Scalable Private Signaling

Sashidhar Jakkamsetti[1], Zeyu Liu[2], and Varun Madathil[3]

[1]Bosch Research
[2]Yale University
[3]North Carolina State University

March 22, 2024

Private messaging systems that use a bulletin board, like privacy-preserving blockchains, have been a popular topic during the last couple of years. In these systems, a private message is typically posted on the board for a recipient and the privacy requirement is that no one can determine the sender and recipient of the message. Until recently, the efficiency of these recipients was not considered, and the party had to perform a naive scan of the board to retrieve their messages.

More recently, works like Fuzzy Message Detection (FMD), Private Signaling (PS), and Oblivious Message Retrieval (OMR) have studied the problem of protecting recipient privacy by outsourcing the message retrieval process to an untrusted server. However, FMD only provides limited privacy guarantees, and PS and OMR greatly lack scalability.

In this work, we present a new construction for private signaling which is both asymptotically superior and concretely *orders of magnitude* faster than all prior work while providing full privacy. Our construction makes use of a trusted execution environment (TEE) and an Oblivious RAM (ORAM) to improve the computation complexity of the server. We also improve the privacy guarantees by keeping the recipient hidden even during the retrieval of signals from the server. Furthermore, we implement a side-channel resistant prototype and show that for a server with a million recipients and ten million messages, the prototype takes less than 70 milliseconds to process a sent message and less than 6 seconds to process a retrieval request (for 100 signals) from a recipient.

1 Introduction

The protection of message contents in messaging applications is well-studied, and end-to-end encryption is nowadays widely practiced. Protecting the metadata, such as sender and recipient identity, is essential in anonymous message delivery systems [1, 2, 3, 4, 5], especially when messages are posted on a publicly visible blockchain or bulletin board. This model of anonymous communication using a bulletin board is not new, in fact, stealth transactions [6] and privacy-preserving transactions[7, 8, 9, 10] both work in the same model. Anonymous messaging services such as Riposte[2], also deploy a bulletin board for parties to post their messages.

However, despite the importance, there is room for improvement in recipient efficiency. Concerning private blockchain transactions, it seems that there is a tradeoff between recipient privacy and efficiency. Namely, if all messages are encrypted on the bulletin board, the only means for a recipient to find their messages would be to try and decrypt each ciphertext on the board. An approach that is popular with stealth addresses is to outsource the search for transactions to a server that attempts to decrypt each transaction with a different viewing key and notify the recipient when a transaction is posted on the chain for them. In this case, the recipient naturally loses all privacy to the server. Thus there is a trade-off between the efficiency and privacy of the receiver in these systems.

Inspired by the privacy-preserving blockchain light client, like for Zcash [9] and Monero [7], recently, there have been efforts to improve the privacy of the recipients while maintaining the efficiency of recipients by using an untrusted server (i.e., users with limited resources can outsource the job of fetching messages to the servers while maintaining privacy).

Fuzzy Message Detection [5] by Beck et al. was the first work to address this. The authors propose a decoy-based approach: the server detects messages including both the real messages addressed to the recipient as well as some randomly chosen false positives. But in this work, there is an implicit trade-off between efficiency and privacy that a recipient can hope to achieve.

Two follow-up works by Madathil et al[11] and Liu and Tromer [12] improved this to entirely hide the set of pertinent messages (i.e., the message addressed to the retrieving recipient) without affecting the efficiency of the receiver. Madathil et. al. [11] define the problem of *private signaling* where a sender sends a signal to a recipient that a message exists on the board through an untrusted server. They propose two solutions, one based on TEE and one based on two non-colluding servers running a garbled-circuit-based protocol. We will describe this construction in more detail in Section 2. Liu and Tromer [12] independently solve a similar problem and called it *oblivious message retrieval*, and propose a solution where a server is able to obliviously detect and retrieve messages for recipients based on leveled homomorphic encryption. The two primitives are built on existing systems like Zcash [13] to help recipients outsource their retrieval jobs to untrusted servers (i.e., the servers read the public bulletin board and send the pertinent messages to the recipient without learning which are the pertinent messages).

Although both works have made great progress in proposing schemes providing full privacy, while preserving efficiency for the recipient, they still lack scalability. For a board with N messages, a server serving M recipients needs to spend $\Omega(N \cdot M)$ work to allocate the messages in a privacy-preserving way. Thus, when N and M are huge, the computational costs are very high.

The focus of this work is on the following natural question:

> *Can we have a scheme that is fully private and the runtime of the server scales sublinear in M and (almost) linear in N?*

1.1 Contributions

In this paper, we firmly answer *yes* to this question.

- **Scalable PS construction.** We propose a private signaling construction such that for N signals and M recipients, a server only needs to do $O(N(\log(M) + \log(N)))$ work to find the signals addressed to each recipient *with full privacy guarantee*.
  - Concretely, with practical parameters (e.g., 10^7 messages and 10^6 recipients), the total cost is only < 70 milliseconds to process a sent signal and < 6 seconds to do a retrieval for a recipient (with 100 signals per retrieval). These are orders of magnitude faster than prior works (the fastest prior work [11] requires > 200 milliseconds to process a sent signal with only 10^3 recipients, each having at most 75 signals; a retrieval of 75 messages takes > 400 seconds). A similar level of advantage is maintained compared to Oblivious Message Retrieval constructions [12, 14].
  - Moreover, our protocol achieves a stronger correctness functionality compared to prior schemes. Schemes in [11] suffer from a simple message spamming attack we describe in Section 2. The scheme in [12] outputs $\mathsf{overflow}$ when there are more than $\bar{\ell}$ messages for a recipient (where $\bar{\ell}$ is some upper bound on the number of messages that a server retrieves for a receiver [1]). However, our scheme does not have any of the issues and thus provides a stronger correctness guarantee.
  - We also show that our construction can be extended to the multi-server setting with essentially the same efficiency without sacrificing any privacy (not achieved by [11]).
- **Strengthened Private Signaling problem.** We present a stronger private signaling functionality than the one introduced in [11], by requiring that the server cannot link two retrieval requests. This

[1] When this happens, the recipient needs to either download the board themselves or re-outsource the job with a larger $\bar{\ell}$. Either way, it introduces a large overhead.

property is denoted as detection key unlinkability in [12]. We define this functionality in the UC framework[15] and show that our protocol UC-realizes it.

