf{negl}(\lambda)$.
Hence $Pr[D(\mathbf{Hyb}_4) = 1] - Pr[D(\mathbf{Hyb}_3) = 1] < \mathsf{negl}(\lambda)$ making the two hybrids indistinguishable. □

Lemma 6. *Assuming existential unforgeable signatures that are secure against chosen message attacks,* $\mathbf{Hyb}_5$ *and* $\mathbf{Hyb}_4$ *are indistinguishable.*

Proof. Note that the difference between $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$ is that in $\mathbf{Hyb}_5$ the event $\mathsf{sigFailure}_1$ can occur. We prove in this section that the probability of this event occurring is negligible.

First we observe that $\mathsf{sigFailure}_1$ occurs when the simulator receives a signature from the adversary that was not created by the simulator on behalf of the $\mathcal{G}_{\text{att}}$.

Assume a distinguisher D can distinguish between $\mathbf{Hyb}_5$ and $\mathbf{Hyb}_4$, i,e. $Pr[D(\mathbf{Hyb}_5) = 1] - Pr[D(\mathbf{Hyb}_4) = 1] > \mathsf{negl}(\lambda)$

This implies that $Pr[\mathsf{sigFailure}_1] > \mathsf{negl}(\lambda)$.

Which implies that $Pr[\mathcal{A}(\cdot) = ((g^a, g^b), \sigma) \wedge \Sigma.\mathsf{Ver}(\mathsf{mpk}, (eid, g^a, g^b), \sigma) = 1] > \mathsf{negl}(\lambda)$

Using this adversary we present a reduction $\mathcal{B}$ that breaks the EUF-CMA property (Def 4) of signature schemes.

1. Simulate the world as in $\mathbf{Hyb}_4$, and receive $\Sigma.\mathsf{vk}$ from the challenger. Set mpk of $\mathcal{G}_{\text{att}}$ as $\Sigma.\mathsf{vk}$.
2. UWhen simulating $\mathcal{G}_{\text{att}}$ and (g^a, g^b, eid) needs to be signed, use $\mathcal{O}_{\mathsf{Sig}_{\Sigma.\mathsf{sk}}}(\cdot)$ and send back the signature to the adversary.
3. Upon receiving $((eid, (g^a, g^b)), \sigma'))$ from $\mathcal{A}$ for which there was no communication with $\mathcal{G}_{\text{att}}$, check that
   $\Sigma.\mathsf{Ver}(\mathsf{mpk}, (eid, (g^a, g^b)), \sigma') = 1$.
4. If yes, output $m = (eid, (g^a, g^b))$ and $\sigma = \sigma'$

Observe that

$$\mathbf{Adv}^{\mathsf{euf-cma}}_{\Sigma,\mathcal{A}} = Pr[\mathbf{Exp}^{\mathsf{euf-cma}}_{\Sigma,\mathcal{A}}(\lambda) = 1]$$
$$= Pr[\Sigma.\mathsf{Ver}(\mathsf{mpk}, (eid, g^a, g^b), \sigma) = 1] > \mathsf{negl}(\lambda)$$

But this is a contradiction since we assume EUF-CMA signatures and therefore $\mathbf{Adv}^{\mathsf{euf-cma}}_{\Sigma,\mathcal{A}} < \mathsf{negl}(\lambda)$

Hence $Pr[\mathsf{sigFailure}_1] < \mathsf{negl}(\lambda)$ and therefore $Pr[D(\mathbf{Hyb}_5) = 1] - Pr[D(\mathbf{Hyb}_4) = 1] < \mathsf{negl}(\lambda)$ □

Lemma 7. *Assuming EUF-CMA signatures,* $\mathbf{Hyb}_6$ *and* $\mathbf{Hyb}_5$ *are indistinguishable.*

Proof. Similar to the proof for Lemma 6 □

Lemma 8. *Assuming EUF-CMA signatures,* $\mathbf{Hyb}_7$ *and* $\mathbf{Hyb}_6$ *are indistinguishable.*

Proof. Similar to the proof for Lemma 6 □

Lemma 9. *Assuming that DDH holds in the chosen algebraic group model,* $\mathbf{Hyb}_8$ *and* $\mathbf{Hyb}_7$ *are computationally indistinguishable.*

Proof. Note that the difference between $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ is that for recipients, the exchanged key is randomly sampled as g^c instead of g^{ab}.

Assume that there exists a distinguisher D that can distinguish between $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$, i.e.

$$Pr[D(\mathbf{Hyb}_8) = 1] - Pr[D(\mathbf{Hyb}_7) = 1] > \mathsf{negl}(\lambda)$$

We use this distinguisher to break the DDH assumption. Recall that the DDH assumption states that no PPT adversary can distinguish between the tuples (g^a, g^b, g^{ab}) and (g^a, g^b, g^c), where $c \leftarrow \mathbb{Z}_q$

1. Simulate the protocol as in $\mathbf{Hyb}_7$
2. Receive (g^a, g^b, g^z) from the challenger
3. In the "keyex" function for a recipient, replace the output as $(eid, (g^a, g^b), \sigma)$ and set the key as g^z.

The simulator $\mathcal{S}$ maintains a public **board** and internally simulates $\mathcal{G}_{\text{att}}$ towards the adversary $\mathcal{A}$

Setup For each recipient R_i: Same as in Fig 17

`WRITE` :

1. Upon receiving $(\texttt{WRITE}, (S_i, m))$ from $\mathcal{F}_{\mathsf{privSignal}}$, update $\mathbf{board} = \mathbf{board} \| m$ and send `ok` to $\mathcal{F}_{\mathsf{privSignal}}$.
2. Upon receiving $(\mathsf{WriteBoard}, m)$ from $\mathcal{A}$, send $(\texttt{WRITE}, m)$ to $\mathcal{F}_{\mathsf{privSignal}}$. Upon receiving $(\texttt{WRITE}, (S_i, m))$ from $\mathcal{F}_{\mathsf{privSignal}}$, update $\mathbf{board} = \mathbf{board} \| m$ and send $(\mathbf{board}, \mathsf{loc})$ to $\mathcal{A}$, where loc is the last position on **board** and send `ok` to $\mathcal{F}_{\mathsf{privSignal}}$.

