Private Signaling

Varun Madathil and Alessandra Scafuro, *North Carolina State University;* István András Seres, *Eötvös Loránd University;* Omer Shlomovits and Denis Varlakov, *ZenGo X*

https://www.usenix.org/conference/usenixsecurity22/presentation/madathil

This paper is included in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

Private Signaling

Varun Madathil[*]
North Carolina State University

Alessandra Scafuro
North Carolina State University

István András Seres
Eötvös Loránd University

Omer Shlomovits
ZenGo X

Denis Varlakov
ZenGo X

Abstract

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

Existing techniques, such as server-aided fuzzy message detection (Beck et al., CCS'21), could be employed to solve the private signaling problem. However, this solution leads to a trade-off between privacy and efficiency, where the complexity of the recipient grows with the required privacy. Specifically, this would require a scan of the entire board to obtain full privacy for the recipient.

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.

Specifically, we provide three contributions: First, we provide a formal definition of private signaling in the Universal Composability (UC) framework and show that it captures several real-world settings where recipient anonymity is desired. Second, we present two server-aided protocols that UC-realize our definitions: one using a single server equipped with a trusted execution environment, and one based on two servers that employ garbled circuits. Third, we provide an *open-source* implementation of both of our protocols, evaluate their performance, and identify for which sets of parameters they can be practical.

1 Introduction

Problem Statement. We focus on the problem of *recipient anonymity*. In its abstraction, there are M recipients $R_1,\dots,R_M$ publicly identified by their public keys $pk_1,\dots,pk_M$. There is a public venue such as a bulletin board that collects messages $(m_1,m_2,m_3,\dots)$ 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, *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 [10] and stealth payments [12]. We describe these specific applications in greater length in Sec 2.1. For the remainder of the introduction, we will focus on the general abstraction above.

Private Signaling: the Naive Inefficient Approach. A straightforward (though inefficient) solution for the private signaling problem would be as follows. The sender who intends to communicate that a message is located at loc to R_i can simply encrypt loc 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 where the messages for the recipient are. 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. In this work, we are interested in reducing the communication and computational complexity

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

[1] Key-private means that by looking at the ciphertext, no one can distinguish which public key was used for encrypting the message [6].

of the recipient.

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, a serverless solution would lead to complexity $O(N)$ for each recipient. Alternatively, one can trade the search time with the signal size. Namely, search complexity can be lowered to $O(\log N)$ per message for the recipient if the size of the signal grows with the total number of possible recipients, that is, $O(M)$, which can be still very inefficient for even moderate M (we describe this in the full version [20]).

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 [5] 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?

1.1 Our contribution

We answer affirmatively to the question above. 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 [9]. Thus, we define an ideal functionality $\mathcal{F}_{\mathsf{privSignal}}$ 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 [17, 22, 32], or provide much weaker security guarantees [5]. We elaborate in Sec 5.
2. **Protocols for private signaling with constant recipient overhead and *provable* UC-security.** The focus of this work is to *minimize the costs* for the recipients and senders. We provide two protocols that UC-realize the ideal functionality $\mathcal{F}_{\mathsf{privSignal}}$ where a sender only needs to perform one (or two) encryptions to compute a signal, and a recipient does not need to perform any scan, and will just perform a number of decryptions that matches the number of received signals. We provide two protocols: one based on garbled circuits that requires two servers, and one leveraging on a Trusted Execution Environment (TEE) which requires a single server only (our approach is explained in Sec. 3).
3. **Open-source Implementations.** We implement both our protocols and measure their efficiency. We compare our performances with related work (we elaborate in Sec. 9).

2 Background for Private Signaling

2.1 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 [31]) 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 [12]. 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 [26]. 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.[2] 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.

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

Anonymous messaging. Modern private messaging applications are mostly focused on providing and improving sender anonymity [10, 21], 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) [30] or to a shared public bulletin board, as in Riposte [10], 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 public 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 in 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.

2.2 Related and Concurrent Work

The closest work to ours is Fuzzy Message Detection by Beck et al [5]. Subsequently to our work, the definition of Oblivious Message Retrieval was introduced by Liu et al [19]. In this section we describe these works. A comparison in terms of asymptotic efficiency in provided in Table 1 and concrete efficiency in Table 4.

Fuzzy Message Detection (FMD) [5] Fuzzy message detection is a primitive that allows a server to do outsourced message detection. The recipient of a message provides the server with a "fuzzy" detection key that identifies the relevant ciphertexts as well non-matching flag ciphertexts with some false positive rate. This false positive rate is set by the recipient of the messages. The untrusted server that performs the fuzzy detection, must be unable to distinguish between a correct detection result and a false-positive.

Privacy. The privacy guaranteed by FMD is k-anonymity, which suffers of known attacks ([18] [28] show how the untrusted server can break recipient unlinkability and relationship anonymity). In this work, we aim at the strongest privacy guarantee, where each recipient has an anonymity set that is as large as the total number of honest recipients and senders (see Sec 5 to for details on our definition).

Efficiency. In FMD, the senders need to compute γ (a constant of the order 10) number of encryptions and send them to the server. If N is the total number of messages that were sent to the server, each recipient will receive ρN messages where ρ is a false positive rate. 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. Note that recipients determine ρ in FMD and can therefore *trade-off privacy for efficiency*. By setting ρ to be a small value, the number of decryptions done by the recipient will also reduce. In this work instead we aim to minimize the work of the sender and the receiver. As we shall see in Sec 6 and 7 and as depicted in Table 1, in our protocol the sender only sends one (or two) encryptions, and the recipient needs to decrypt exactly the number of signals it receives.

Assumptions and Threat-model. FMD relies on a single untrusted server only. Instead, in this work we rely either on two non-colluding servers, or on the trusted execution environment (TEE) [11, 24]. In FMD, security is provided via game-based proofs in presence of a semi-honest server. In contrast, in this work we define an ideal functionality for private signaling in the UC-model and consider either two semi-honest server or a malicious server equipped with TEE.

Oblivious Message Retrieval (OMR) [19] OMR is a recent work by Liu et al. that appeared subsequently to our work. OMR is another primitive that allows the recipient to provide a detection key to an untrusted server so that they can receive pertinent private messages that are posted to a public board. Their aim is to not only detect messages but also to retrieve the messages from the server. They present two protocols OMR2 and OMR3, based on fully homomorphic encryption (FHE). In their protocols, a sender encrypts the message under the receiver's public key and post it to a board. A recipient requests its messages from a server by sending a detection (FHE key) along with a bound on the number of messages it may receive. The server then rencrypts each ciphertext on the board under this new FHE key such that it either decrypts to the message if it corresponds to the recipient or to zero otherwise. Finally, the encryptions are cleverly compacted so that the recipient does not have to do decryptions linear in the number of total messages on the board.

Efficiency. OMR2 requires the receiver to do $O(N)$ decryptions, where N is the total number of messages on the board. OMR3 on the other hand, is optimized with compact detection but still requires $O(\mathrm{poly}\log(N))$ computation for the recipient. In contrast, in our protocol recipients will only need to perform decryptions equal to the number of messages they receive on the board. In both OMR2 and OMR3, the detection cost for the servers grows with N, whereas in our protocols, the detection cost grows with M, which is the number of recipients that are served by that server.

Assumptions and Threat-model. As in the case of FMD [5], both OMR2 and OMR3 rely only on a single untrusted server, whereas we make stronger assumptions as described above. In OMR, the authors present game-based proofs against a semi-honest adversary, whereas we present UC proofs.

Other properties. OMR achieves DoS resistance, where DoS attacks are defined as signals being pertinent for more than one receiver. Moreover, their protocols allow the receiver to determine the value ℓ which is the number of messages they expect to receive, and the recipients also get an explicit overflow message in the case the specified ℓ is less than the number of messages they actually receive.

	Privacy	Security	Recipient	Server	#Servers	Setup Assumptions
Naïve scan	full	–	$O(N)$	$\emptyset$	0	$\emptyset$
FMD [5]	k-anon	G,SH	$O(pN)$	$O(pM)$	1	$\emptyset$
OMR2 [19]	full	G, SH	$O(N+\log^2(\hat{k})\log(\varepsilon_n^{-1})+\hat{k}^3)$	$O(N(\log^2(\hat{k})+\log\varepsilon_p^{-1}))$	1	$\emptyset$
OMR3 [19]	full	G, SH	$O(\hat{k}\log(\hat{k})\log(\varepsilon_n^{-1})\log^4(N)+\hat{k}^3)$	$O(N(\log(\hat{k})\log(\varepsilon_n^{-1})\log^4(N)+\log\varepsilon_p^{-1}))$	1	$\emptyset$
Π_{TEE}	full	UC, M	$O(\bar{\ell})$	$O(\ell M)$	1	TEE
Π_{GC}	full	UC, SH	$O(\ell)$	$O(\ell M)$	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) [5]. For simplicity, we assume that each recipient has the same false positive rate p. ℓ denotes the maximum number of detectable incoming messages per each recipient and $\bar{\ell}$ denotes the actual number of messages that are sent to a recipient. From OMR [19] we have $\hat{k} = \tilde{O}(\bar{\ell} + \varepsilon_p N)$. Moreover, ε_p is a false positive rate and ε_n is a false negative rate. Finally the server computation is based on a single message received by the server(s).