- **Open-sourced implementation with side-channel resistance.** We implement and evaluate our protocol, and our implementation is (anonymously) open-sourced at [16]. We observed that our prototype is significantly more performant and scalable than all prior work. Furthermore, to mitigate side-channel attacks against TEE, we build our prototype on top of ZeroTrace [17] – a side-channel resistant ORAM library for Intel SGX.

2 Technical Overview

In this section, we give an overview of the main technique in our construction of an efficient private signaling protocol.

Original protocol review. We start by reviewing the recent TEE-based protocol of Madathil et al[11].

In their protocol, the TEE stores a mapping of parties and their pertinent signals (i.e., the signals addressed to the corresponding party). Since this mapping is stored inside the TEE, all information about the signals is private. In more detail, their construction works as follows: The TEE stores a table T of size $M \times \bar{\ell}$, where M is the total number of participants and $\bar{\ell}$ is the total number of signals the TEE stores per recipient.

To send a signal, a sender computes an encryption of the recipient's identity and the location on the board that the sender wants to communicate to the recipient. The encryption is done using the public key of the TEE. This ciphertext constitutes the signal and is sent to the server running the TEE. The server runs the TEE with this ciphertext as input; the TEE decrypts the ciphertext and appends the location to the recipient's row in the table T.

At any point in time, a party can choose to retrieve its locations by sending an authenticated message to the server. The server verifies the signature and inputs this authenticated retrieval request to the TEE. Upon verification, TEE encrypts all the locations in the corresponding row of the recipient under their public key and outputs it to the server, which then forwards it to the recipient. Upon every retrieval, the TEE flushes the row of the retrieving recipient so they can receive the next set of signals. Privacy is guaranteed since all the signals are stored inside the TEE, and upon retrieval, they are all encrypted under the public key of the recipient. Therefore no entity can link either the senders or the locations of messages on the blockchain to a recipient.

In addition, to mitigate certain side-channel attacks, [11] proposes to update the entire table after each signal arrives. Thus, for each signal sent to TEE, the TEE runtime is $O(M\bar{\ell})$. TEE local storage is also $O(M\bar{\ell})$. Both of these limit the efficiency and scalability of the original protocol.

An inherent issue with [11]. Despite this being a secure solution, the work of [11] has a subtle correctness issue. Recall that the constructions presented in [11] only allow for storage of up to $\bar{\ell}$ signals per recipient. Now, what would happen if a particular recipient were to receive more than $\bar{\ell}$ signals before they retrieve their signals from the TEE?

In an earlier version of the same work by Madathil et. al. [18]: the authors suggest that the older signals would be overwritten upon receipt of new signals that have not been retrieved. In the newer version [11], the authors do not deal with this directly, but assume that the recipient retrieves the signals frequently enough, such that there are at most $\bar{\ell}$ signals between two retrievals.

Both of these are clearly unsuitable for cases when a recipient receives more than $\bar{\ell}$ messages in a short period of time. Note that one could set $\bar{\ell}$ to be a very large number, but the size of $\bar{\ell}$ will be constrained by the runtime and the storage space of the TEE since they are both linear in $\bar{\ell}$.

Moreover, their designs could actually lead to a message spamming attack where an adversary simply sends $\bar{\ell}$ dummy signals to the recipient via the TEE. This would ensure that an honest recipient cannot retrieve its actual signals: the dummy signals would overwrite all the honest ones.

Our approach in a nutshell. In this work we show how we can improve the scalability of the

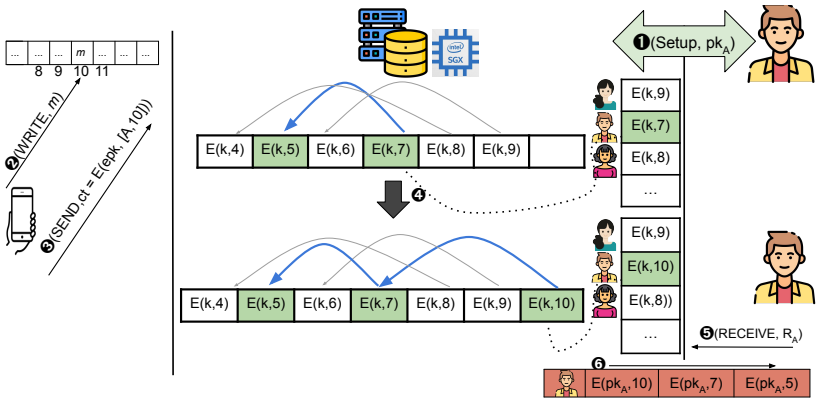


Figure 1: Overview: ❶ Party A registers with his public key pk_A with the TEE. ❷ A sender writes a message m at position 10 on the board for party A. ❸ The sender then sends a signal $(\mathsf{Enc}(\mathsf{epk}, [A, 10]))$ to the server. ❹ The TEE maintains a linked list of encrypted signals and a table with the latest signals of each recipient on the server. The TEE then appends an encryption of 10 and adds a link to party A's previous signal which is an encryption of 7. ❺ The party A requests its signals from the TEE. ❻ The TEE retrieves all the signals by iterating through the linked list (via ORAM) which is 10, 7, and 5, and returns them encrypted under A's public key.

TEE-based private signaling protocol of [11], and ensure that the above-mentioned issue with $\bar{\ell}$ is mitigated.

We present a construction that maintains the signals pertinent to a recipient in the form of a *linked list*. Moreover, these linked lists are not stored inside the TEE but with the server that runs the TEE. Since there are no storage constraints on the server, these linked lists can be of arbitrary length. To prevent the server from learning which signals are retrieved, the signals are stored in an Oblivious RAM[19, 20, 21].

We describe below our ideas of changing data structure in Section 2.1, an overview of the ORAM-based construction in Section 2.2, and an overview of our new retrieval mechanism in Section 2.3.

2.1 Changing data structure to a linked list

In this subsection, we will first present how to mitigate the issue of the server's ability to store only $\bar{\ell}$ number of signals per recipient. These changes will have certain privacy and efficiency drawbacks, that we fix in the next subsection.