`SEND`: Upon receiving $(\texttt{SEND}, S_i)$ from $\mathcal{F}_{\mathsf{privSignal}}$, same as in Fig 17.
Upon receiving $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid_j, (\text{“send”}, \vec{L}_j, \mathsf{ct}_{\mathsf{Signal}}))$ on behalf of $\mathcal{G}_{\text{att}}$ for a $\mathsf{ct}_{\mathsf{Signal}}$ that was not created by the simulator

Let $\mathsf{msg} = \mathsf{Dec}(\mathsf{sk}_j, \mathsf{ct}_{\mathsf{Signal}})$.
if $\mathsf{msg}[0:\lambda] = 0^\lambda$ **then**
  Let $\mathsf{loc} = \mathsf{msg}[\lambda : 2\lambda]$
  Send $(\texttt{SEND}, R_j, \mathsf{loc})$ to $\mathcal{F}_{\mathsf{privSignal}}$ on behalf of $\mathcal{A}$ and receive back $(\texttt{SEND}, \mathcal{A})$.
for i in $[1, \ell]$ **do**
  $\vec{L}_j[i] = \mathsf{Enc}(\mathsf{SGXkey}_j, 0)$
return $\vec{L}_j$

`RECEIVE` Same as in Fig 17

Figure 14: Simulator $\mathcal{S}_s$ for the case corrupt server and sender

4. Send the transcript to the distinguisher D and output what the D outputs.

Note that when $z = ab$, the distinguisher receives a transcript as in $\mathbf{Hyb}_7$ where as in the case of $\mathbf{Hyb}_8$, D receives a transcript as in $\mathbf{Hyb}_8$.
Thus $Pr[\mathcal{A}(g^a, g^b, g^{ab}) = 1] = Pr[D(\mathbf{Hyb}_7) = 1]$ and $Pr[\mathcal{A}(g^a, g^b, g^c) = 1] = Pr[D(\mathbf{Hyb}_8) = 1]$
Thus $|Pr[\mathcal{A}(g^a, g^b, g^{ab}) = 1] - Pr[\mathcal{A}(g^a, g^b, g^c) = 1]|$
$= Pr[D(\mathbf{Hyb}_8) = 1] - Pr[D(\mathbf{Hyb}_7) = 1]$ which is $> \mathsf{negl}(\lambda)$
But this is a contradiction since this breaks the DDH assumption.
Therefore $\mathbf{Hyb}_8$ and $\mathbf{Hyb}_7$ are indistinguishable. □

B.4 Case 2: S and Server are corrupt

Simulator Overview In this case a subset of the senders are corrupt along with the server. To prove security we need to construct a simulator that interacts with the adversary and the $\mathcal{F}_{\mathsf{privSignal}}$ functionality that is indistinguishable from the real world.

Proof by hybrids Through hybrid arguments we move from the real world to the ideal world and prove that each pair of intermediate hybrids are indistinguishable.

- $\mathbf{Hyb}_0$ is the real world.
- $\mathbf{Hyb}_1$ is the same as $\mathbf{Hyb}_0$ except that for an honest sender, encryptions of 0 are sent instead of the actual locations in the `SEND` command. By the CPA security of the underlying encryption

scheme, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are indistinguishable.

- $\mathbf{Hyb}_2$ is the same as $\mathbf{Hyb}_1$ except that for an honest sender, a fresh public key is used to encrypt the location instead of the public key of the recipient. By the key-privacy of the underlying encryption scheme, $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are indistinguishable.
- $\mathbf{Hyb}_3$ is the same as $\mathbf{Hyb}_2$ except that for a malicious sender, the `SEND` command is done as in the simulation. By the CPA property of the encryption scheme the hybrids $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ are indistinguishable.
- $\mathbf{Hyb}_4$ is the same as $\mathbf{Hyb}_3$ except that for `RECEIVE` command, the simulator returns encryptions of 0 to the server as $\mathcal{G}_{\text{att}}$. By the CPA security of the underlying encryption scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_5$ is the same as $\mathbf{Hyb}_4$ except that the "keyex" of **Setup** procedure is done as in the simulation and the simulator might abort with $\mathsf{sigFailure}_1$. By the EUF-CMA property of the signature scheme, the two hybrids are indistiguishable.
- $\mathbf{Hyb}_6$ is the same as $\mathbf{Hyb}_5$ except that the "setup" of **Setup** procedure is done as in the simulation and the simulator might abort with $\mathsf{sigFailure}_2$. By the EUF-CMA property of the signature scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_7$ is the same as $\mathbf{Hyb}_6$ except that the `RECEIVE` is done as in the simulation and the simulator might abort with $\mathsf{sigFailure}_2$. By the EUF-CMA property of the signature scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_8$ is the same as $\mathbf{Hyb}_7$ except that in the **Setup** procedure, the key k_i is computed as $k_i \leftarrow g^c$ where $c \leftarrow_\$ \mathbb{Z}_q$. Under the DDH assumption this occurs with negligible probability and proof is the same as in Lemma 9

B.5 Case 3: Server and R are corrupt

Simulator overview In this case, the simulator needs to simulate a view towards a malicious recipient that is indistinguishable from the real world interaction. The challenge in this simulation is that the simulator needs to know when a malicious recipient requests from the Server for its row. But since we design the protocol such that the Server must request the $\mathcal{G}_{\text{att}}$ functionality for the signals, the simulator is notified of this `RECEIVE` request. Next the simulator needs to to send the correct locations to the recipient, even though the simulated $\vec{L}$ are just an encryption of 0s. To this end, the simulator simply sends the `RECEIVE` command to the $\mathcal{F}_{\mathsf{privSignal}}$ functionality and learns the locations that correspond to the malicious recipient. It then simulates the $\mathcal{G}_{\text{att}}$ functionality to compute an encryption of the locations and sends it back to the Server.

Proof by hybrids

- $\mathbf{Hyb}_0$ The real world protocol.
- $\mathbf{Hyb}_1$ is the same as $\mathbf{Hyb}_0$ except that upon receiving a `SEND` command, the $\mathsf{ct}_{\mathsf{Signal}}$ is replaced with an encryption to 0 instead of the actual location. By the CPA security (Def 3) of the underlying encryption scheme we prove in Lemma 2 that the two hybrids are indistinguishable.
- $\mathbf{Hyb}_2$ is the same as $\mathbf{Hyb}_1$ except that for each `SEND` command, the loc is now encrypted under a fresh public key as in the simulation. By the key privacy (Def 5) of the underlying encryption scheme we prove in Lemma 3 that the two hybrids are indistinguishable.
- $\mathbf{Hyb}_3$ is the same as $\mathbf{Hyb}_2$ except that in the `SEND` command, the $\mathcal{G}_{\text{att}}$ functionality returns encryptions of 0 under SGXkey_j instead of actual locations. By the CPA security (Def 2) of the encryption scheme, the two hybrids are indistinguishable.