Metadata-private messaging systems Previous works [29] and [16] describe dialing and add-friend protocols. These protocols enable one party to add another party as a friend and establish a connection with this friend such that an adversary cannot learn the friend's identity. This can be seen as a special case of the signaling problem, but these works do not consider the efficiency of the two parties that are involved. They require that all parties continuously send messages over the network (either cover traffic or actual protocol messages).

3 Our Approach to Build Private Signalling

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 for the recipient. We explain the general approach first, and then the 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. The recipients hire a Srv and register themselves with the server. (See Fig 1) The sender after posting a message to the board, sends a signal which is the encryption (under the pk of the server) of the the recipient's identity and the location of the message to the Srv through the board. The Srv maintains a table $\mathbb{T}$, with one row for each recipient. It decrypts the signal using its own secret key and adds the signal to the row of recipient. When a recipient R_i sends `RECEIVE` to the server, it simply responds with the corresponding 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.

Finally, note that each recipient might receives a different number of signals over time. To prevent leaking of this information, in our protocol, we fix the size of each recipients' row to be an upper bound ℓ, reflecting the signals recipients are expected to receive in a certain interval of time (e.g., per day, per-month, depending on the application).

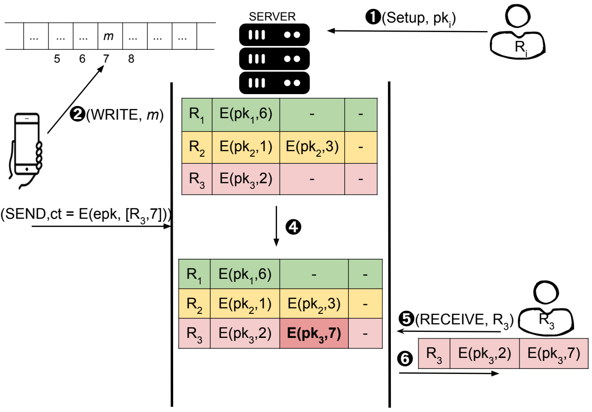
SERVER
❶(Setup, pk$_i$)
R_i
... ... m ... ...
5 6 7 8
❷(WRITE, m)
❸(SEND,ct = E(epk, [R$_3$,7]))
R$_1$ E(pk$_1$,6) - -
R$_2$ E(pk$_2$,1) E(pk$_2$,3) -
R$_3$ E(pk$_3$,2) - -
❹
R$_1$ E(pk$_1$,6) - -
R$_2$ E(pk$_2$,1) E(pk$_2$,3) -
R$_3$ E(pk$_3$,2) E(pk$_3$,7) -
❺(RECEIVE, R$_3$)
R$_3$
❻
R$_3$ E(pk$_3$,2) E(pk$_3$,7)

Figure 1: The no-privacy solution: ❶ Each recipient R_i registers with the Srv and sends its pk_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 Srv for the posted message. The signal is an encryption of $R_3, 7$ under the public key of the server. ❹ The server decrypts the signal using its secret key. The Srv then adds the encryption to the next available location in R_3's row. ❺ R_3 requests its row from the Srv in an authenticated way, and ❻, the Srv responds with the encryptions in that row.

TEE-based 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

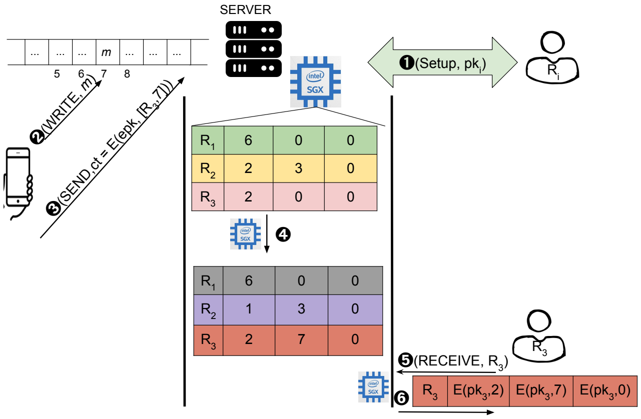

Figure 2: Single-server protocol. ❶ R_i securely communicates its pk_i with the enclave. ❷ ❸ The sender writes a message m for recipient R_3 on position 7 of the board and sends a signal (encryption of 7 under TEE's public key epk) to the Srv for the posted message. ❹ The enclave takes as input the encryption of the signal, decrypts it and updates column 2 in R_3's row with 7. ❺ R_3 requests for its row via the server to the enclave. ❻ If valid, the TEE releases R_3's row encrypted under the public key of R_3 via the server.

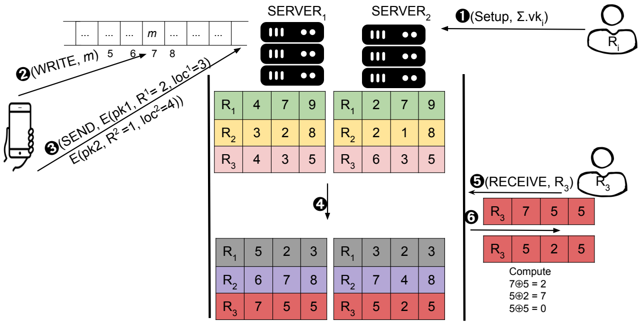

Figure 3: Two-server. ❶ R_i registers with the two servers. ❷ Sender writes message m for R_3 on position 7 of the board. ❸ Create shares of 3 e.g. $(2,1)$ and $7 = (3,4)$, send $(2,3)$ to Srv_1 and $(1,4)$ to Srv_2. ❹ Srv_1 and Srv_2 run a 2PC with inputs $(\mathbb{T}_1, 2, 3)$ and $(\mathbb{T}_2, 1, 4)$ and some fresh randomness and output new tables $\mathbb{T}_1$ and $\mathbb{T}_2$ such that in the next available position of R_3's row (column 2), is updated with fresh shares of 7, e.g. (5 and 2) 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 requests its row. ❻ If valid, Srv_1 sends [7,5,5] and Srv_2 sends [5,2,5]. R_3 reconstructs locations by computing $[7 \oplus 5, 5 \oplus 2, 5 \oplus 5] = [2, 7, 0]$.

that all computations inside the enclave are hidden from the server. With this tool in hand, the idea is that the recipients will first register with the server by providing their public key.

After this setup phase where recipients register with the enclave, the enclave maintains a vector of zeros for each user i. This is equivalent to initializing the table $\mathbb{T}$.

Each enclave will implement the following program: on input a signal ciphertext $\mathsf{ct}_{\mathsf{Signal}}$ and the table $\mathbb{T}$, first decrypt $\mathsf{ct}_{\mathsf{Signal}}$ with its secret key. If the decryption results in a valid plaintext location loc and recipient index i, then the enclave will update the row i of the table $\mathbb{T}$ with the loc in the next available position, and just re-write the other indexes. As in the solution with no privacy, a sender can communicate a signal for location loc to R_i, by simply encrypting the location and the recipient index under the enclave's public key pk, that is, $\mathsf{ct}_{\mathsf{Signal}} = \mathsf{Enc}(\mathsf{pk}, R_i||\mathsf{loc})$ and send $\mathsf{ct}_{\mathsf{Signal}}$ to the server. The server will then run the enclave on input $\mathsf{ct}_{\mathsf{Signal}}, \mathbb{T}$ 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 Sec 6, and the protocol is described in Fig 7. 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 et al. in [24] as the ideal functionality $\mathcal{G}_{\text{att}}$. Our proof is provided in the full version [20] and withstands malicious adversaries (for privacy). We present an illustration of this single-server approach in Fig 2. In the above approach we assumed that the enclave can store the table $\mathbb{T}$ in its internal memory. We note that this need not always be possible since the total space that is available in an enclave's (Intel SGX) internal memory is only 128MB. In the full version [20] we present a modification of the protocol where the $\mathbb{T}$ is stored by the server.

Limitations of Intel SGX: TEEs need to rely on a trusted authority (Intel in the case of Intel SGX), they are known to be prone to some side channel attacks [8] [14] and finally there are memory limitations [11]

Two-server Solution To accomplish the goal of obliviously updating the table of signals $\mathbb{T}$, we can use two servers Srv_1 and Srv_2 and have the table secret-shared among them. Srv_1 (resp., Srv_2) holds a table $\mathbb{T}_1$ (resp., $\mathbb{T}_2$) of strings that look random to Srv_1(resp., Srv_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 $\mathsf{Srv}_1, \mathsf{Srv}_2$ will update their tables by running a *secure computation protocol* (e.g., Yao's garbled circuits [7, 33]), 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.

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. We provide formal proofs of this work in the full version [20] 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.

4 Preliminaries

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.

Public Board: $\mathcal{G}_{\text{ledger}}$. We assume that all parties have read and write access to a public board, which we abstract via a public ledger ideal functionality $\mathcal{G}_{\text{ledger}}$ functionality introduced in [3]. $\mathcal{G}_{\text{ledger}}$ maintains a global variable called `state` and parties can read from and write to this global state through the commands `READ` and `SUBMIT`. An abridged version of $\mathcal{G}_{\text{ledger}}$ is presented in the App A.4, Fig. 18.

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 App A.

Trusted Execution Environment: $\mathcal{G}_{\text{att}}$. The TEE is modeled as a single, globally-shared ideal functionality that is denoted as $\mathcal{G}_{\text{att}}$ following the definition of [24]. The $\mathcal{G}_{\text{att}}$ functionality is depicted in Fig. 4 There are two types of invocations to the trusted hardware - *installation*, that allows to install a software and a *stateful resume*, that allows to execute it on an input. More details are provided in App A.3.