We take a completely different approach from [11] where the TEE stores an $M \times \bar{\ell}$ size table. We use linked lists to store the list of signals pertinent to a recipient. Every incoming signal is stored and includes a pointer to the previous pertinent signal for the same recipient. This way, the TEE only needs to keep track of the latest signal pertinent to each recipient. The previous signals are retrieved by using the pointer to the prior relevant signal in the linked list.

In more detail, for each signal the TEE receives, it stores an encrypted pointer in the server. Each pointer points to the *previous* pertinent signal for that recipient. Upon processing a new signal, the TEE updates the recipient's last signal and adds an encryption of the pointer to the previous pertinent signal. The TEE also does dummy updates to the last signal of all the other parties as well to prevent information leakage. Thus per each signal, the TEE only needs to perform $O(M)$ computation as compared to $O(M \times \bar{\ell})$ in [11].

To retrieve their signals, a receiver sends an authenticated request to the TEE. The TEE looks up the last signal pertinent to this receiver, and also the pointer to the previous signal pertinent to this receiver. The TEE requests the ciphertext corresponding to the previous signal from the server and determines the next pointer. The TEE requests these ciphertexts until the pointer is a default value

(e.g. -1). The TEE then encrypts these corresponding signals under the public key of the recipient and outputs this list of ciphertexts to the recipient.

We present an overview of this technique in Figure 1.

2.2 ORAMs to prevent information leakage

Note that the previous approach does not provide privacy from the server since it sees the exact ciphertexts that are retrieved by the TEE. To prevent this leakage we must hide the access patterns of the TEE. To this end, we use an Oblivious RAM (ORAM) to store all the signals as well as the pointers to the previously pertinent signals which ensures the server cannot know which locations are accessed by the TEE and which locations were written by the TEE.

Therefore upon receiving a signal, instead of directly writing the encrypted pointer, the TEE writes it with ORAM access. We note that the total cost to process a signal is just $O(M + \log(N))$ as opposed to the previous $O(M \times \bar{\ell})$, as each ORAM operation takes $O(\log(N))$ [21].

Getting rid of the linear dependency on M. Recall that in the approach described above the server maintains a table of size M that stores the last pertinent signal to each recipient. The TEE would need to do a dummy update to the entire table for every incoming new signal to ensure that the server does not learn the recipient of the signal. We, therefore, propose to use another ORAM to store the last pertinent signal for each recipient. The TEE therefore only needs to access the corresponding ciphertext from the ORAM, update it, and write back to the ORAM. This ensures that the runtime cost per signal sending simply becomes $O(\log(M) + \log(N))$ and the local storage remains $O(1)$ [21].

2.3 Private retrieval

As described earlier, to retrieve their signals, a recipient sends a signed request message. This signed message includes an incrementing counter value to prevent replay attacks. If the signature is correct, TEE reads all the pertinent signals from ORAM and sends them back encrypted under the recipient's public key (using a key-private PKE scheme).

Hiding the number of pertinent signals. Note that the above-described solution leaks the *number* of pertinent signals to a particular recipient. To prevent this the TEE responds to every retrieval request with exactly $\bar{\ell}$ ciphertexts, where the TEE computes encryptions of zero if the recipient has less than $\bar{\ell}$ signals or the TEE encrypts a special message that indicates that there exist more pertinent messages for that recipient.

Retrieval privacy. In the discussion, so far we have not shown how the privacy of the recipient is maintained. For example, the server will always learn when a recipient is attempting to retrieve its signals. This could also be an issue when a recipient has more than $\bar{\ell}$ signals. Since the server can identify who is performing the retrieval, if a recipient requests to retrieve signals within a short interval, the server could infer that the recipient has greater than $\bar{\ell}$ pertinent signals.

An easy way to resolve this issue is to have the recipient use the TEE's public key to encrypt the signed request. By CPA security, the recipient's identity remains private. Of course, there might be some network-level leakage. This can be resolved by, for example, using TOR. This follows the same assumption as prior works [11, 12] and some other works like meta-data privacy for Messaging Layer Security [22]. Since this is beyond the scope of this paper, we assume this is already addressed appropriately. However, solving this network-level issue via another way or analyzing the TOR solution in more detail are interesting questions left for future works.[2]

To verify the signature, we could have all the verification keys inside the TEE, but since this leads to $O(M)$ overhead in local storage, we store these keys in yet another ORAM on the server. The TEE can then privately retrieve the verification key corresponding to the recipient and verify its identity.

[2] As a side note, sender privacy also requires similar network-level care. **Thomas:** Added this footnote.

By putting everything together, we have a privacy-preserving signal retrieval method, with a cost of $O(\bar{\ell}\log(N) + \log(M))$ per retrieval.

3 Related work

Private Signaling. Private Signaling (PS) [18] tries to address the problem of recipient privacy when retrieving from a privacy-preserving message system. They present their definition of private signaling in the UC framework and present an ideal functionality $\mathcal{F}_{\mathsf{privSignal}}$, that captures the property that all the messages addressed to a recipient can be retrieved and no one except for the sender and the recipient learns the recipient of a message.

The paper provides two solutions, one based on a server with TEE and one based on the garbled circuit with two servers. A later version [11] of the paper modified the construction based on TEE, where the table of recipients mapped to their messages is stored inside the TEE. Both versions of the TEE-based solution lack some scalability, in different ways, as we discussed at the beginning of Section 2. Moreover, the constructions in [18, 11] also only achieve a slightly weaker ideal functionality (that some old messages may not be received by the recipient in some circumstances). In comparison, in this work, all messages can be retrieved by a recipient. Furthermore, we strengthen the ideal functionality (by capturing an unlinkability property, details in Section 5), and show a construction that fully realizes the ideal functionality.

Fuzzy Message Detection. FMD [5] uses a decoy-based privacy notion for the recipients. In more detail, they introduce some false positives together with the true positives, such that adversaries cannot distinguish the false positives from the true positives. This is a weaker privacy guarantee susceptible to attacks [24, 25]. Thus, FMD provides a tradeoff between privacy and efficiency for the recipient , where-in for optimal efficiency, the recipient would decrypt only its signals but its full anonymity to the server. On the other hand, to achieve full privacy the recipient needs to decrypt all signals.