The simulator $\mathcal{S}$ maintains a public **board** and internally simulates $\mathcal{G}_{\text{att}}$ towards the adversary $\mathcal{A}$

Setup For each honest recipient R_i: Same as in Fig 17
Upon receiving $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid_i, \text{“keyex”}, g^a)$ from the corrupt Server (for which g^a was not created by $\mathcal{S}_r$) on behalf of $\mathcal{G}_{\text{att}}$:

1. Randomly sample g^b and send (g^a, g^b, σ) and store $k = g^{ab}$ as the secret key of the corrupt recipient.
2. Compute $\mathsf{Sig}(\mathsf{msk}, (eid_i, (g^a, g^b)))$ and send $((g^a, g^b), \sigma)$ to the Server

Upon receiving $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid_i, (\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i})$ from Server on behalf of $\mathcal{G}_{\text{att}}$:

1. Compute $(\mathsf{sk}_i, \Sigma.\mathsf{vk}_i) = \mathsf{Dec}(k_i, \mathsf{ct}_{\mathsf{keys},i})$ and compute pk_i from sk_i.
2. Compute $\vec{L} = \{\mathsf{Enc}(\mathsf{SGXkey}, 0)\}_{j=0}^{\ell}$
3. Set $\mathsf{index} = 0$ and $\mathsf{ctr} = 0$
4. Compute $\sigma = \Sigma.\mathsf{Sig}(\mathsf{msk}, (eid_i(\mathsf{pk}, \vec{L})))$ and send $(\mathsf{pk}, \vec{L}, \sigma)$ to $\mathcal{A}$ and store $(\mathsf{pk}_i, \mathsf{sk}_i, \Sigma.\mathsf{vk}_i, \mathsf{index}_i, \mathsf{ctr}_i)$.

`WRITE` : Same as in Fig 17

`SEND`: Same as in Fig 17

`RECEIVE` For honest recipients, same as in Fig 17.
// *For malicious recipients*

1. Receive $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid_i, (\text{“receive”}, \mathsf{ctr}_i, \sigma_i, \vec{L}_i))$ from Server on behalf of $\mathcal{G}_{\text{att}}$. If $\mathsf{Sig.Ver}(\Sigma.\mathsf{vk}_i, \mathsf{ctr}_i, \sigma_i) = 0$, return $\perp$. Else if i corresponds to that of an honest recipient, abort with $\mathsf{sigFailure}$. Else:
2. Send $(\texttt{RECEIVE}, R_i)$ to $\mathcal{F}_{\mathsf{privSignal}}$ on behalf of R_i and get back $[\mathsf{loc}_1 \dots \mathsf{loc}_\ell]$. If less than ℓ locations received, pad with 0.
3. Compute $\mathsf{ct}_{\mathsf{loc},i} = (\mathsf{Enc}(\mathsf{pk}_i, \mathsf{loc}_1) \dots \mathsf{Enc}(\mathsf{pk}_i, \mathsf{loc}_\ell))$ and compute $\vec{L}_i = \{\mathsf{Enc}(\mathsf{SGXkey}, 0)\}_{j=0}^{\ell}$.
4. Compute $\sigma_T = \Sigma.\mathsf{Sig}(\mathsf{msk}, (\vec{L}, \mathsf{ct}_{\mathsf{loc},i}))$ and send $(\vec{L}_i, \mathsf{ct}_{\mathsf{loc},i}, \sigma_T)$ to $\mathcal{A}$.

Figure 15: Simulator $\mathcal{S}_r$ for the case corrupt server and recipients

The simulator $\mathcal{S}$ maintains a public **board** and internally simulates $\mathcal{G}_{\text{att}}$ towards the adversary $\mathcal{A}$

Setup For each honest recipient R_i: Same as in Fig 17
For each malicious recipient R_i: Same as in Fig 15

`WRITE` : Same as in Fig 17

`SEND`: Same as in Fig 14

`RECEIVE` For honest recipients, same as in Fig 17.
For each malicious recipient R_i: Same as in Fig 15

Figure 16: Simulator $\mathcal{S}_{sr}$ for the case corrupt server, server and recipients

- $\mathbf{Hyb}_4$ is the same as $\mathbf{Hyb}_3$ except that in the `RECEIVE` command for an honest recipient, the simulator returns encryptions of 0 as $\mathsf{ct}_{\mathsf{loc},i}$ to the server on behalf of $\mathcal{G}_{\text{att}}$. By the CPA security (Def 3) of the underlying encryption scheme, we prove in Lemma 5 that the two hybrids are indistinguishable.
- $\mathbf{Hyb}_5$ is the same as $\mathbf{Hyb}_4$, except that for malicious recipients, the simulator may abort with $\mathsf{sigFailure}$. Since we assume EUF-CMA signatures these two hybrids are indistinguishable.
- $\mathbf{Hyb}_6$ is the same as $\mathbf{Hyb}_5$ except that in the **Setup** procedure, the simulator aborts with $\mathsf{sigFailure}_1$. We prove in Lemma 6 that this occurs with negligible probability.
- $\mathbf{Hyb}_6$ is the same as $\mathbf{Hyb}_5$ except that in the **Setup** procedure, the simulator aborts with $\mathsf{sigFailure}_2$. We prove in Lemma 7 that this occurs with negligible probability.
- $\mathbf{Hyb}_7$ is the same as $\mathbf{Hyb}_6$ except that in the `RECEIVE` command, the simulator may abort with $\mathsf{sigFailure}_3$. We prove in Lemma 8 that this occurs with negligible probability.
- $\mathbf{Hyb}_8$ is the same as $\mathbf{Hyb}_7$ except that in the **Setup** procedure, the key k_i is computed as $k_i \leftarrow g^c$ where $c \leftarrow_\$ \mathbb{Z}_q$. We prove in Lemma 9 that this occurs with negligible probability.

B.6 Case 4: Corrupt Server, S and R

Simulator overview This simulator is a combination of the previous simulators, where the simulator simulates the `SEND` command as in the case when the Server and the sender S are corrupt, for the **Setup** and `RECEIVE` commands the simulator simulates as in the case when the Server and the recipient R are corrupt.

Proof by hybrids

- $\mathbf{Hyb}_0$ is the real world.
- $\mathbf{Hyb}_1$ is the same as $\mathbf{Hyb}_0$ except that for an honest sender, encryptions of 0 are sent instead of the actual locations in the `SEND` command. By the CPA security of the underlying encryption scheme, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are indistinguishable.
- $\mathbf{Hyb}_2$ is the same as $\mathbf{Hyb}_1$ except that for an honest sender, a fresh public key is used to encrypt the location instead of the public key of the recipient. By the key-privacy of the underlying encryption scheme, $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are indistinguishable.