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

//initialization
On initialize: $(\mathsf{mpk}, \mathsf{msk}) := \Sigma.\mathsf{KeyGen}(1^\lambda), (\mathsf{epk}, \mathsf{esk}) := \mathsf{Enc}.\mathsf{KeyGen}(1^\lambda), T = \emptyset$
// public query interface:
On receive* getpk() respond with $(\mathsf{mpk}, \mathsf{epk})$

    Enclave operations

    //local interface – install an enclave:
    On receive* install$(idx, \mathsf{Prog})$ 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$(eid, \mathsf{inp})$ 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 $(\mathsf{outp}, \sigma)$ to $\mathcal{P}$.
```

Figure 4: Global functionality modeling SGX-like secure processor [24]

Garbled Circuits A garbling scheme $\mathcal{G}$ (described in App. A.2) consists of five polynomial time algorithms $(\mathsf{Garble}, \mathsf{Encode}, \mathsf{Eval}, \mathsf{Decode}, \mathsf{evaluate})$. Garble takes as input a function f and returns a garbled circuit F, encoding information e, and decoding information d. Encode takes e and an input x, and returns a garbled input X. Eval takes in the garbled circuit F and X, and returns a garbled output Y. Decode takes in the decoding information d and Y, and returns the plaintext output $y = f(x)$.

Oblivious Transfer In the oblivious transfer functionality (formally defined in App A.1, Fig 16), sender S has a pair of input strings s_0 and s_1 and a receiver R has a choice bit b. R learns only s_b while S learns nothing.

5 UC-Definition of Private Signaling

We define the problem of private signaling in the UC-framework [9]. 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).

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.
Sending a signal (SEND): Upon receiving $(\mathtt{SEND}, R_j, \mathsf{loc})$ from a sender S_i, send $(\mathtt{SEND}, S_i)$ to the adversary. Upon receiving $(\mathtt{SEND}, \mathtt{ok})$ from the adversary, append loc to $\mathbb{T}[R_j]$.
Retrieving signals (RECEIVE): Upon receiving $(\mathtt{RECEIVE})$ from some R_j, send $(\mathtt{RECEIVE}, R_j)$ to the adversary. Upon receiving $(\mathtt{RECEIVE}, \mathtt{ok})$ from the adversary, send $(\mathtt{RECEIVE}, \mathbb{T}[R_j])$ to the recipient R_j and update $\mathbb{T}[R_j] = []$

Figure 5: Private Signaling functionality

Private Signaling Ideal Functionality The functionality $\mathcal{F}_{\mathsf{privSignal}}$ provides the following interface - SEND and RECEIVE. The ideal functionality allows parties to send signals to a receiver, that informs that there exists a message at a particular location of the board maintained by an ideal ledger functionality $\mathcal{G}_{\mathsf{ledger}}$. To add a signal for a recipient R_j, a sender 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.

Corruption model We consider two settings. In protocol Π_{TEE}(Section 6) we consider a single-server with a TEE. Here we assume that the server can be malicious but the TEE is trusted. In protocol Π_{GC} (Section 7) we consider two-servers that do not collude. Furthermore, we allow any collusion between recipients and servers and prove that we achieve only privacy against malicious adversaries.

Communication model In our protocols we do not allow any out-of-band communication between the senders and the recipients. All entities have access to a global ledger functionality - $\mathcal{G}_{\mathsf{ledger}}$. We assume that recipients have direct channels with the Srv(s). We show in Section 8 that our protocols can be extended such that there are multiple instances of these servers that serve different recipients.

6 Private Signaling Protocol with TEE (Π_{TEE})

$\mathcal{G}_{\text{att}}$	Secure processor functionality Figure 4
$\vec{L}_i$	Encrypted locations for R_i
$\mathsf{ct}_{\mathsf{keys}}$	Encryption of encryption 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
$\mathsf{epk}, \mathsf{esk}$	Encryption keys of $\mathcal{G}_{\text{att}}$

Table 2: Notations for Π_{TEE}

The protocol Π_{TEE} assumes as hybrids a TEE functionality - $\mathcal{G}_{\text{att}}$ (defined in Sec 4). This TEE runs a program Prog that is defined in Fig 6. We only present Prog here. $\mathcal{G}_{\text{att}}$ attests (see Figure 4) to any computation that is done inside the processor and outputs a signed message to the server. As described in Figure 4 the $\mathcal{G}_{\text{att}}$ functionality generates a signing key and an encryption key, and any entity can securely query the functionality to get the signing verification key and the encryption public key.

The TEE maintains a vector $\vec{L}$ for each recipient. Each recipient R_i registers with the $\mathcal{G}_{\text{att}}$ functionality by sending it a "setup" command through the server where it communicates its public key and verification key to the $\mathcal{G}_{\text{att}}$ functionality. The program then initializes a vector $\vec{L}_i$ with all zeros corresponding to this recipient.

To send a signal to a recipient R_i, the sender encrypts the public key of the recipient along with the location of the message - $(\mathsf{pk}_i, \mathsf{loc})$, under the public key of the processor epkand submits this as a transaction to the $\mathcal{G}_{\text{ledger}}$ functionality. The Srv sends a READ command to the $\mathcal{G}_{\text{ledger}}$ functionality and retrieves the signals. To process these signals, the Srv inputs ("send", $\mathsf{ct}_{\mathsf{Signal}}$) to the TEE. The TEE decrypts $\mathsf{ct}_{\mathsf{Signal}}$ to get the public key pk_i of the recipient and location loc. The vector $\vec{L}_i$ for recipient R_i is updated with the loc and all other indices of $\vec{L}_i$ and all other $\vec{L}_j$ for $j \neq i$ are rewritten so that a malicious server that may observe memory access patterns cannot trivially learn the recipient of the signal.

```
On input* ("setup", $\mathsf{ct}_{\mathsf{keys}}$)
    Compute $(\mathsf{pk}, \Sigma.\mathsf{vk}) = \mathsf{Dec}(\mathsf{esk}, \mathsf{ct}_{\mathsf{keys}})$.
    Compute $\vec{L} = \{0\}_{j=0}^{\ell}$
    Set $\mathsf{index} = 0$ and $\mathsf{ctr} = 0$
    Initialize $\mathcal{T}[\mathsf{pk}] = (\vec{L}, \Sigma.\mathsf{vk}, \mathsf{index}, \mathsf{ctr})$
    return pk
On input* ("send", $\mathsf{ct}_{\mathsf{Signal}}$)
    $[\mathsf{msg}[0], \mathsf{msg}[1]] \leftarrow \mathsf{Dec}(\mathsf{esk}, \mathsf{ct}_{\mathsf{Signal}})$ and $\mathsf{msg}[0] = \mathsf{pk}$
    Read $\mathcal{T}[\mathsf{pk}] = (\vec{L}, \Sigma.\mathsf{vk}, \mathsf{index}, \mathsf{ctr})$
    Update $\mathsf{index} = (\mathsf{index} + 1) \mod \ell$,
    Update $\vec{L}[\mathsf{index}] = \mathsf{msg}[1]$
    Rewrite $\vec{L}[j]$ for $j \neq \mathsf{index}$
    Rewrite $\mathcal{T}[i]$ for all $i \neq \mathsf{pk}$
On input* ("receive", $(\mathsf{ctr}', \sigma)$)
    Read $\mathcal{T}[\mathsf{pk}] = (\vec{L}, \Sigma.\mathsf{vk}, \mathsf{index}, \mathsf{ctr})$
    if $\Sigma.\mathsf{Ver}(\Sigma.\mathsf{vk}, \mathsf{ctr}', \sigma) = 1$ and $\mathsf{ctr} = \mathsf{ctr}'$ then
        Let $(\mathsf{loc}_1 \ldots \mathsf{loc}_\ell) = \vec{L}$
        Compute $\vec{\mathsf{ct}}_{\mathsf{loc}} = \mathsf{Enc}_{\mathsf{pk}}(\mathsf{loc}_1), \ldots, \mathsf{Enc}_{\mathsf{pk}}(\mathsf{loc}_\ell)$
        Update $\mathsf{ctr} = \mathsf{ctr} + 1$, $\mathsf{index} = 0$ and $\vec{L} = \{0\}_{j=0}^{\ell}$
        return $(\vec{\mathsf{ct}}_{\mathsf{loc}})$
    else
        return $\perp$
```

Figure 6: Program $\mathsf{Prog}[\ell]$ run by $\mathcal{G}_{\text{att}}$

To receive its list of signals, a recipient R_i sends a signature along with a counter value denoted ctr. The TEE authenticates the recipient and checks that the counter value matches with the internally stored counter value of the recipient.