Oblivious Message Retrieval. Oblivious Message Retrieval (OMR) [12] also tries to address the problem of recipient privacy when retrieving from a privacy-preserving message system. Similar to PS, they achieve full privacy. The major OMR constructions are based on leveled homomorphic encryptions. The construction is later improved in [14] and achieves better server runtime efficiency, but still lacks scalability. However, their concrete cost is still relatively high, and asymptotically, their cost is linear in both the total number of messages and the number of recipients doing the retrieval with the server. An extended group version Group OMR was recently introduced by [14]. We also briefly discuss how we can extend to the group setting (i.e., a message may be addressed to more than one recipient).

ZLiTE. ZLiTE [23] tries to deal with a similar problem. Their result is also based on a trusted execution environment and oblivious RAM. However, their approach is to let the recipient send a secret key (viewing key) to the TEE and then ask the TEE to scan the entire chain and send back the signals addressed to the recipient. This results in a server time linear in $M \cdot N$. Furthermore, the model is not entirely the same: all the works above and our work focus on having the servers as a separate component from the blockchain or the other parts of the system, while for ZLiTE, the server is also a full node in Zcash. In contrast, our model is more generic and therefore applicable in systems other than cryptocurrency as well.

3.1 Comparison with prior works

In Table 1, we compare our construction asymptotically with other works. PS1 and PS2 are introduced in [18], and PS1N is a modified version of PS1 introduced in [11]. The central idea remains the same, but for PS1, the table is stored outside the TEE and the TEE re-encrypts the table each time when

	Server runtime per send	Server runtime per retrieve	Total server runtime N sends + M retrievals	Recipient Time	Security Assumption	Environment Assumption	TEE Local Storage	Privacy
PS1 [18]	$O(\ell M)$	$O(\ell)$	$O(N\ell M)$	$O(\ell)$	PKE	TEE	$O(1)$	No unlinkability
PS1N [11]	$N\ell M$	$O(\ell)$	$O(N\ell M)$	$O(\ell)$	PKE	TEE	$O(\ell M)$	No unlinkability
PS2 [18, 11]	$O(N\bar{\ell}M)$	$O(\bar{\ell})$	$O(N\bar{\ell}M)$	$O(\bar{\ell})$	GC	Two non-colluding servers	N/A	No unlinkability
OMR [12, 14]	$O(1)$	$O(N\mathsf{polylog}(\bar{\ell}))$	$O((N\mathsf{polylog}(\bar{\ell})M)$	$O(\bar{\ell}^3)$	LWE	N/A	N/A	Full
ZLiTE [23]	$O(1)$	$O(N\log(\bar{\ell}))$	$O((N\log(\bar{\ell})M)$	$O(\bar{\ell})$	PKE	TEE	O(1)	No unlinkability
Our work	$O(\log(N)+\log(M))$	$O(\ell\log(N)+\log(M))$	$O(N\log(M)+M\ell\log(N))$	$O(\ell)$	PKE	TEE	$O(1)$	Full

Table 1: Asymptotics and privacy comparisons with prior works. N is the total number of messages on the board. M is the total number of recipients registered with a server. $\bar{\ell}$ is the number of messages that can be retrieved by the recipient per retrieval. Server runtime is calculated for N messages with M recipients, each having $\bar{\ell}$ as the upper bound of the number of messages. Unlinkability refers to the sender unlinkability and recipient unlinkability defined in Section 5. The sender runtime is $O(1)$ for all the schemes so we omit it from the table.

it updates the table; while for PS1N, they store this table inside TEE, thus trading off local storage for runtime. Since they provide trade-offs and both have some scalability issues in different ways, we compare our scheme with both.

As shown in the table, our scheme gives the second-best asymptotic server runtime to process a sent signal and a retrieve operation and is only a log factor worse than the best ones. In addition, for the total runtime for N sends and M retrievals, our scheme is undoubtedly the best. Moreover, our scheme can keep the TEE local storage $O(1)$. Thus, our scheme offers the best scalability compared to all other works.

Privacy-wise, our construction also offers the strongest privacy. In addition to that the adversary does not learn which messages are addressed to a recipient, they also do not learn who is the sender and who is the recipient (which is the unlinkability property, details in Section 5). Note that while OMR also achieves full privacy, our privacy guarantee is defined by UC, and thus provides better composability and extendability compared to OMR.

4 Preliminaries

Notation. Let λ be the security parameter, $\mathsf{poly}(\cdot)$ be a polynomial function, $\mathsf{polylog}(\cdot)$ be poly-log function, and let $\mathsf{negl}(\cdot)$ be a negligible function. M denotes the total number of recipients and N denotes the number of signals on the board. $\tilde{O}(x)$ denotes $x\log(x)$ in our paper (as opposed to $x\mathsf{polylog}(x)$).

Public board: $\mathcal{G}_{\mathsf{ledger}}$. We assume that all parties have read and write access to a public board, which we abstract via a public ledger ideal functionality $\mathcal{G}_{\mathsf{ledger}}$ (Fig 13) functionality introduced in [26]. $\mathcal{G}_{\mathsf{ledger}}$ maintains a global variable called `state` and parties can read from and write to this global state through the commands `READ` and `SUBMIT`.

In this section, we present the crucial definitions and security guarantees of the primitives used in our protocols.

Trusted Execution Environment: $\mathcal{G}_{\mathbf{att}}$. A Trusted Execution Environment (TEE[3]) is a hardware-enforced primitive. It protects the confidentiality and integrity of privacy-sensitive software and data from untrusted software including OS/hypervisor, and other applications residing on the same computer. In this paper, we model TEE as a single, globally-shared ideal functionality that is denoted as $\mathcal{G}_{\text{att}}$ following the definition from [27]. The $\mathcal{G}_{\text{att}}$ functionality is depicted in Fig. 14 in Appendix B. There are two types of invocations to $\mathcal{G}_{\text{att}}$ - *install*, which installs a piece of software, and a *(stateful) resume*, which executes the software with a given input.

[3]also referred to as *enclave*.

There are a few TEE solutions available commercially, hosted on public clouds. Among others, AWS has Nitro Enclaves [28], Microsoft Azure offers Confidential Compute using Intel SGX [29], and GCP provides AMD SEV [30] for trusted execution. We use Intel SGX to implement our protocols – for two reasons: (1) SGX has a lower Trusted Computing Base (TCB) compared to others, and (2) our construction uses ORAMs to store huge databases outside the enclave, and since SGX is an application-level TEE, it incurs lower memory-access overhead to read/write from untrusted host memory. However, we note that any TEE adhering to $\mathcal{G}_{\text{att}}$ can be used to realize our protocols.