- $\mathbf{Hyb}_3$ is the same as $\mathbf{Hyb}_2$ except that for a malicious sender, the `SEND` command is done as in the simulation. By the CPA property of the encryption scheme the hybrids $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ are indistinguishable.
- $\mathbf{Hyb}_4$ is the same as $\mathbf{Hyb}_3$ except that for `RECEIVE` command, the simulator returns encryptions of 0 to the server as $\mathcal{G}_{\text{att}}$. By the CPA security of the underlying encryption scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_5$ is the same as $\mathbf{Hyb}_4$, except that for malicious recipients, the simulator may abort with $\mathsf{sigFailure}$. Since we assume EUF-CMA signatures these two hybrids are indistinguishable.
- $\mathbf{Hyb}_6$ is the same as $\mathbf{Hyb}_5$ except that the "keyex" of **Setup** procedure is done as in the simulation and the simulator might abort with $\mathsf{sigFailure}_1$. By the EUF-CMA property of the signature scheme, the two hybrids are indistiguishable.
- $\mathbf{Hyb}_7$ is the same as $\mathbf{Hyb}_6$ except that the "setup" of **Setup** procedure is done as in the simulation and the simulator might abort with $\mathsf{sigFailure}_2$. By the EUF-CMA property of the signature scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_8$ is the same as $\mathbf{Hyb}_7$ except that the `RECEIVE` is done as in the simulation and the simulator might abort with $\mathsf{sigFailure}_2$. By the EUF-CMA property of the signature scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_9$ is the same as $\mathbf{Hyb}_8$ except that in the **Setup** procedure, the key k_i is computed as $k_i \leftarrow g^c$ where $c \leftarrow_\$ \mathbb{Z}_q$. Under the DDH assumption this occurs with negligible probability and proof is the same as in Lemma 9

C Proof of security for theorem 2

C.1 Correctness

We require the following correctness guarantees from our protocol.

1. If a sender S sends a signal to recipient R using `SEND` for location loc, then $\mathsf{processSignal}$ ensures that upon `RECEIVE` from from R, it learns of loc.
2. Upon receiving a `RECEIVE` request from the R the rows for R maintained by the servers must be updated to shares of 0.

To check that the first guarantee is satisfied, let there be only one recipient R and the $\mathbb{T}^{(i)}$ maintained by the servers have one column each with the value r. Let $\mathbb{L}^{(i)}$ also be initialized with idx. This implies both tables maintain shares of 0.

That is the last updated index for R is $idx \oplus idx = 0$ and the value at this index is $r \oplus r = 0$

Now Server_1 upon receiving $(R^{(1)}, \mathsf{loc}^{(1)})$ and Server_2 upon receiving $(R^{(2)}, \mathsf{loc}^{(2)})$ from the sender S, run the $\mathsf{processSignal}$ procedure.

Server_i samples r_i and idx_i and creates a garbled circuit using Garble, which includes key labels for the r, idx, maintained by the other Server and the randomly sampled r_i and idx_i by both Server. These key labels are sent to the $\mathcal{F}_{\mathsf{ot}}$ functionality and the keys for $R^{(i)}, \mathsf{loc}^{(i)}$ and r_i, idx_i that are known only to Server_i are directly sent to the other Server. Moreover to evaluate its own circuit, the Server_i must receive same labels from $\mathcal{F}_{\mathsf{ot}}$ and directly from the other Server. By the correctness of the GC protocol, the circuit evaluates the function $\mathsf{UpdateTable}$ and $\mathsf{UpdIndex}$ we have that $\mathbb{T}^{(i)}[R]$ is updated as $\mathsf{loc}^{(i)} \oplus r_1 \oplus r_2 \oplus (r)$ and $\mathbb{L}^{(1)}[R]$ is updated as $1 \oplus idx \oplus idx_1 \oplus idx_2$ and $\mathbb{L}^{(2)}[R]$ is updated as $idx \oplus \mathsf{index}_1 \oplus idx_2$.

Now note that the two $\mathbb{L}^{(1)} \oplus \mathbb{L}^{(2)}$ maintains the invariant of the last updated index for R which is now $1 \oplus idx \oplus idx_1 \oplus idx_2 \oplus idx \oplus idx_1 \oplus idx_2 = 1$ and the value at this index is $\mathbb{T}^{(1)} \oplus \mathbb{T}^{(2)} = \mathsf{loc}^{(1)} \oplus r_1 \oplus r_2 \oplus (r) \oplus \mathsf{loc}^{(2)} \oplus r_1 \oplus r_2 \oplus (r) = \mathsf{loc}^{(1)} \oplus \mathsf{loc}^{(2)} = \mathsf{loc}$.

C.2 Protocol $\Pi_{\mathsf{privSignal2}}$ realizes the $\mathcal{F}_{\mathsf{privSignal}}$ functionality

Proof. To prove that $\Pi_{\mathsf{privSignal}}$ UC-realizes $\mathcal{F}_{\mathsf{privSignal}}$ we show that there exists a simulator $\mathcal{S}$ interacting with $\mathcal{F}_{\mathsf{privSignal}}$ that generates a transcript that is indistinguishable from the transcript generated by the real-world adversary running protocol $\Pi_{\mathsf{privSignal}}$. We consider the following different cases of corruption and define a simulator for each case:

- Simulator $\mathcal{S}_N$ for the case when neither a sender nor a recipient is malicious and only one of the two servers (w.l.o.g. Server_1) is corrupt.
- Simulator $\mathcal{S}_s$ for the case when a sender S is corrupt and colludes with one of the servers.
- Simulator $\mathcal{S}_r$ for the case when a recipient R is corrupt and colludes with one of the servers.
- Simulator $\mathcal{S}_{rs}$ for the case when both R and S are corrupt. Here we consider the case when the recipient and sender colludes with one of the two servers (Server_1).

We discuss these simulators in more detail below.