```
Enclave setup
1. Srv: Run $\mathcal{G}_{\text{att}}.\mathsf{install}(\mathsf{Prog}[\ell])$ to get $eid$.
Registration
Recipient $R_i$:
1. Let $(\mathsf{mpk}, \mathsf{epk}) := \mathcal{G}_{\text{att}}.\mathsf{getpk}()$
2. Compute $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Enc.KeyGen}(1^\lambda)$,
   $(\Sigma.\mathsf{sk}_i, \Sigma.\mathsf{vk}_i) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$.
3. Set $\mathsf{ct}_{\mathsf{keys},i} = \mathsf{Enc}(\mathsf{epk}, (\mathsf{pk}_i, \Sigma.\mathsf{vk}_i))$ and send
   ("setup", $\mathsf{ct}_{\mathsf{keys},i}$) to Srv.
4. Await $((eid, \mathsf{pk}_i), \sigma_{\mathcal{T}})$ from Srv.
5. Assert $\Sigma.\mathsf{Ver}_{\mathsf{mpk}}((eid, \mathsf{pk}_i), \sigma_{\mathcal{T}}) = 1$ and publish
   $\mathsf{pk}_i$. Initialize $\mathsf{ctr}_i = 0$.
Srv:
1. Upon receiving ("setup", $\mathsf{ct}_{\mathsf{keys},i}$) from $R_i$, let
   $((\mathsf{pk}_i), \sigma_{\mathcal{T}}) = \mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{"setup"}, \mathsf{ct}_{\mathsf{keys},i})$.
   Send $((eid, \mathsf{pk}_i), \sigma_{\mathcal{T}})$ to $R_i$.
Procedure (SEND, $R_i$, loc)
1. Sender $S$ gets $(\mathsf{mpk}, \mathsf{epk}) := \mathcal{G}_{\text{att}}.\mathsf{getpk}()$ and
   computes $\mathsf{ct}_{\mathsf{Signal}} = \mathsf{Enc}(\mathsf{epk}, [\mathsf{pk}_i, \mathsf{loc}])$ and sends
   (SUBMIT, (SEND, $\mathsf{ct}_{\mathsf{Signal}}$)) to $\mathcal{G}_{\text{ledger}}$.
2. Srv: Send READ to $\mathcal{G}_{\text{ledger}}$ and
   upon receiving (SEND, $\mathsf{ct}_{\mathsf{Signal}}$): Call
   $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{"send"}, \mathsf{ct}_{\mathsf{Signal}}))$.
Procedure RECEIVE
Recipient $R_i$:
1. Compute $\sigma_i = \mathsf{Sig}(\Sigma.\mathsf{sk}_i, \mathsf{ctr}_i)$ and
   send (RECEIVE, $\mathsf{ctr}_i, \sigma_i$) to Srv. Await
   $((eid, \vec{\mathsf{ct}}_{\mathsf{loc},i}), \sigma_{\mathcal{T}})$ from Srv
2. Assert $\Sigma.\mathsf{Ver}_{\mathsf{mpk}}((eid, \vec{\mathsf{ct}}_{\mathsf{loc},i}), \sigma_{\mathcal{T}}) = 1$
3.   Initialize $\mathsf{locns} = []$, $j = 0$
     while $(\mathsf{loc}_j = \mathsf{Dec}(\mathsf{sk}_i, \vec{\mathsf{ct}}_{\mathsf{loc}}[j])) \neq 0$ do
         $\mathsf{locns}.add(\mathsf{loc}_j)$
         $j = j + 1$
     $\mathsf{ctr}_i = \mathsf{ctr}_i + 1$
     return locns.
Srv:
1. Upon receiving (RECEIVE, $\mathsf{ctr}_i, \sigma_i$)
   from $R_i$, let $((eid, \vec{\mathsf{ct}}_{\mathsf{loc},i}), \sigma_{\mathcal{T}}) =$
   $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{"receive"}, \mathsf{ctr}_i, \sigma_i))$.
2. Send $((eid, \vec{\mathsf{ct}}_{\mathsf{loc},i}), \sigma_{\mathcal{T}})$ to $R_i$
```

Figure 7: The protocol for private signaling in the $\mathcal{G}_{\text{att}}$ hybrid world

If valid, the TEE returns encrypts the vector $\vec{L}_i$ under the public key of the recipient R_i and resets the vector $\vec{L}$ to a vector of all zeros and returns the list of encryptions. The recipient decrypts each ciphertext until it decrypts to a zero which indicates that the recipient has received all the messages. The server does the checks to ensure that every request to the

TEE is fresh and prevents a replay attack where a malicious server can simply send a previously received signature. This prevents the TEE from resetting the vector $\vec{L}$ that corresponds to an honest user.

Theorem 1. *Assume that the signature scheme Σ is existentially unforgeable under chosen message attacks, the encryption scheme* Enc *is CPA secure. Then the protocol* Π_{TEE} *in the* ($\mathcal{G}_{att}$, $\mathcal{G}_{\mathsf{ledger}}$)*-hybrid world UC-realizes the* $\mathcal{F}_{\mathsf{privSignal}}$ *functionality.*

Proof. (Sketch) To prove UC-security, we need to show that there exists a PPT simulator interacting with $\mathcal{F}_{\mathsf{privSignal}}$ that generates a transcript that is indistinguishable from the transcript generated in the real world where the adversary interacts with ($\mathcal{G}_{\mathsf{att}}$, $\mathcal{G}_{\mathsf{ledger}}$) ideal functionalities. The simulator internally simulates the $\mathcal{G}_{\mathsf{att}}$ and the $\mathcal{G}_{\mathsf{ledger}}$ functionalities to the adversary. We consider two cases of corruption here and in both cases we need to show that the simulator can simulate without learning the locations of honest recipients. We briefly describe the main idea in the simulation of the above mentioned corruption cases:

- *Sender and server are corrupt*: The simulator receives a $(\text{“send”}, \mathsf{ct}_{\mathsf{Signal}})$ command via the $\mathcal{G}_{\mathsf{att}}$ interface from the adversary. The simulator decrypts $\mathsf{ct}_{\mathsf{Signal}}$ using the secret key of the simulated TEE functionality esk and learns the recipient (R_i) and the location (loc). The simulator sends $(\mathtt{SEND}, R_i, \mathsf{loc})$ to the $\mathcal{F}_{\mathsf{privSignal}}$ ideal functionality on behalf of the adversary.
- *Receiver and server are corrupt*: The simulator receives a $(\text{“receive”}, \mathsf{ctr}, \sigma)$ command via the $\mathcal{G}_{\mathsf{att}}$ interface from the adversary. The simulator verifies the signature σ and sends RECEIVE to the $\mathcal{F}_{\mathsf{privSignal}}$ functionality on behalf of the adversary. The simulator receives a vector of locations that correspond to the adversary. The simulator encrypts these locations under the public key of the receiver and returns the vector of encryptions.

We defer the proofs to the full version of the paper [20]. □

7 Private Signaling with Two Servers Π_{GC}

Π_{GC} is run among two servers $\mathsf{Srv}_1, \mathsf{Srv}_2$. Each Srv_i for $i = 1, 2$ maintains a table (denoted by $\mathbb{T}^{(i)}$) that stores information on the signals. 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. The vector $\left[(\mathbb{T}^{(1)}[R][1] \oplus \mathbb{T}^{(2)}[R][1]), \ldots, (\mathbb{T}^{(1)}[R][\ell] \oplus \mathbb{T}^{(2)}[R][\ell])\right]$ represents a vector of locations that have been signaled by senders to recipient R. The servers also 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. Our protocol uses: the ideal oblivious transfer functionality $\mathcal{F}_{\mathsf{ot}}$, garbled circuits and EUF-CMA signatures (defined in Section 4 and Appendix A).

Registering with the servers: Each recipient R_i registers with the two servers by sending shares of vector of $\ell + 1$ zeros. This is achieved by sending $r_0 \ldots r_\ell$ to both the servers. The servers add a row to $\mathbb{T}^{(a)}$ and $\mathbb{L}^{(a)}$ - $\mathbb{T}^{(a)}[R] = [r_1 \ldots r_\ell]$ and $\mathbb{L}^{(a)}[R] = r_0$. 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 later.

$\mathbb{T}^{(i)}$	Table ($M \times \ell$) of locations maintained by Srv_i
$\mathbb{L}^{(i)}$	Table of available indices denoted index
$R^{(i)}$	Share of R received by Srv_i
$\mathsf{loc}^{(i)}$	Share of loc received by Srv_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 Π_{GC}

Sending a signal: The sender (denoted S) sending a signal to recipient R that a message exists for them at location loc does the following: Create shares of $\mathsf{pk}_R = \mathsf{pk}^{(1)}_R \oplus \mathsf{pk}^{(2)}_R$ and $\mathsf{loc} = \mathsf{loc}^{(1)} \oplus \mathsf{loc}^{(2)}$ and compute $\mathsf{ct}_{\mathsf{Signal},a} = \mathsf{Enc}(\mathsf{pk}_a, (\mathsf{pk}^{(1)}_R, \mathsf{loc}^{(\mathsf{a})}))$ where pk_a is the public key of Srv_a. The sender then submits these encryptions as transactions to the $\mathcal{G}_{\mathsf{ledger}}$ functionality. The servers periodically send READ commands to $\mathcal{G}_{\mathsf{ledger}}$ to learn the signals. They then decrypt the $\mathsf{ct}_{\mathsf{Signal},a}$ to receive $(\mathsf{pk}^{(1)}_R, \mathsf{loc}^{(\mathsf{a})})$. Since we assume that the servers do not collude, they do not learn any information about the recipient and the location. The two servers now run a 2PC Protocol processSignal that updates the tables according to the UpdateTable function. This function updates the tables maintained by the two servers in the following way: for the next available index (retrieved from the shares stored in tables $\mathbb{L}^{(1)}$ and $\mathbb{L}^{(2)}$) for receiver R store re-randomizations of the received shares and for every other index re-randomize the original shares. Since every index is updated, at the end of the protocol the two servers do not know which index was updated with the location, therefore hiding both the recipient's identity and the location of the signal. The UpdateTable also updates the tables $\mathbb{L}^{(1)}$ and $\mathbb{L}^{(2)}$ such that for receiver R, the tables store shares of an incremented index and for all other parties the shares are simply rerandomized.

Receiving a signal: To receive their vector of signals, the recipient sends a RECEIVE request to the two servers. This request includes a signature on freshly sampled random values that serve as new shares for the corresponding row on the table. Upon successful authentication, the servers send the corresponding table row to the receiver, who simply recombines the shares to receive their signals.

Theorem 2. *The protocol* Π_{GC} *UC-realizes the* $\mathcal{F}_{\mathsf{privSignal}}$