Threat model of SGX. SGX enclaves are strongly isolated from the untrusted host and OS by encrypting all their memory using a Memory Encryption Engine. Apart from providing confidentiality and integrity of computations inside the enclave; SGX also offers an authentication mechanism via *attestation* using which a recipient can verify the correctness of the enclave. After verification, recipients can encrypt their data towards the enclave with the key in the *attestation* report.

However, there are a few limitations for SGX. In particular, it is not resilient against side-channel attacks, including page table-based [31], cache-based [32], branch-prediction-based [33], and other physical [34] side channels. An adversary can launch these side channels either via a compromised OS or host application and extract secrets using timing or control-flow analysis. To ensure confidentiality against software-based side channels, the program inside enclaves must implement oblivious memory primitives such as constant-time programming, oblivious memory-access patterns, and oblivious branching. To this end, several techniques have been proposed recently and we refer the reader to [35, 36, 37, 17] for a detailed description. Notably, ZeroTrace [17] built a library that securely implements ORAM controllers inside SGX enclaves. ZeroTrace mitigates software-based side-channel attacks by implementing the aforementioned oblivious memory primitives. In our implementation, we use ZeroTrace library to ensure our prototype is free from software-based side-channels.

Regarding the enclave memory limit, the previous processors – Intel Coffee Lake, 2017 – running SGXv1 can load up to 256 MB of enclave memory. However, the newer processors – Intel Ice Lake, 2020 – running SGXv2 can load up to 1 TB of enclave memory, hence, the memory limit is no more a concern for in-enclave storage-intensive applications. However, since a ledger may easily exceed 1TB in the future, we target smaller enclave storage for better scalability. In our implementation, we stick with SGXv1, as our protocols do not store a lot of data within the enclave. Our experiments in section 7.2 show that the enclave requires less than 50 MB of heap beside the enclave code and stack.

Oblivious RAM (ORAM). We adapt the ORAM security definition to TEE and the server [20].

At a high level, the goal of ORAM is to hide the data access pattern (which blocks are accessed and whether it is a read or write) from the server. Thus, we need two sequences of access operations with the same length to be (computationally) indistinguishable to anyone except for the TEE.

Definition 1. *An ORAM construction contains one PPT algorithm:*

- $\mathsf{data}' \leftarrow \mathsf{Access}(\mathsf{DB}, \mathsf{op}, \mathsf{a}, \mathsf{data})$*: on the input of an identifier* DB *to some database, an operation* op *that can be either* Read *or* Write*, an identifier of the data block (e.g., memory location of the data block), and a data block* data*, returns a data block* data*.*

Let $\mathsf{ORAM.Read}(\mathsf{DB}, \mathsf{a})$ *denote* $\mathsf{Access}(\mathsf{DB}, \mathsf{Read}, \mathsf{a}, \cdot)$ *(here* $\cdot$ *is a dummy input) and let* $\mathsf{ORAM.Write}(\mathsf{DB}, \mathsf{a}, \mathsf{data})$ *denote* $\mathsf{Access}(\mathsf{DB}, \mathsf{Write}, \mathsf{a}, \mathsf{data})$*.*

Let $\vec{x} = ((\mathsf{DB}, \mathsf{op}_1, \mathsf{a}_1, \mathsf{data}_1), \ldots, (\mathsf{DB}, \mathsf{op}_Q, \mathsf{a}_Q, \mathsf{data}_Q))$ *denote a data request sequence of length* Q*, where* op_i *is the operation that can be either* Read *OR* Write*,* a_i *is the identifier of the data block, and* data_i *is the data being written. We define ORAM construction's correctness and privacy as follows:*

- *(Correctness) On the input* $\vec{x}$*, it returns data that is consistent with* $\vec{x}$ *with probability* $\geq 1 - \mathsf{negl}(|\vec{x}|)$*.*
- *(Computational Privacy) Let* $A(\vec{x})$ *sequence of accesses to the remote storage given the sequence of data requests* $\vec{x}$*. An ORAM construction is said to be computationally private if for any two data request sequences* $\vec{y}$ *and* $\vec{z}$ *of the same length, their access patterns* $A(\vec{y})$ *and* $A(\vec{z})$ *are computationally indistinguishable by PPT algorithm except for the TEE.*

State-of-the-art ORAM has achieved the asymptotics of $O(\log(N))$ server time per access in the worst case, with $O(1)$ local memory (i.e., memory required in the TEE) [21].

In addition, for better flexibility, we require the ORAM to have an interface of $\mathsf{ORAM.resize}(\mathsf{DB}, N')$ that takes an ORAM database DB of size N and resize it to an ORAM database of size $N' > N$. This is naively achievable by starting a new ORAM database DB' of size N', and then read every entry from DB and write it to DB' and set $\mathsf{DB} \leftarrow \mathsf{DB}'$. Even with this naive method, the amortized efficiency per entry is simple $O(\log(N))$. One can use more efficient techniques, e.g. [38], to achieve resize more efficiently.

5 UC-Definition of Private Signaling

Threat model. We assume all the adversaries are computationally bounded, and they can read all public information, including all signals on the bulletin board, all the public keys, all accesses to the bulletin board, and all the communication between the server and the recipients. The adversaries can also act as a sender, a recipient, or a server.

In our model we require that proving privacy even against malicious servers colluding with any subset of parties. For correctness, we assume that the server is semi-honest as in prior works [11, 18, 12, 14], while other parties like the senders and the recipients can be malicious.

We assume semi-honest security for correctness since a malicious server can always choose to mount a denial-of-service attack and not send messages to the TEE or not reply to a recipient. Such an assumption is justified since mounting DoS attacks can easily be identified by users and they will just not use the service thereafter. However, a malicious sender or recipient cannot be easily identified or avoided. Note that we can easily extend our model to have multiple servers (thus the recipient can choose which server for the service), and we discuss this in more detail in Section 8.

Our primitive is expected to be used in tandem with existing privacy-preserving messaging systems (e.g., Zcash [13]). Moreover, we do not consider network-level protection (e.g., the use of TOR [39]) which is out of scope of this paper.

Formal definition. We define the problem of private signaling in the UC-framework [15]. 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, and we adapt the definition from [11] and improve upon it.