□

C.3 Case 1: Neither S nor R is corrupt

Simulator overview. When neither S nor R are corrupt, then the only corrupt party is Server_1. Thus the simulator must interact with Server_1 (the adversary) and the functionality $\mathcal{F}_{\mathsf{privSignal}}$ to simulate a view that is indistinguishable from the real world protocol. The simulator maintains **board** and any oracle access for the board ($\mathsf{WriteBoard}, \mathsf{ReadBoard}$) is handled by the simulator. The simulator $\mathcal{S}_N$ simulates the following commands it receives from $\mathcal{F}_{\mathsf{privSignal}}$:

- `WRITE`: The simulator simply updates the **board** as **board**$\|m$.
- `SEND`: The simulator does not know the recipient or the location of the signal, since it only receives the sender identity from $\mathcal{F}_{\mathsf{privSignal}}$. $\mathcal{S}_N$ thus sets $R = 0$ and $\mathsf{loc} = 0$ and creates shares of the same. A share of each is sent to Server_1. Then using the simulator of the garbled circuit $\mathcal{S}_{\mathsf{Garble}}$, it simulates a GC such that the output of the $\mathbb{T}^{(1)}$ and $\mathbb{L}^{(1)}$ are completely random. Note that in the real world as well each index is $\oplus$-ed with a random value from Server_2, thus intuitively this simulation should be indistinguishable from the real world.
- `RECEIVE`: The simulator requests Server_1 for the row corresponding to R_j, where R_j is the recipient for which $\mathcal{F}_{\mathsf{privSignal}}$ received a `RECEIVE` command.

We present the simulator more precisely in Fig 17

Proof by hybrids. We prove security via a sequence of hybrids where we start from the real world and move to the ideal world.

- $\mathbf{Hyb}_0$ The real world protocol.
- $\mathbf{Hyb}_1$ is the same as $\mathbf{Hyb}_0$ except for the `SEND` command the garbled circuit computation is done by $\mathcal{S}_{\mathsf{Garble}}$. We prove in Lemma 10 that $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$ are indistinguishable by the privacy property (Def 1) of garbled circuits.
- $\mathbf{Hyb}_2$ is the same as $\mathbf{Hyb}_1$ except for the `SEND` command, the signal message sent to $\mathcal{A}$ is replaced by a share of 0. Since we use XOR, information theoretically $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are indistinguishable to a computationally unbounded adversary. And this is equivalent to the ideal world.

Lemma 10. *If secure garbled circuits with privacy (Def 1) are used then* $\mathbf{Hyb}_0$ *and* $\mathbf{Hyb}_1$ *are indistinguishable to a PPT adversary.*

The simulator $\mathcal{S}$ maintains a public **board** and is initialized with $\mathcal{S}_{\mathsf{Garble}}$ which is the simulator for garbled circuits.

Setup For each recipient R_i:

1. $(\Sigma.\mathsf{sk}_i, \Sigma.\mathsf{vk}_i) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$ and publish $\Sigma.\mathsf{vk}_i$.
2. Randomly sample $(r_0, \ldots, r_\ell) \leftarrow \mathbb{Z}_q^{\ell+1}$, and initialize $\mathsf{ctr}_i = 0$.
3. Compute $\sigma_i = \Sigma.\mathsf{Sig}(\Sigma.\mathsf{sk}_i, ((r_0 \ldots, r_\ell), \mathsf{ctr}_i))$ and send $(\mathbf{Setup}, (r_0 \ldots, r_\ell), \mathsf{ctr}_i))$ to $\mathcal{A}$.

`WRITE` : Upon receiving $(\texttt{WRITE}, (S_i, m))$ from $\mathcal{F}_{\mathsf{privSignal}}$, update $\mathbf{board} = \mathbf{board} \| m$ and send `ok` to $\mathcal{F}_{\mathsf{privSignal}}$.

`SEND`: Upon receiving $(\texttt{SEND}, S_i)$ from $\mathcal{F}_{\mathsf{privSignal}}$

1. Sample $r_1, r_2 \leftarrow \{0,1\}^\lambda$ and set $R^{(1)} = R^{(2)} = r_1$ and $\mathsf{loc}^{(1)} = \mathsf{loc}^{(2)} = r_2$ and send $R^{(1)}, \mathsf{loc}^{(1)}$ to Server_1.
2. (Simulating $\mathcal{F}_{\mathsf{ot}}$): Receive from $\mathcal{A}$ $(\texttt{SEND}, (s_0, s_1))$ for bits in $|\mathbb{T}^{(2)}|, |\mathsf{loc}^{(2)}|, |R^{(2)}|, \mathbb{L}^{(2)}$ and $|r_{(i,j)}|$. Store these strings and send `SEND` to $\mathcal{A}$.
3. Upon receiving $(\texttt{RECEIVE}, b)$ for bits in $|\mathbb{T}^{(1)}|, |\mathsf{loc}^{(1)}|, |R^{(1)}|$, $\mathbb{L}^{(1)}$ and $|r_{(i,j)}|$, compute $(F, X, d) \leftarrow \mathcal{S}_{\mathsf{Garble}}(1^k, y, (\mathsf{UpdateTable}, \mathsf{UpdIndex}))$, where $y = \mathbb{T}^{(*)}, \mathbb{L}^{(*)}$ and $\mathbb{T}^{(*)} \leftarrow_\$ \{0,1\}^{M \times \ell}$ and $\mathbb{L}^{(*)} \leftarrow_\$ \{0,1\}^M$. Send corresponding bits of X via $\mathcal{F}_{\mathsf{ot}}$ to $\mathcal{A}$ and send (F, d) to $\mathcal{A}$.

`RECEIVE` : Upon receiving $(\texttt{RECEIVE}, R_i)$ from $\mathcal{F}_{\mathsf{privSignal}}$,

1. Randomly sample $(r_0, \ldots, r_\ell) \leftarrow \mathbb{Z}_q^{\ell+1}$.
2. Send $\sigma_i = \Sigma.\mathsf{Sig}(\Sigma.\mathsf{sk}_i, (\mathsf{ctr}_i(r_0, \ldots, r_\ell) \leftarrow \mathbb{Z}_q^{\ell+1}))$ to $\mathcal{A}$
3. Update $\mathsf{ctr}_i = \mathsf{ctr}_i + 1$

Figure 17: Simulator $\mathcal{S}_N$ for the case of only one corrupt server

Proof. Assume towards a contradiction that an adversary $\mathcal{A}_{01}$ can distinguish between $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$. Using this $\mathcal{A}_{01}$ we construct an adversary $\mathcal{A}_{\mathsf{Garble}}$ that breaks the privacy of the garbled circuits (def 1).

$\underline{\mathcal{A}_{\mathsf{Garble}}}$

1. Activate $\mathcal{A}_{01}$.
2. Simulate requests for $\texttt{WRITE}, \texttt{READ}$ as in the real world. For a $\texttt{SEND}$ command, as the garbler of the GC create the encodings as in the protocol, and receive (F, X, d) from the challenger, where X is received from an OT oracle.
3. Forward (F, X, d) to $\mathcal{A}_{01}$.
4. Output whatever $\mathcal{A}_{01}$ outputs.