```
Setup
Srv$_a$, for $a \in \{1,2\}$:
  1. Generate encryption keys $(\mathsf{pk}_a, \mathsf{sk}_a) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and publish $\mathsf{pk}_a$
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. Sample $r_i \leftarrow_\$ \{0,1\}^\lambda$ for $i \in [0,\ell]$.
  3. Initialize $\mathsf{ctr}_i = 0$.
  4. Compute $\sigma_i = \Sigma.\mathsf{Sig}(\Sigma.\mathsf{sk}_i, ((r_0 \ldots, r_\ell), \mathsf{ctr}_i))$
  5. Send $(\textbf{Setup}, (r_0 \ldots, r_\ell), \mathsf{ctr}_i), \sigma_i)$ to $\mathsf{Srv}_i$.
Srv$_a$, for $a \in \{1,2\}$:
Upon receiving $(\textbf{Setup}, ((r_0 \ldots, r_\ell), \mathsf{ctr}_i), \sigma_i)$ from $R_i$:
  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 $\mathsf{ctr}_i$ and set $\mathbb{T}^{(a)}[R_i] = (r_1, \ldots, r_\ell)$ and $\mathbb{L}^{(a)}[R_i] = r_0$.
Procedure $(\mathtt{SEND}, R, \mathsf{loc})$
Sender $S$:
  1. Compute $R^{(1)}$ and $R^{(2)}$ s.t. $R = R^{(1)} \oplus R^{(2)}$.
  2. Compute $\mathsf{loc}^{(1)}$ and $\mathsf{loc}^{(2)}$ s.t. $\mathsf{loc} = \mathsf{loc}^{(1)} \oplus \mathsf{loc}^{(2)}$
  3. Compute $\mathsf{ct}_{\mathsf{Signal},a} = \mathsf{Enc}(\mathsf{pk}_a, \mathsf{Signal}_a)$, where $\mathsf{Signal}_a = (R^{(a)}, \mathsf{loc}^{(\mathsf{a})})$ for $a \in \{1,2\}$
  4. Send $(\mathtt{SUBMIT}, (\mathtt{SEND}, \mathsf{ct}_{\mathsf{Signal},1}, \mathsf{ct}_{\mathsf{Signal},2})$ to $\mathcal{G}_{\mathsf{ledger}}$.
Srv$_a$ for $a \in \{1,2\}$:
  1. Participate in protocol processSignal and update $(\mathbb{T}^{(1)}, \mathbb{L}^{(1)})$ and $(\mathbb{T}^{(2)}\ \mathbb{L}^{(2)})$ respectively.
Procedure RECEIVE
Recipient $R_i$:
  1. Sample $r_i \leftarrow_\$ \{0,1\}^\lambda$ for $i \in [0,\ell]$.
  2. Compute $\sigma_i = \Sigma.\mathsf{Sig}(\Sigma.\mathsf{sk}_i, (r_0, \ldots, r_\ell, a, \mathsf{ctr}_i))$ for $a \in \{1,2\}$.
  3. Send $(\mathsf{vk}_i, (r_0, \ldots, r_\ell, a, \mathsf{ctr}_i), \sigma_i)$ to $\mathsf{Srv}_a$
  4. Receive $\mathbb{T}^{(a)}[R_i]$ from $\mathsf{Srv}_a$
  5. Compute $\mathbb{T}^{(1)}[R][j] \oplus \mathbb{T}^{(2)}[R][j]$ until $\mathbb{T}^{(1)}[R][j] \oplus \mathbb{T}^{(2)}[R][j] = 0$.
  6. Update $\mathsf{ctr}_i = \mathsf{ctr}_i + 1$
Srv$_a$:
  1. Check if $\Sigma.\mathsf{Ver}(\Sigma.\mathsf{vk}_i, (\mathsf{ctr}', (r_0, \ldots, r_\ell)), \sigma_i) = 1$ and $\mathsf{ctr}' = \mathsf{ctr}_i$. Ignore if false.
  2. Else send $\mathbb{T}^{(a)}[R]$ to $R$.
  3. Update $\mathsf{ctr}_i = \mathsf{ctr}_i + 1$
```

Figure 8: Private signaling protocol with 2 servers

functionality in the $\mathcal{F}_{\mathsf{ot}}$-hybrid model assuming secure garbled circuits (Definition 1) and existential-unforgeable signature schemes.

Proof. (Sketch) To prove UC-security we need to show that there exists a PPT simulator interacting with $\mathcal{F}_{\mathsf{privSignal}}$ that generates a transcript that is indistinguishable from the real world where the adversary interacts with the $\mathcal{F}_{\mathsf{ot}}$ ideal functionality that is internally simulated by the simulator. We consider two cases of corruption:

- *Sender and* Srv_1 *are corrupt*: The simulator simulates Srv_2 and will receive shares $R^{(2)}$ and $\mathsf{loc}^{(2)}$ from the corrupt sender. It learns exact bits of $\mathsf{loc}^{(1)}$ and $R^{(1)}$ via the $\mathcal{F}_{\mathsf{ot}}$ functionality. The simulator computes $R = R^{(1)} \oplus R^2$ and $\mathsf{loc} = \mathsf{loc}^{(1)} \oplus \mathsf{loc}^{(2)}$ and sends $(\mathtt{SEND}, R, \mathsf{loc})$ to $\mathcal{F}_{\mathsf{privSignal}}$ on behalf of the corrupt sender.
- *Receiver and* Srv_1 *are corrupt*: When a corrupt R_i request its row, it must request both Srv_1 and Srv_2. The simulator then 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. Since the two servers maintain shares of 0, simply $\oplus$-ing R's row in $\mathbb{T}^{(2)}$ with the locations it received from the functionality gives the corrupt recipient its locations.

We defer the proofs to the full version of the paper [20]. □

```
Protocol processSignal
Srv$_a$ (where $a \in \{1,2\}$) upon sending READ to $\mathcal{G}_{\mathsf{ledger}}$ and receiving $\mathsf{ct}_{\mathsf{Signal},a}$. Decrypt to get $\mathsf{Signal}_a$.
  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 $\mathsf{Garble}(1^\lambda(\mathsf{UpdateTable})) \rightarrow (F, e, d)$, 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 $(\mathtt{OT\text{-}SEND}, (s_0, s_1))$ 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 $i \in [1,M], j \in [1,\ell]$, $\mathsf{loc}^{(\mathsf{a})}$, $R^{(a)}$.
Srv$_a$, upon receiving $(F,d)$ from the other server:
  1. (As evaluator of GC) Upon receiving OT-SEND from $\mathcal{F}_{\mathsf{ot}}$, send $(\mathtt{OT\text{-}RECEIVE}, b)$ 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 9: GC protocol to update two tables

```
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:     $\mathbb{L}^{(1)}[i] = \mathbb{L}^{(1)}[i] \oplus r^{(1)}_{(i)} \oplus r^{(2)}_{(i)}$
7:     for $j$ in $[1, \ell]$ do
8:         $\mathbb{T}^{(1)}[i][j] = \mathbb{T}^{(1)}[i][j] \oplus r^{(1)}_{(i,j)} \oplus r^{(2)}_{(i,j)}$
9: return $\mathbb{T}^{(1)}, \mathbb{L}^{(1)}$
```

Figure 10: The function to update the tables $\mathbb{T}^{(1)}$ and $\mathbb{L}^{(1)}$. The same algorithm updates the tables for Srv_2, except in step 4: the circuit updates $\mathbb{L}^{(2)}[R] = 0$

8 Extensions

Privately Fetching the Message from $\mathcal{G}_{\mathsf{ledger}}$. In this work we only focus on having the recipients privately learning the *location* on the ledger where a message was written for them. The problem of privately reading a block of interest from the board, is not the scope of this work. Luckily, however, there exist techniques from the literature that can be used to solve this problem. Furthermore, our protocols can be easily modified to privately fetch the message (instead of the location).

Privately Fetching using Existing Techniques: If a client could download the entire blockchain (ledger), privately reading is easily accomplished. The client will just use the secret location to read the relevant portions of the blockchain. However, this is not suitable for clients with small storage space, that are referred to as light clients. There is an extensive literature for privacy-preserving reads for light clients [17, 22, 25, 32] motivated by the problem of private cryptocurrency. In all of these works, the light client asks one (or more) powerful server(s) to learn its balance and other relevant information. To preserve the privacy of the light client, Qin et al. apply private information retrieval [25], Wüst et al. [32] employ TEEs, while Le et al. [17] use Oblivious RAM techniques. Crucially, the underlying assumptions of all such works is that the light clients (the recipient in our setting) *already know* the location of the blockchain that they wish to fetch. Privately communicating this location (without out-of-band communication, without the recipient being aware of the existence of the server) is what our protocols offers. Hence, our problem and techniques are complementary to the problem of light clients, and can be used in addition to the systems proposed in [17, 22, 32] so that a recipient privately learns these addresses (without having to communicate with the sender).