- (Correctness) Correctness means that a recipient R_i should be able to learn all signals that are intended for them.
- (Privacy) Privacy contains more complicated components:
  - (Sender unlinkability) An adversary cannot identify the sender of a signal.
  - (Signal recipient hiding) An adversary cannot learn the recipient of a signal.
  - (Recipient unlinkability) An adversary cannot learn the identity of a recipient when they retrieve their signals.
  - (Number of signals a recipient receives) An adversary cannot learn the number of pertinent signals for each recipient.

Note that *Sender unlinkability* and *Recipient unlinkability* are not captured by the prior definition in [11].

Also note that recipient unlinkability is quite essential, as our model assumes the servers to be semi-honest for the correctness of the retrieval process. Thus, if a server does not know who is doing the retrieval, they have fewer incentives to behave maliciously.

Furthermore, we want to capture the following inherent leakage. 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 some 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).

All these properties and inherent leakage are captured by the following ideal functionality.[4]

Functionality $\mathcal{F}_{\mathsf{privSignal}}$
The functionality maintains a table denoted $\mathcal{T}$ indexed by recipient R_j, that contains information on the locations of signals for the corresponding recipient.
Sending a signal (`SEND`). Upon receiving $(\texttt{SEND}, R_j, \mathsf{loc})$ from a sender S_i, send $(\texttt{SEND})$ to the adversary. Upon receiving $(\texttt{SEND}, \texttt{ok})$ from the adversary, append loc to $\mathcal{T}[R_j]$.
Retrieving signals (`RECEIVE`). Upon receiving $(\texttt{RECEIVE})$ from some R_j, send $(\texttt{RECEIVE})$ to the adversary. Upon receiving $(\texttt{RECEIVE}, \texttt{ok})$ from the adversary, send $(\texttt{RECEIVE}, \mathcal{T}[R_j])$ to the recipient R_j and update $\mathcal{T}[R_j] = []$.

Figure 2: Private Signaling functionality

Private Signaling ideal functionality. The functionality $\mathcal{F}_{\mathsf{privSignal}}$ provides the following interface - `SEND` and `RECEIVE`. Furthermore, the functionality maintains a table denoted $\mathcal{T}$. This table maintains a mapping of a recipient R_j to their pertinent locations.

- `SEND` allows a sender to send a signal by specifying the recipient and location of the message on the board. The functionality leaks to the adversary that a `SEND` request has been received and then appends the location to the recipient's row in table $\mathcal{T}$
- `RECEIVE` allows a recipient to request their signals from the functionality. The functionality leaks to the adversary that a party is trying to retrieve their messages, and then returns the corresponding row of the table to the party. Finally, the functionality flushes the same row.

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 requirements as expected.

6 A Scalable Private Signaling Protocol With TEE (Π_{TEE})

Π_{TEE} is based on the following building blocks:

1. A $\mathcal{G}_{\text{att}}$ functionality [27] as defined in Section 4. This functionality models the TEE. The program run by the TEE is described in Fig 3.
2. An Oblivious RAM scheme - $(\mathsf{ORAM.Read}, \mathsf{ORAM.Write}, \mathsf{ORAM.resize})$
3. A signature scheme $(\mathsf{Sig}, \mathsf{Ver})$
4. A key-private encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$

In Table 2 we describe all the notations and variables used in our construction.

Overview. We present a high-level overview of our construction, and the formal description is presented in Figs. 3 and 4.

Enclave Setup. The server Srv installs the program (Figure 4) for the TEE and initializes it with the following public parameters: N, an upper-bound on the total number of signals that can be stored; M, an upper-bound on the total number of parties that can register with the server; and $\bar{\ell}$, the number of signals that are retrieved by a recipient upon each request.

[4] Recall that network level leakage is not within the scope of this work.

On input* ("initialization", pp)

Store $\mathsf{pp} = (N, M, \bar{\ell})$ and locally keep a counter $\mathsf{CurIdx} \leftarrow 0, \mathsf{ctr}_{\mathsf{recip}} \leftarrow 0$.
Initialize an empty database of size N with ORAM and store its identifier DB.
Initialize an empty database of size M with ORAM and store its identifier DBkeys.
Initialize an empty database of size M with ORAM and store its identifier DBrecip.

On input* ("setup", $\mathsf{ct}_{\mathsf{keys}}$)

$(\mathsf{pk}, \Sigma.\mathsf{vk}) \leftarrow \mathsf{Dec}(\mathsf{esk}, \mathsf{ct}_{\mathsf{keys}})$, set $\mathsf{RID} \leftarrow \mathsf{ctr}_{\mathsf{recip}}$.
$\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DBkeys}, \mathsf{RID}, \mathsf{Enc}(\mathsf{esk}, (\Sigma.\mathsf{vk}, \mathsf{pk}, 1)))$
$\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DBrecip}, \mathsf{RID}, \mathsf{Enc}(\mathsf{esk}, [-1, -1]))$
$\mathsf{ctr}_{\mathsf{recip}} \leftarrow \mathsf{ctr}_{\mathsf{recip}} + 1$
return $\mathsf{pk}, \mathsf{RID}, \mathsf{ctr}_{\mathsf{recip}}$

On input* ("send", $\mathsf{ct}_{\mathsf{Signal}}$)

$[\mathsf{RID}, \mathsf{loc}] \leftarrow \mathsf{Dec}(\mathsf{esk}, \mathsf{ct}_{\mathsf{Signal}})$
$\mathsf{data} \leftarrow \mathsf{ORAM.Read}(\mathsf{DBrecip}, \mathsf{RID}, \cdot)$ ▷ $\cdot$ is a dummy data variable
$[\mathsf{PrevIdx}, \mathsf{PrevLoc}] \leftarrow \mathsf{Dec}(\mathsf{esk}, \mathsf{data})$
$\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DBrecip}, \mathsf{RID}, \mathsf{Enc}(\mathsf{esk}, [\mathsf{CurIdx}, \mathsf{loc}]))$
$\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DB}, \mathsf{CurIdx}, \mathsf{Enc}(\mathsf{esk}, [\mathsf{PrevIdx}, \mathsf{PrevLoc}]))$
$\mathsf{CurIdx} \leftarrow \mathsf{CurIdx} + 1$
return CurIdx.