Analysis
Since $\mathcal{A}_{01}$ can distinguish between $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$ we have:

$$Pr[\mathcal{A}_{01}(\mathbf{Hyb}_0) = 1] - Pr[\mathcal{A}_{01}(\mathbf{Hyb}_1) = 1] > \mathsf{negl}(\lambda)$$

Observe that when $b = 1$ in the PrvSim game, the transcript seen by $\mathcal{A}_{01}$ is exactly as in the $\mathbf{Hyb}_0$ since the garbled circuit is created honestly. Similarly, when $b = 0$, the transcript seen by $\mathcal{A}_{01}$ is exactly as in $\mathbf{Hyb}_1$ since the simulator $\mathcal{S}_{\mathsf{Garble}}$ is used to compute (F, X, d).

Thus we have $Pr[\mathcal{A}_{\mathsf{Garble}}(\mathsf{PrvSim}) = 1 | b = 1] = Pr[\mathcal{A}_{01}(\mathbf{Hyb}_0) = 0]$ and $Pr[\mathcal{A}_{\mathsf{Garble}}(\mathsf{PrvSim}) = 0 | b = 0] = Pr[\mathcal{A}_{01}(\mathbf{Hyb}_1) = 1]$

Thus $Pr[\mathsf{PrvSim}^{\mathcal{A}}_{\mathcal{G},\phi,\mathcal{S}}(k)]$ which is $1/2 Pr[\mathcal{A}_{\mathsf{Garble}}(\mathsf{PrvSim}) = 1 | b = 1] + 1/2 Pr[\mathcal{A}_{\mathsf{Garble}}(\mathsf{PrvSim}) = 0 | b = 0]$

And this is equal to

$$1/2 | Pr[\mathcal{A}_{01}(\mathbf{Hyb}_0) = 0] + 1/2 Pr[\mathcal{A}_{01}(\mathbf{Hyb}_1) = 1] |$$

$$= | 1/2 - 1/2 Pr[\mathcal{A}_{01}(\mathbf{Hyb}_0) = 1] + 1/2 Pr[\mathcal{A}_{01}(\mathbf{Hyb}_1) = 1] |$$

$$> 1/2 + \mathsf{negl}(\lambda)$$

which is non-negligible and this implies

$$\mathbf{Adv}^{\text{prv.sim},\phi,\mathcal{S}}_{\mathcal{G}}(\mathcal{A}, k) > 2(1/2 + \mathsf{negl}(\lambda)) - 1 > \mathsf{negl}(\lambda)$$

which is a contradiction and that concludes our proof.

□

C.4 S and Server_1 are corrupt

Simulator Overview In this setting, the simulator is similar to the previous case, except that the simulator cannot directly compute the recipient and the location. It cannot do so because it receives only one share of the recipient's index ($R^{(2)}$) and the location ($\mathsf{loc}^{(2)}$). But note that the simulator internally simulates $\mathcal{F}_{\mathsf{ot}}$ towards the $\mathcal{A}$. And the $\mathcal{A}$ must send bits of $R^{(1)}$ and $\mathsf{loc}^{(1)}$ to get the corresponding keys to evaluate its GC. At this point the simulator can learn the bits of $\mathsf{loc}^{(1)}$ and $R^{(1)}$. The simulator then proceeds as in the previous case and sends a $\texttt{SEND}, R, \mathsf{loc}$ message to the functionality. To simulate the rest of the interaction with Server_1, the simulator simply calls $\mathcal{S}_{\mathsf{Garble}}$ as in Case C.3.

Proof by hybrids

The simulator $\mathcal{S}$ maintains a public **board** and also has access to $\mathcal{S}_{\mathsf{Garble}}$

Setup As in Fig 17

`WRITE` :

1. Upon receiving $(\texttt{WRITE}, (S_i, m))$ from $\mathcal{F}_{\mathsf{privSignal}}$, update $\mathbf{board} = \mathbf{board} \| m$ and send `ok` to $\mathcal{F}_{\mathsf{privSignal}}$.
2. Upon receiving $(\mathsf{WriteBoard}, m)$ from $\mathcal{A}$, send $(\texttt{WRITE}, m)$ to $\mathcal{F}_{\mathsf{privSignal}}$. Upon receiving $(\texttt{WRITE}, (S_i, m))$ from $\mathcal{F}_{\mathsf{privSignal}}$, update $\mathbf{board} = \mathbf{board} \| m$ and send $(\mathbf{board}, \mathsf{loc})$ to $\mathcal{A}$, where loc is the last position on **board** and send `ok` to $\mathcal{F}_{\mathsf{privSignal}}$.

`SEND`:

1. Upon receiving $(\texttt{SEND}, S_i)$ from $\mathcal{F}_{\mathsf{privSignal}}$, simulate `SEND` as in Fig 17
2. Upon receiving $(R^{(2)}, \mathsf{loc}^{(2)})$ from $\mathcal{A}$
   - (As $\mathcal{F}_{\mathsf{ot}}$): Send `SEND` to $\mathcal{A}$ and receive $(\texttt{RECEIVE}, b)$ from $\mathcal{A}$ for each bit in $\mathbb{T}^{(1)}, \mathbb{L}^{(1)}, \mathsf{loc}^{(1)}$ and $R^{(1)}$.
   - Compute $\mathsf{loc}^{(1)}$ and $R^{(1)}$ from the received bits and compute $\mathsf{loc} = \mathsf{loc}^{(1)} \oplus \mathsf{loc}^{(2)}$ and $R = R^{(1)} \oplus R^{(2)}$.
   - Send $(\texttt{SEND}, R, \mathsf{loc})$ to $\mathcal{F}_{\mathsf{privSignal}}$ on behalf of $\mathcal{A}$ and upon receiving $(\texttt{SEND}, \mathcal{A})$ from $\mathcal{F}_{\mathsf{privSignal}}$ use $\mathcal{S}_{\mathsf{Garble}}$ to compute $(F, X, d) \leftarrow \mathcal{S}_{\mathsf{Garble}}(1^k, y, (\mathsf{UpdateTable}, \mathsf{UpdIndex}))$, where $y = \mathbb{T}^{(*)}, \mathbb{L}^{(*)}$ and $\mathbb{T}^{(*)} \leftarrow_{\$} \{0,1\}^{M \times \ell}$ and $\mathbb{L}^{(*)} \leftarrow_{\$} \{0,1\}^{M}$. Send keys for corresponding bits of X via $\mathcal{F}_{\mathsf{ot}}$ to $\mathcal{A}$ and send (F, d) to $\mathcal{A}$.
   - Upon receiving $(\texttt{SEND}, (s_0, s_1))$ from $\mathcal{A}$ for bits in $|\mathbb{T}^{(2)}|, |\mathsf{loc}^{(2)}|, |R^{(2)}|$, $\mathbb{L}^{(2)}$ and $|r_{(i,j)}|$. Store these strings and send `SEND` to $\mathcal{A}$.