Modifying Private Signaling into Private Signaling & Fetching: Our current Π_{TEE} and Π_{GC} protocols can be modified to achieve a private READ functionality. The idea is to have the signal directly carry the message. Assuming that the messages are of fixed size, we modify the signal as follow. In Π_{TEE} the sender encrypts the message under the public key of the TEE and in Π_{GC} the sender creates two shares of the message and encrypts one share under the public key of one server, for both servers.

Supporting Multiple Servers As presented, our protocols assume that there is only one server (or a pair of servers) serving all the parties. In practice, there could exist several servers that offer the same service. To this end, we describe how our protocols can be extended so that they support multiple servers. First, we note that any entity (including the servers) must not map an encrypted signal to the corresponding server. This is necessary, otherwise we reduce the anonymity set of each signal to be the set of recipients served by that server. To this end, we use key-private encryptions [6]. Second, we note that if the server processes a signal for a recipient that it does not serve, the tables are simply re-randomized in the GC-based protocol (in the TEE-based protocol, the TEE does nothing) and this is oblivious to a server by design. Thus the sender just computes an encryption of the signal as before using a key-private encryption scheme and posts it on the board. All servers will attempt to process the signal and update their tables. By design, the table that stores signals for the recipient will be updated with the signal and all other tables will just be re-randomized.

9 Implementation and Evaluation

Implementation We used Intel SGX [23] to instantiate the $\mathcal{G}_{\mathrm{att}}$ functionality. RSA-OAEP [13] was used as the public key encryption scheme since it can be modified to make it key-private as was noted by Bellare et al. [6]. We benchmarked our schemes on an 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. For the garbled circuit protocol (processSignal) we use the compiler of Ball et al. [4].[3]. We used AES for the symmetric key encryption and SHA-256 for the hash functions (which is used in the OT protocol [2] that realizes the $\mathcal{F}_{\mathrm{ot}}$ functionality). All protocols are implemented in Rust.

Parameters. There are two parameters in our protocols - the

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

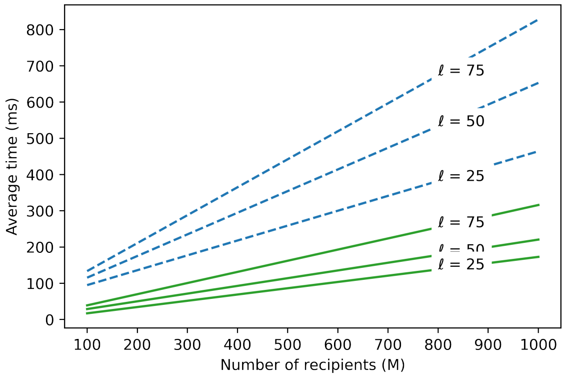

Figure 11: Comparing the average time taken to process a signal for the single-server protocols: Π_{TEE}(solid lines) and $\Pi_{\mathsf{TEE\text{-}ext}}$(dashed lines) by varying M from 100 to 1000 and ℓ between 25, 50 and 75.

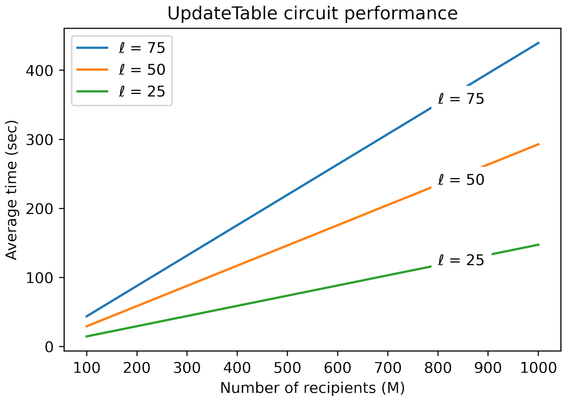

Figure 12: Evaluating the time taken to process signal when the value of $\ell \in \{25, 50, 75\}$ for Protocol Π_{GC} and varying M from 100 to 1000.

number of recipients M and an upper bound ℓ on the number of signals that a recipient is expected to receive in a certain interval of time. For instance, in a private-cryptocurrency application, recipients might expect to receive at most $\ell = 25$ private payments a week, and they will connect to the server(s) once a week to download their vector of signals.

In our measurements (see Figures 11 and 12) we choose to test for ℓ varying $25, 50$ and 75. This choice of parameters was inspired by applications of stealth payments in cryptocurrencies, and was informed by the following data we have at time of writing. The number of stealth payments [1] via Umbra [27] on the Ethereum blockchain was 416 transactions over a period of 5 months, which roughly amounts to a total (for all the recipients) of 20 private transactions per week. On the Zcash blockchain, the number of shielded (private) transactions between Oct 2016 to Jan 2018 was 6934 [15], which amounts to roughly total 140 transactions per week. In a system like Umbra, setting up $\ell = 20$ and having the recipients retrieve once a week is suitable. In a more active system such as Zcash, setting $\ell = 75$ (or $\ell = 25$ checking every day) might be more suitable.

We vary the number M of recipients from 100 to 1000. The computation complexity of the servers increases linearly with M (although in the TEE-based construction there are slight differences). Thus, one can simply extrapolate for any M. Furthermore, as we explained in Section 8, our protocols can support multiple servers that split the workload, while not splitting the anonymity set. Hence, if multiple servers are employed, the total number M can be split into smaller M_j - the set supported by server S_j, while the anonymity set is still M (the number of total recipients).

Evaluation

Server's Running Time. Fig. 11 and Fig. 12 show the time it takes for the server(s) to process *one* signal. This time is a function of M and ℓ, and for both protocols this time increases asymptotically linearly with M (for fixed values of ℓ). Concretely, for TEE-based protocols Π_{TEE} and $\Pi_{\mathsf{TEE\text{-}ext}}$, the graphs show that even for $M = 1000$ (with $\ell = 50$) it takes less than a second to process a signal.

For our two-server protocol - Π_{GC} (Fig. 12), the running time to process a signal is in the order of minutes. Though this protocol is not very useful for applications such as anonymous messaging where there are a lot of private messages, it may still practical be in systems like stealth payments assuming that the stealth transactions are distinguishable from the non-private ones (hence the servers do not need to process every transaction on the blockchain).

Latency for the recipients. For our protocols, we envision a setting where the servers process signals as soon as they are confirmed on the blockchain (in other words, our servers work in the background constantly rather than starting to work only after the recipients has asked to fetch its signals). This assumption is very natural for a server-client model. When the receiver connects to the server(s) it receives immediately every signal fetched by the server so far, up to the latest block of the blockchain that was processed by the servers. If the latest block processed by the servers is indeed the latest block that was posted to the blockchain, the recipients latency is 0. Our measurements of TEE-based protocols Π_{TEE} and $\Pi_{\mathsf{TEE\text{-}ext}}$ in Fig 11 shows how the running times vary as a function of M and ℓ and this information, in combination with information about frequency of the signals posted on the blockchain, can be used to evaluate the latency. These values depend on the specific application and the blockchain used. For concreteness (and following the analysis of OMR [19]), if we consider a blockchain such as Bitcoin, there are approximately 4000 transactions per block[4] (500K per day), and a block is confirmed, roughly, every 10 minutes. Not all 4000 transactions

[4] https://www.blockchain.com/charts/n-payments-per-block

	Baseline	Π_{TEE}	Π_{GC}	OMR [19]	FMD [5]
Server computation	NA	0.114s	146s	0.405s	0.001s
Recipient computation	0.012s	0.012s	0.001s	0.02s	2.1s
Total latency ($N = 500,000$)	100 min	0.012s	210 days	2.45 days	32s / 9.1hrs
Signal size	NA	64 bytes	128 bytes	68 bytes	956 bytes

Table 4: Computation cost and signal size comparisons of FMD [5], OMR [19] and protocols Π_{TEE} and Π_{GC}, assuming a recipient connect once a day. For Π_{TEE}, Π_{GC} and OMR the measurements are taken for $\ell = 50$. The total number of messages N is set as $500,000$ (approximate number of transactions in a day in Bitcoin) and for Π_{TEE} and Π_{GC}, $M = 500$. These numbers are take from OMR [19] and FMD [5] directly (FMD assumes a false positive rate ρ and we present latency for $\rho = 2^{-15}$ and 2^{-5}.

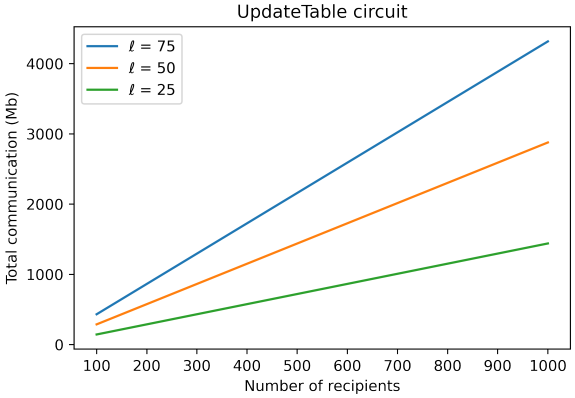

Figure 13: Evaluating the communication between the two servers when running the 2PC protocol in Π_{GC}. We vary M from 100 to 1000 and ℓ between $25, 50$ and 75.

will be private (or signals), however, combining these value with the running times shown in Fig. 11, we see that for a number of recipients M up to 700 (and for $\ell = 50$), the latency for a recipient is 0. This is because, the time to process a signal is 156.7ms, and hence will take about 10.44min to process all signals (assuming 4000) in block, which is approximately the time for a new block to be confirmed. Similarly for protocol $\Pi_{\mathsf{TEE\text{-}ext}}$ (where the encryptions are not stored in the TEE, but in the memory of the server) the overall computation time to process a signal for $M = 100$ and $\ell = 50$ is about 115.5ms, and hence will take 7.7 min to process all signals which is less than the time taken to confirm a block. Our two-server protocol - Π_{GC}, the average time taken to process a signal is in the order of minutes. Therefore this protocol would provide an acceptable latency in all the applications where the number of private signals in each block is limited.