On input* ("receive", σ)

$\sigma' \leftarrow \mathsf{Dec}(\mathsf{esk}, \sigma)$
Parse $\sigma' = \mathsf{Sig}(\mathsf{RID}, \mathsf{ctr})$
$\mathsf{data} \leftarrow \mathsf{ORAM.Read}(\mathsf{DBkeys}, \mathsf{RID}, \cdot)$
$(\Sigma.\mathsf{vk}, \mathsf{pk}, \mathsf{ctr}_{\mathsf{RID}}) \leftarrow \mathsf{Dec}(\mathsf{esk}, \mathsf{data})$
if $\Sigma.\mathsf{Ver}(\Sigma.\mathsf{vk}, \sigma') = 1$ and $\mathsf{ctr} = \mathsf{ctr}_{\mathsf{RID}}$ **then**
  Initialize an empty vector $\vec{R}$
  $[\mathsf{index}, \mathsf{loc}] \leftarrow \mathsf{Dec}(\mathsf{esk}, \mathsf{ORAM.Read}(\mathsf{DBrecip}, \mathsf{RID}, \cdot))$
  while $|\vec{R}| \leq \bar{\ell}$ **do**
    if $|\vec{R}| = \bar{\ell}$ **then**
      $\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DBrecip}, \mathsf{RID}, \mathsf{Enc}(\mathsf{esk}, [\mathsf{index}, \mathsf{loc}]))$
      Append -1 to $\vec{R}$ if index from previous iteration is -1, else append -2
      Break the loop
    Append loc to $\vec{R}$
    if index from the previous iteration is -1 **then**
      Do a dummy ORAM read and set $[\mathsf{index}, \mathsf{loc}] = [-1, -1]$
    else
      $[\mathsf{index}, \mathsf{loc}] \leftarrow \mathsf{Dec}(\mathsf{esk}, \mathsf{ORAM.Read}(\mathsf{DB}, \mathsf{index}, \cdot))$
  Update $\mathsf{ctr}_{\mathsf{RID}} = \mathsf{ctr}_{\mathsf{RID}} + 1$
  $\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DBkeys}, \mathsf{RID}, \mathsf{Enc}(\mathsf{esk}, (\Sigma.\mathsf{vk}, \mathsf{pk}, \mathsf{ctr}_{\mathsf{RID}})))$
  return $\mathsf{Enc}(\mathsf{pk}, \vec{R})$
else
  return $\perp$

On input* ("extend", N')

$\mathsf{DB}' \leftarrow \mathsf{ORAM.resize}(\mathsf{DB}, N')$
Clear DB and set $\mathsf{DB} \leftarrow \mathsf{DB}'$
Update $\mathsf{pp}.N \leftarrow N'$

On input* ("extendrecip", M')

$\mathsf{DBkeys}' \leftarrow \mathsf{ORAM.resize}(\mathsf{DBkeys}, M')$
Clear DBkeys and set $\mathsf{DBkeys} \leftarrow \mathsf{DBkeys}'$
$\mathsf{DBrecip}' \leftarrow \mathsf{ORAM.resize}(\mathsf{DBrecip}, M')$
Clear DBrecip and set $\mathsf{DBrecip} \leftarrow \mathsf{DBrecip}'$
Update $\mathsf{pp}.M \leftarrow M'$

Figure 3: Program Prog run by $\mathcal{G}_{\mathrm{att}}$

$\mathcal{G}_{\text{att}}$	Secure processor functionality Figure 14
loc	Location of message on the board
N	Total number of messages on the board
M	Total number of recipients registered in TEE
$\bar{\ell}$	Number of signals per retrieval
RID	Identifier of the recipient
$\mathsf{ct}_{\mathsf{keys}}$	Ciphertext of encryption key and & verification key
ctr_i	Counter to prevent replayability of signatures for party i
$\mathsf{ct}_{\mathsf{Signal}}$	Encryption of (RID, loc)
mpk, msk	Attestation keys of $\mathcal{G}_{\text{att}}$ functionality
epk, esk	Encryption keys of $\mathcal{G}_{\text{att}}$
DB	ORAM database of N encrypted pointers
DBrecip	ORAM database of the latest signal of the M recipients
DBkeys	ORAM database of the M recipients' keys
CurIdx	A counter locally kept by TEE on how many signals have been received.
$\mathsf{ctr}_{\mathsf{recip}}$	A counter locally kept by TEE on how many recipients have been registered.
PrevIdx	The entry index in DB of the previous pertinent signal.
PrevLoc	The location of the previous pertinent signal.

Table 2: Notations for Π_{TEE}

Upon running the initialization command, the TEE initializes three databases with an ORAM scheme. The first ORAM denoted DB, is of size N and stores all the incoming signals to the server. The second ORAM denoted DBkeys, is of size M and stores the verification keys of the registered recipients. The third ORAM denoted DBrecip, is of size M and stores information on the last received signal per registered recipient. TEE also locally initializes two counters CurIdx and $\mathsf{ctr}_{\mathsf{recip}}$ as zeros. These counters keep track of the number of incoming signals and the number of registered recipients respectively.

Note that the three ORAM databases are upper-bounded by N or M (usually required by the underlying ORAM construction). If the above two counter values reach N and M respectively, we use resize described in Section 4 to resize the databases to $2N, 2M$ respectively and set $N \leftarrow 2N, M \leftarrow 2M$. Note that we can start with large N, M (e.g., 2^{40}) with only limited effects on performance (as our performance scales only with $\log(N)$ and $\log(M)$), and thus in practice, resize may not need to be used.

Also note that all the ORAM data entries are encrypted under TEE secret key: esk.

Registration. To register with the server, a party R_i first obtains the encryption key epk and the verification key mpk of the TEE. R_i then generates its own encryption keys $(\mathsf{pk}_i, \mathsf{sk}_i)$ (recall that this encryption scheme is key private) and signature keys $(\Sigma.\mathsf{vk}_i, \Sigma.\mathsf{sk}_i)$. R_i then encrypts its public key and verification key under the public key of the TEE and sends them to the server indicating that it wants to register with it.

The server then executes the TEE with input "setup" and the ciphertext containing the public key (pk_i) and the verification key $(\Sigma.\mathsf{vk}_i)$ of R_i (the party to wants to register). The TEE first decrypts the ciphertext to obtain $(\mathsf{pk}_i, \Sigma.\mathsf{vk}_i)$. The TEE then encrypts the $(\mathsf{pk}_i, \Sigma.\mathsf{vk}_i)$ along with a counter $\mathsf{ctr}_i = 1$ under the key esk. The counter is used to record the number of retrieval requests made by R_i (including the next one, thus initialized as 1).