`RECEIVE`: Simulate `RECEIVE` as in Fig 17

`READ`: Upon receiving $(\mathsf{ReadBoard}, \mathsf{loc})$ from $\mathcal{A}$, send $(\texttt{READ}, \mathsf{loc})$ to $\mathcal{F}_{\mathsf{privSignal}}$ and upon receiving $(\texttt{READ}, m)$, check that $\mathbf{board}[\mathsf{loc}] = m$. If yes, send m to $\mathcal{A}$ else abort.

Figure 18: Case when a sender and Server_1 are corrupt

The simulator $\mathcal{S}$ maintains a public **board** and also has access to $\mathcal{S}_{\mathsf{Garble}}$

`WRITE` : Upon receiving $(\texttt{WRITE}, (S_i, m))$ from $\mathcal{F}_{\mathsf{privSignal}}$, update $\mathbf{board} = \mathbf{board} \| m$ and send `ok` to $\mathcal{F}_{\mathsf{privSignal}}$.

`SEND`: Upon receiving $(\texttt{SEND}, S_i)$ from $\mathcal{F}_{\mathsf{privSignal}}$, simulate as in Fig 17.

`RECEIVE` :

- Upon receiving $(\texttt{RECEIVE}, R_j)$, randomly sample $(r_0, \ldots, r_\ell) \leftarrow \mathbb{Z}_q^{\ell+1}$. Send $\sigma_i = \Sigma.\mathsf{Sig}(\Sigma.\mathsf{sk}_i, (r_0, \ldots, r_\ell), a)$ to Server_a and upon receiving $\mathbb{T}^{(1)}[R_j]$ from $\mathcal{A}$, send `ok` to $\mathcal{F}_{\mathsf{privSignal}}$.
- Upon receiving $\sigma_j = \Sigma.\mathsf{Sig}(\Sigma.\mathsf{sk}_j, (\mathsf{ctr}', r_0, \ldots, r_\ell), 2)$ from $\mathcal{A}$ (on behalf of party R_j), check that the signatures verify and $\mathsf{ctr} = \mathsf{ctr}'$. Else ignore.
- If the signatures correspond to that of an honest party abort with error $\mathsf{UnforgeError}_1$, else send `RECEIVE` to $\mathcal{F}_{\mathsf{privSignal}}$ on behalf of R_j and receive $\mathsf{loc}_1 \ldots \mathsf{loc}_\ell$ from $\mathcal{F}_{\mathsf{privSignal}}$. Send $\left[\mathsf{loc}_1 \oplus \mathbb{T}^{(2)}[R_j], \ldots, \mathsf{loc}_\ell \oplus \mathbb{T}^{(2)}[R_j]\right]$ to $\mathcal{A}$.

Figure 19: Simulator for the case when a recipient and Server_1 are corrupt

- $\mathbf{Hyb}_0$: The real world
- $\mathbf{Hyb}_1$ is the same as $\mathbf{Hyb}_0$ except that $\mathcal{S}_{\mathsf{Garble}}$ is used to create the garbled circuit and the encodings for Server_1. $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are indistinguishable by the privacy property of garbled circuits and the proof follows from Lemma 10.
- $\mathbf{Hyb}_2$ is the same as $\mathbf{Hyb}_1$ except that if $\mathcal{A}$ is calls a `SEND` procedure, the simulator now extracts $\mathsf{loc}^{(1)}$ and $R^{(1)}$ from $\mathcal{F}_{\mathsf{ot}}$, computes R and loc and sends the `SEND` command to $\mathcal{F}_{\mathsf{privSignal}}$. Since we use ideal $\mathcal{F}_{\mathsf{ot}}$ and assume that $\mathsf{loc}^{(1)} \oplus \mathsf{loc}^{(2)}$ will give some loc and $R^{(1)} \oplus R^{(2)}$ gives some valid recipient, the two hybrids are indistinguishable. And $\mathbf{Hyb}_2$ is equal to the ideal world.

C.5 R and Server_1 are corrupt

Overview of the simulator The simulator for the `SEND` procedure will work similar to the one in Sec C.3. This is so because we have one server corrupt and the sender is not corrupt. Thus instead of the actual R and loc we use shares of 0. Moreover $\mathcal{S}_{\mathsf{Garble}}$ will be used to simulate the garbled circuit towards the corrupted server. But then for the `RECEIVE` command, the simulator does not have the loc for the corresponding recipient. To this end, the simulator just sends the `RECEIVE` command to the $\mathcal{F}_{\mathsf{privSignal}}$ functionality on behalf of the adversary $\mathcal{A}$. Upon receiving $\mathsf{loc}_1 \ldots \mathsf{loc}_\ell$, the simulator updates $\mathbb{T}^{(2)}[R]$ by $\oplus$ing each $\mathbb{T}^{(2)}[R][i]$ with loc_i. This ensures that simulation is correct since, the $\mathcal{A}$ now receives this updated $\mathbb{T}^{(2)}[R]$ which when $\oplus$-ed with $\mathbb{T}^{(1)}[R]$ will give the locations of signals for the adversary. We present the simulator formally in Fig 19

Proof by hybrids

- $\mathbf{Hyb}_0$ is the real world.
- $\mathbf{Hyb}_1$ is the same as $\mathbf{Hyb}_0$ except that for each `SEND` the simulator now uses a simulated GC instead of the real world GC and responses to `RECEIVE` are done as in the simulation. . $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are indistinguishable by the privacy property of GC and this follows the same proof as in Lemma 10. Note that the two tables maintain shares of 0. That is $\mathbb{T}^{(1)}[R_i][j] \oplus \mathbb{T}^{(2)}[R_i][j] = 0$ for

all i and j. Since in the simulation, the simulator sends $\mathsf{loc}_j \oplus \mathbb{T}^{(2)}[R_i][j]$ to the $\mathcal{A}$ (who already has $\mathbb{T}^{(1)}[R_i]$) the $\mathcal{A}$ can get the correct locations as $\mathsf{loc}_j \oplus \mathbb{T}^{(2)}[R_i][j] \oplus \mathbb{T}^{(1)}[R_i] = \mathsf{loc}_j \oplus 0 = \mathsf{loc}_j$.

- $\mathbf{Hyb}_2$ is the same as $\mathbf{Hyb}_1$ except that the simulator may abort with $\mathsf{UnforgeError}_1$. We prove in Lemma 11 that UF-CMA property of the underlying signature scheme $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are indistinguishable.