Communication complexity. All signals are communicated to the servers via a blockchain (which we model in the paper as $\mathcal{G}_{\mathsf{ledger}}$). For Π_{TEE} the size of the signal is 64 bytes and for Π_{GC} it is 128 bytes. There is no communication overhead in Π_{TEE} at the server since the server simply forwards the signal to the TEE, whereas in Π_{GC} the two servers need to run a 2PC protocol and scales with both M and ℓ. As can be seen from Fig 13 Finally, for a `RECEIVE` command, the communication is an authentication from the recipient and the corresponding row from the servers. In Π_{TEE} this authentication is 512 bytes and the server returns ℓ encryptions, each of size 512 bytes. In Π_{GC} the recipient sends an authenticating message of size 512 bytes and receives two vectors of size ℓ from each server.

Optimizations. To improve the performance of Π_{GC} we modify the UpdateTable function to process K signals in the same GC protocol. We observe that after a point $K = 20$ (see Figure 15), there is no significant improvement in the normalized time taken to process a signal. In Figure 14 we observe that for $K = 5$ and $K = 10$ the overall computation improved by 2.33× and 3.95× respectively. This gain in overall processing time can be attributed to lesser number of garbled tables that need to be computed and communicated.

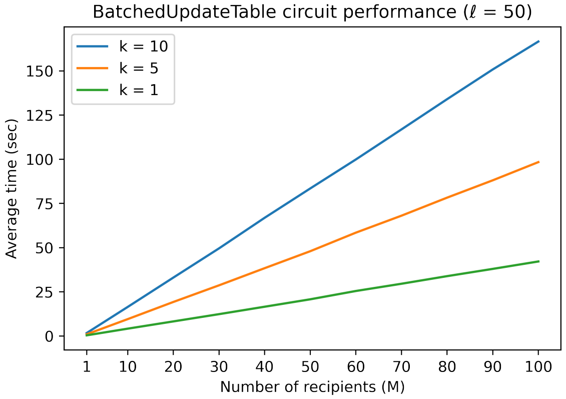

Figure 14: Evaluating the improvement in computation time to protocol Π_{GC} when signals are batched in groups of 5 and 10. We vary the number of recipients from 1 to 100 here.

Availability. The source code of our implementations of our protocols can be found at `https://github.com/anon-submission-1100/pps`.

Comparison with related work and baseline. We compare our protocols with the most related work FMD [5] and OMR [19] (discussed in Sec. 2.2) and with the *baseline* "naive" solution [5] in Table 4. Recall that in our approach the running

[5] The baseline naive protocol is the one where a recipient simply attempts

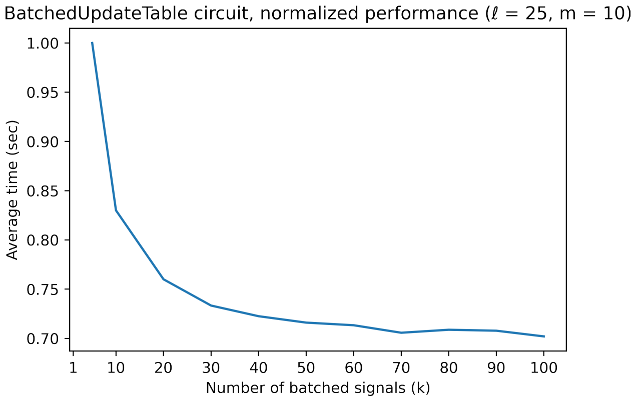

Figure 15: Effect of batching fixing $\ell = 25$ and $M = 10$

time per signal depends on parameters M and ℓ, and recall that we assume that the servers continuously process signals (regardless of whether the recipients is retrieving or not) as they appear on the ledger. In contrast, in OMR [19], the running time *per signal* depends on N (the *total* number of signals posted *ever*) and the upper bound ℓ of actual signals the receiver expects to receive (in their construction ℓ might change for each recipient). Furthermore, in OMR, the computation is done *on demand, per-recipient*, namely, when recipient asks to retrieve. Hence, even assuming that all requests are served in parallel, a recipient querying the server at time δ, will need to wait until the server process all the signals posted since the beginning of time until time δ.

In FMD, the server processes signals continuously, like ours, and the receiver obtains a list of signals. However, depending on the *privacy* parameter chosen by the receiver, this list can be as large as N. Indeed the privacy parameter ρ corresponds to a false positive rate. The higher the false positive rate, the higher is the computation expected for the recipient (and hence the latency). For instance for $\rho = 2^{-15}$, the error rate is very small and the overhead for the recipient is minimal, however, the anonymity achieved is $\rho \times N$, which is only 15 for $N = 500,000$. On the other hand for $\rho = 2^{-5}$ the recipient gets an anonymity set of 15625 but takes up to 9 hours to detect its signals.

Due to these crucial differences, comparing with these approaches (and especially with OMR) is somewhat application dependent, since the choice of M, ℓ, N, and whether the servers process signals *continuously* or whether they start to compute upon request, generates significant differences in the performances.

For the comparison, in Table 4 we considered settings and parameters used in OMR and FMD, and compared with the baseline solution. We considered a scenario where a recipient connects once a day; the number of total signals is $N = 500,000$ and the number of expected signals per recipient is $\ell = 50$. We set $M = 500$ for our protocols, and for FMD we considered error/privacy rate at both side of the spectrum (i.e., $\rho - 2^{-5}$ and $\rho - 2^{-15}$).

We found that our TEE-based solutions is the fastest. Our GC-based solution, on the other hand, takes a long time to process $500,000$ signals. However, we stress that the processing time of the servers does not impact the *running time* of the recipient that is still constant (regardless of M, ℓ, N). This running time impacts the latency, and suggest that GC-based solution should be used for applications with a lower volume of private signal per-day, and we are interested in saving computation and on-line time for the recipients.

Cross-over point with baseline solution. Comparison with baseline is relevant when the recipient connects at fixed interval to download and process the messages in bulk [6]. In this case, given an instantiation of the ledger, one can analyse the cross-over point where our solutions provide less latency than the baseline solution. For our TEE-based solution, assuming the ledger is implemented with bitcoin the cross-over point is 10 blocks. This is because it takes 480s to decrypt 10 blocks (assuming 0.012s to decrypt a signal, 4000 signals per block we have 10*4000*0.012). In our TEE-based solution, the server takes 7.5 min per block (paramters $M = 500, \ell = 50$) and is continuously working. At the 10th block as well, the server take 7.5 min to process the block. But a receiver coming online then will take 8 min and this is the crossover point.

10 Conclusion and Open Problems

We have introduced the problem of *private signaling* that abstracts several real-world recipient-anonymous applications. We provide 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 achieve 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.

to decrypt every transaction on the board, after having downloaded it (for instance, once a day).

[6] If the recipients were always on-line and decrypting each signal, then no server-aided solution could beat this rate.

References

[1] Address 0xfb2dc580eed955b528407b4d36ffafe3da685401 | etherscan.

[2] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions. *Journal of Cryptology*, 30(3):805–858, 2017.

[3] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *Annual international cryptology conference*, pages 324–356. Springer, 2017.

[4] 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.

[5] Gabrielle Beck, Julia Len, Ian Miers, and Matthew Green. Fuzzy message detection. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1507–1528, 2021.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

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

[13] 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.

[14] 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.

[15] George Kappos, Haaroon Yousaf, Mary Maller, and Sarah Meiklejohn. An empirical analysis of anonymity in zcash. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 463–477, 2018.

[16] David Lazar and Nickolai Zeldovich. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 571–586, 2016.

[17] 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.

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

[19] Zeyu Liu and Eran Tromer. Oblivious messsage retrieval. *Cryptology ePrint Archive*, 2021.

[20] Varun Madathil, Alessandra Scafuro, István András Seres, Omer Shlomovits, and Denis Varlakov. Private signaling. *Cryptology ePrint Archive*, 2021.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

[27] ScopeLift. Scopelift/umbra-protocol: privacy preserving shielded payments on the ethereum blockchain.

[28] István András Seres, Balázs Pejó, and Péter Burcsi. The effect of false positives: Why fuzzy message detection leads to fuzzy privacy guarantees? *arXiv preprint arXiv:2109.06576*, 2021.

[29] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152, 2015.

[30] 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.

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

[32] 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.

[33] 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 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 $\mathcal{F}_{\text{ot}}$ in Figure 16.

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

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

Figure 16: Ideal functionality for oblivious transfer

A.2 Garbled circuits

We present a formal definition for garbled circuits. We present the definitions of [7].