Lemma 11. *Assuming existential unforgeable signatures that are secure against chosen message attacks,* $\mathbf{Hyb}_2$ *and* $\mathbf{Hyb}_1$ *are indistinguishable.*

Proof. Note that the difference between $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ is that in $\mathbf{Hyb}_2$ the event $\mathsf{UnforgeError}_1$ can occur. We prove in this section that the probability of this event occurring is negligible.

First we observe that $\mathsf{UnforgeError}_1$ occurs when the simulator receives a signature from the adversary that is valid and corresponds to that of an honest party.

Assume a distinguisher D can distinguish between $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$, i,e. $Pr[D(\mathbf{Hyb}_2) = 1] - Pr[D(\mathbf{Hyb}_1) = 1] > \mathsf{negl}(\lambda)$

This implies that $Pr[\mathsf{UnforgeError}_1] > \mathsf{negl}(\lambda)$.

Which implies that $\mathcal{A}$ sends a $Pr[\Sigma.\mathsf{Ver}(\Sigma.\mathsf{vk}_j, (\mathsf{ctr}', (r_0, \ldots r_\ell), 2), \sigma) = 1 \wedge R_j \text{ is honest } \wedge \mathsf{ctr}'_j = \mathsf{ctr}_j] > \mathsf{negl}(\lambda)$

Using this adversary we present a reduction $\mathcal{B}$ that breaks the EUF-CMA property (Def 4) of signature schemes.

1. Simulate the world as in $\mathbf{Hyb}_1$, and receive $\Sigma.\mathsf{vk}$ from the challenger. Set an honest recipient R_j's verification key to be $\Sigma.\mathsf{vk}$ and publish it.
2. Upon receiving $(\mathsf{ctr}', (r_0, \ldots, r_\ell, 2, \sigma)$, check if $\mathsf{ctr} = \mathsf{ctr}'$ and $\Sigma.\mathsf{Ver}(\Sigma.\mathsf{vk}_j, (\mathsf{ctr}', (r_0, \ldots r_\ell), 2), \sigma) = 1$.
3. If yes, output $(\mathsf{ctr}', (r_0, \ldots, r_\ell, 2), \sigma)$ to the challenger

Observe that

$$\mathbf{Adv}^{\mathsf{euf-cma}}_{\Sigma,\mathcal{B}} = Pr[\mathbf{Exp}^{\mathsf{euf-cma}}_{\Sigma,\mathcal{B}}(\lambda) = 1]$$
$$= Pr[\Sigma.\mathsf{Ver}(\Sigma.\mathsf{vk}_j, (\mathsf{ctr}', (r_0, \ldots r_\ell), 2), \sigma) = 1] > \mathsf{negl}(\lambda)$$

But this is a contradiction since we assume EUF-CMA signatures and therefore $\mathbf{Adv}^{\mathsf{euf-cma}}_{\Sigma,\mathcal{A}} < \mathsf{negl}(\lambda)$

Hence $Pr[\mathsf{UnforgeError}_1] < \mathsf{negl}(\lambda)$ and therefore $Pr[D(\mathbf{Hyb}_2) = 1] - Pr[D(\mathbf{Hyb}_1) = 1] < \mathsf{negl}(\lambda)$. □

C.6 S, R and Server_1 are corrupt

Overview of simulator In this case the simulator needs to simulate a view on behalf of Server_2 only. To this end, when a `SEND` command is received, the simulation is done exactly as in Case C.4 and when for `RECEIVE`, the simulation is done as in Case C.5

Proof by hybrids

- $\mathbf{Hyb}_0$ is the real world hybrid.
- $\mathbf{Hyb}_1$ is the same as $\mathbf{Hyb}_0$ except that the GC is now simulated and response to `RECEIVE` is returned as in the simulation. By a proof following Lemma 10 we have that $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are indistinguishable.

The simulator $\mathcal{S}$ maintains a public **board** and also has access to $\mathcal{S}_{\mathsf{Garble}}$

`WRITE` : Upon receiving $(\texttt{WRITE}, (S_i, m))$ from $\mathcal{F}_{\mathsf{privSignal}}$, update $\mathbf{board} = \mathbf{board} \| m$ and send `ok` to $\mathcal{F}_{\mathsf{privSignal}}$.

`SEND`: Similar to simulation in Fig 18

`RECEIVE`: Similar to simulation in Fig 19

Figure 20: Simulator for the case when a recipient, sender and Server_1 are corrupt

- $\mathbf{Hyb}_2$ is the same $\mathbf{Hyb}_1$ except that the `SEND` is now replaced by shares of 0. Since Server_1 can only receive one share, information theoretically the Server cannot distinguish if its a share of 0 or the actual location. Thus $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ are indistinguisihable.
- $\mathbf{Hyb}_3$ is the same as $\mathbf{Hyb}_2$ except that the simulator may abort if $\mathsf{UnforgeError}_1$ in a `RECEIVE` command occurs. By a proof following Lemma 11, $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_2$ are indistinguishable. And this is the same as in the ideal world.

C.7 Inefficient solutions with no servers

As described in the introduction a naive scan for messages on the **board** by the recipient would require $\mathcal{O}(N)$ computation from the recipient, where N is the total number of messages on the **board**.

An idea to improve the recipient's complexity in a setting without any servers is the following: Each recipient R_i initializes a counter that is encrypted under its public key using an additively homomorphic encryption scheme: $\mathsf{ct}^i_{\mathsf{ctr}} = \mathsf{Enc}(\mathsf{pk}_i, 0)$. Now when a sender wishes to signal that a message is for recipient R_i, the sender does the following computation: for R_i update $\mathsf{ct}^i_{\mathsf{ctr}} = \mathsf{ct}^i_{\mathsf{ctr}} + \mathsf{Enc}(\mathsf{pk}_i, 1)$ and for all other R_j update $\mathsf{ct}^j_{\mathsf{ctr}} = \mathsf{ct}^j_{\mathsf{ctr}} + \mathsf{Enc}(\mathsf{pk}_j, 0)$ and post $\{\mathsf{ct}^i_{\mathsf{ctr}}\}_{i=1}^M$ to the **board** along with the message. At a later point of time, the recipient R_i retrieves the latest $\mathsf{ct}^i_{\mathsf{ctr}}$ and decrypts it to learn the value of ctr_i. If $\mathsf{ctr}_i > 0$ then there exists a message for R_i. The recipient can then perform a binary search on the **board** to find in which locations the counter was incremented and thus learn the messages for itself. If the total number of messages on the **board** is N and the number of messages for the recipient is ℓ, then the recipient will have to perform $\mathcal{O}(\ell \log N)$ computation. But the sender computation and the signal size blows up to $\mathcal{O}(M)$.