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

1. $\mathsf{Garble}(1^\lambda, f) \to (F, e, d)$. *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. $\mathsf{Encode}(e, x) \to X$. *The encoding algorithm* Encode *takes in the encoding information e and an input x, and returns a garbled input X.*
3. $\mathsf{Eval}(F, X) \to Y$. *The evaluation algorithm* Eval *takes in the garbled circuit F and the garbled input X, and returns a garbled output Y.*
4. $\mathsf{Decode}(d, Y) \to y$. *The decoding algorithm* Decode *takes in the decoding information d and the garbled output Y, and returns the plaintext output y.*
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 17.

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.

```
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 17: The $\mathsf{PrvSim}_{\mathcal{G},\phi,\mathcal{S}}$ game

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 $X = (X_i^{v_i})_{i \in \mathsf{indices}}$.

A.3 Attested Execution Processers

In this section we present more details on the formalization of attested execution processers as described in [24]

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 Srv 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.

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.

- Upon receiving $(\mathtt{SUBMIT}, \mathsf{tx})$ from a party P_i:
  1. Choose a unique transaction ID txid and set $\mathtt{BTX} := (\mathsf{tx}, \mathsf{txid}, \tau_L, P_i)$
  2. If $\mathsf{Validate}(\mathtt{BTX}, \mathtt{state}, \mathtt{buffer}) = 1$ then $\mathtt{buffer} := \mathtt{buffer} \cup \mathtt{BTX}$
  3. Send $(\mathtt{SUBMIT}, \mathtt{BTX})$ to $\mathcal{A}$.
- Upon receiving READ from a party P_i, send $\mathtt{state}_{P_i}$ to P_i. If received from $\mathcal{A}$, send $(\mathtt{state}, \mathtt{buffer})$ to $\mathcal{A}$.

Figure 18: Abridged $\mathcal{G}_{\mathsf{ledger}}$ functionality

A.4 Ledger functionality

In our protocols we model the public board for reads and writes in the form of a $\mathcal{G}_{\mathsf{ledger}}$ ideal functionality presented here. We present an abridged version of the functionality where we present the READ and SUBMIT commands. For the complete description of the functionality, we refer the reader to Pages 339-340 of [3].

B Protocol $\Pi_{\mathsf{TEE\text{-}ext}}$

Protocol Overview The protocol $\Pi_{\mathsf{TEE\text{-}ext}}$ is the same as Π_{TEE}, except that the TEE does not store the encryptions inside its internal memory but stores it on the server and requires the server to send the table to the TEE to process the signals.

```
On input ("setup", $\mathsf{ct_{keys}}$)
    Compute $(\mathsf{pk}, \Sigma.\mathsf{vk}) = \mathsf{Dec}(\mathsf{esk}, \mathsf{ct_{keys}})$.
    Compute $\vec{L} = \{\mathsf{Enc}(\mathsf{epk}, \mathsf{pk}\|0)\}_{j=0}^{\ell}$
    Set $\mathsf{index} = 0$ and $\mathsf{ctr} = 0$
    return pk, store $\vec{L}$ in external memory and $(\mathsf{pk}, \Sigma.\mathsf{vk}, \mathsf{index}, \mathsf{ctr})$ in internal memory.
On input* ("send", $i, \vec{L}_i, \mathsf{ct_{Signal}}$)
    Read $\mathsf{index}_i, \mathsf{pk}_i$ from internal memory corresponding to $i$.
    Let $\mathsf{msg} = \mathsf{Dec}(\mathsf{esk}, \mathsf{ct_{Signal}})$.
    if $\mathsf{msg}[0] = \mathsf{pk}_i$ then
        Update $\mathsf{index} = (\mathsf{index} + 1) \mod \ell$
        for $j$ in $[1, \ell]$ do
            Let $\mathsf{curr} = \mathsf{Dec}(\mathsf{esk}, \vec{L}[j])$
            if $j = \mathsf{index}$ then
                $\vec{L}[j] = \mathsf{Enc}(\mathsf{epk}, \mathsf{msg}[0]\|\mathsf{msg}[1])$
            else
                $\vec{L}[j] = \mathsf{Enc}(\mathsf{epk}, \mathsf{curr})$
    else
        for $j$ in $[1, \ell]$ do
            $\vec{L}[j] = \mathsf{Enc}(\mathsf{epk}, \mathsf{Dec}(\mathsf{esk}, \vec{L}[j]))$
    return $\vec{L}$
On input* ("receive", $\mathsf{ctr}, \sigma, \vec{L}, j$)
    if $\Sigma.\mathsf{Ver}(\Sigma.\mathsf{vk}_j, \mathsf{ctr}', \sigma) = 1$ and $\mathsf{ctr} = \mathsf{ctr}'$ and $\forall i, \mathsf{Dec}(\mathsf{esk}, \vec{L}[i][0:\lambda] = \mathsf{pk}_j)$ then
        Compute $\mathsf{loc}_i = \mathsf{Dec}(\mathsf{esk}, \vec{L}[i])$ for $i \in [1, \ell]$
        Compute $\vec{\mathsf{ct}}_{\mathsf{loc}} = \mathsf{Enc}_{\mathsf{pk}_j}(\mathsf{loc}_1), \ldots, \mathsf{Enc}_{\mathsf{pk}}(\mathsf{loc}_\ell)$
        Update $\mathsf{ctr} = \mathsf{ctr} + 1$ and $\mathsf{index} = 0$
        Update $\vec{L} = \{\mathsf{Enc}(\mathsf{epk}, \mathsf{pk}_j\|0)\}_{k=0}^{\ell}$
        return $(\vec{L}, \vec{\mathsf{ct}}_{\mathsf{loc}})$
    else
        return $\perp$
```

Figure 19: Program Prog run by $\mathcal{G}_{\mathrm{att}}$

```
Enclave setup
1. Run $\mathcal{G}_{\mathrm{att}}.\mathsf{install}(\mathsf{Prog})$ to get $eid$.
Setup
Recipient $R_i$:
1. 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 $\mathsf{ct}_{\mathsf{keys},i} = \mathsf{Enc}(\mathsf{epk}, (\mathsf{pk}_i, \Sigma.\mathsf{vk}_i))$ and send ("setup", $\mathsf{ct}_{\mathsf{keys},i}$) to Srv, and await $((eid, \mathsf{pk}_i), \sigma)$ from Srv. Assert $\Sigma.\mathsf{Ver}_{\mathsf{mpk}}((eid, \mathsf{pk}_i), \sigma) = 1$ and publish $\mathsf{pk}_i$. Initialize $\mathsf{ctr}_i = 0$.
Srv:
1. Upon receiving ("setup", $\mathsf{ct}_{\mathsf{keys},i}$) from $R_i$, let $((\mathsf{pk}_i, \vec{L}_i), \sigma) = \mathcal{G}_{\mathrm{att}}.\mathsf{resume}(eid, (\text{"setup"}, \mathsf{ct}_{\mathsf{keys},i})$. Send $((eid, \mathsf{pk}_i, \vec{L}_i), \sigma)$ to $R_i$.
Procedure $(\mathtt{SEND}, R_i, \mathsf{loc})$
1. Sender $S$ gets $(\mathsf{mpk}, \mathsf{epk}) := \mathcal{G}_{\mathrm{att}}.\mathsf{getpk}()$ and computes $\mathsf{ct_{Signal}} = \mathsf{Enc}(\mathsf{epk}, [\mathsf{pk}_i, \mathsf{loc}])$ and sends $(\mathtt{SUBMIT}, (\mathtt{SEND}, \mathsf{ct_{Signal}}))$ to $\mathcal{G}_{\mathrm{ledger}}$.
2. Srv: Upon receiving $(\mathtt{SEND}, \mathsf{ct_{Signal}})$ from $\mathcal{G}_{\mathrm{ledger}}$ after sending READ: For $j \in [1, M]$, call $\mathcal{G}_{\mathrm{att}}.\mathsf{resume}(eid, (\text{"send"}, i, \vec{L}_j, \mathsf{ct_{Signal}}))$ 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 $(\mathtt{RECEIVE}, \mathsf{ctr}_i, \sigma_i)$ to Srv. Await $((eid, \vec{L}_i, \vec{\mathsf{ct}}_{\mathsf{loc},i}), \sigma_T)$ from Srv
2. Assert $\Sigma.\mathsf{Ver}_{\mathsf{mpk}}((eid, \vec{L}_i, \vec{\mathsf{ct}}_{\mathsf{loc},i}), \sigma_T) = 1$
3.  Initialize $\mathsf{locns} = [], j = 0$
    while $(\mathsf{loc}_j = \mathsf{Dec}(\mathsf{sk}_i, \vec{\mathsf{ct}}_{\mathsf{loc}}[j])) \neq \mathsf{pk}\|0$ do
        $\mathsf{locns}.add(\mathsf{loc}_j)$
        $j = j + 1$
    return locns.
Srv:
1. Upon receiving $(\mathtt{RECEIVE}, \mathsf{ctr}_i, \sigma_i)$ from $R_i$, let $((eid, \vec{L}_i, \vec{\mathsf{ct}}_{\mathsf{loc},i}), \sigma_T) = \mathcal{G}_{\mathrm{att}}.\mathsf{resume}(eid, (\text{"receive"}, \mathsf{ctr}_i, \sigma_i, \vec{L}_i))$.
2. Send $((eid, \vec{L}_i, \vec{\mathsf{ct}}_{\mathsf{loc},i}), \sigma_T)$ to $R_i$ and update $\vec{L}_i$
Procedure (READ) Send READ to $\mathcal{G}_{\mathrm{ledger}}$ and receive state.
```

Figure 20: The protocol for private signaling in the $\mathcal{G}_{\mathrm{att}}$ hybrid world