These keys are then written (encrypted under esk, written using ORAM) to the DBkeys at location RID, where $\mathsf{RID} = \mathsf{ctr}_{\mathsf{recip}}$ is simply a unique index of the recipient. Since the write accesses are hidden using an ORAM and encrypted under esk, the server does not learn the RID or the keys that were

Enclave setup

Srv:

1. Run $\mathcal{G}_{\text{att}}.\mathsf{install}(\mathsf{Prog})$ to get eid.
2. $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“initialization”}, \mathsf{pp} = (N, M, \bar{\ell})))$
   ▷ pp are parameters passed to initialize the protocol

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 $(\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i})$ to Srv.
4. Await $((eid, \mathsf{pk}_i, \mathsf{RID}), \sigma_T)$ from Srv.
5. Assert $\Sigma.\mathsf{Ver}_{\mathsf{mpk}}((eid, (\mathsf{pk}, \mathsf{RID}, \mathsf{ctr}_{\mathsf{recip}})), \sigma_T) = 1$ and publish $(\mathsf{pk}_i, \mathsf{RID})$. Initialize $\mathsf{ctr}_i = 0$.
   ▷ Note that if the returned pk_i is not the pk_i sent, abort.

Srv:

1. Upon receiving $(\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i})$ from R_i, let $((\mathsf{pk}_i), \sigma_T) = \mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i}))$. Upon receiving ORAM request $\mathsf{data} \leftarrow \mathsf{ORAM.Access}(\cdot,\cdot,\cdot,\cdot)$ from TEE when executing the $\mathcal{G}_{\text{att}}.\mathsf{resume}(\cdot,\cdot)$ operation: process and return data to $\mathcal{G}_{\text{att}}$.
2. Send $((eid, (\mathsf{pk}, \mathsf{RID}, \mathsf{ctr}_{\mathsf{recip}})), \sigma_T)$ to R_i.
3. Upon return of $\mathsf{ctr}_{\mathsf{recip}}$ from TEE, if $\mathsf{ctr}_{\mathsf{recip}} = M$, call procedure $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“extendrecip”}, 2M))$ and update $M \leftarrow 2M$.

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

1. Sender S gets $(\mathsf{mpk}, \mathsf{epk}) := \mathcal{G}_{\text{att}}.\mathsf{getpk}()$ and computes $\mathsf{ct}_{\mathsf{Signal}} = \mathsf{Enc}(\mathsf{epk}, [\mathsf{RID}, \mathsf{loc}])$. S sends $(\texttt{SUBMIT}, (\texttt{SEND}, \mathsf{ct}_{\mathsf{Signal}}))$ to $\mathcal{G}_{\mathsf{ledger}}$

Srv:

1. Send READ to $\mathcal{G}_{\mathsf{ledger}}$
2. Upon receiving $(\texttt{SEND}, \mathsf{ct}_{\mathsf{Signal}})$: Call $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“send”}, \mathsf{ct}_{\mathsf{Signal}}))$. Upon receiving ORAM requests $\mathsf{data} \leftarrow \mathsf{ORAM.Access}(\cdot,\cdot,\cdot,\cdot)$: process and return data to $\mathcal{G}_{\text{att}}$.
3. Upon return of CurIdx from TEE. If $\mathsf{CurIdx} = N$, call procedure $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“extend”}, 2N))$ and update $N \leftarrow 2N$.

Procedure RECEIVE

Recipient R_i:

1. Compute $\sigma_i = \mathsf{Enc}(\mathsf{epk}, \mathsf{Sig}(\mathsf{RID}, \mathsf{ctr}_i))$ and send $(\texttt{RECEIVE}, \sigma_i)$ to Srv. Await $((eid, \mathsf{ct}), \sigma_T)$ from Srv
2. **return** $\vec{R} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$.

Srv:

1. Upon receiving $(\texttt{RECEIVE}, \sigma_i)$ from R_i, let $((eid, \mathsf{ct}), \sigma_T) = \mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“receive”}, \sigma_i))$. Upon receiving ORAM requests $\mathsf{data} \leftarrow \mathsf{ORAM.Access}(\cdot,\cdot,\cdot,\cdot)$: process and return data to $\mathcal{G}_{\text{att}}$.
2. Send $((eid, \mathsf{ct}), \sigma_T)$ to R_i

Procedure ExtendDB

Srv:

1. $N \leftarrow 2N$.
2. $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“extend”}, N))$. Upon receiving ORAM request $\mathsf{data} \leftarrow \mathsf{ORAM.Access}(\cdot,\cdot,\cdot,\cdot)$ from TEE when executing the $\mathcal{G}_{\text{att}}.\mathsf{resume}(\cdot,\cdot)$ operation: process and return data to $\mathcal{G}_{\text{att}}$
3. Free the space of the original databases DB

Procedure ExtendRecip

Srv:

1. $M \leftarrow 2M$.
2. $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“extendrecip”}, M))$. Upon receiving ORAM request $\mathsf{data} \leftarrow \mathsf{ORAM.Access}(\cdot,\cdot,\cdot,\cdot)$ from TEE when executing the $\mathcal{G}_{\text{att}}.\mathsf{resume}(\cdot,\cdot)$ operation: process and return data to $\mathcal{G}_{\text{att}}$
3. Free the space of the original databases $\mathsf{DB}_{\mathsf{recip}}, \mathsf{DB}_{\mathsf{keys}}$

Figure 4: The protocol for private signaling Π_{TEE} in the $\mathcal{G}_{\text{att}}$ hybrid world.

written to the ORAM. Next, the TEE initializes the last message received by R_i as a default $[-1,-1]$. Here the first -1 is denoted as $\mathsf{PrevIdx}$, indicating the index of the previous signal received by the recipient stored by the TEE on the server; and the second -1 is denoted as loc, indicating the location that the current signal wants to give on the public bulletin board. The first index is needed for the TEE to complete the linked-list-like retrieval; and the second index is used for the recipient the fetch the real message pointed by the signal.

The TEE then encrypts these initialized values under its secret key esk and writes them to location RID in the $\mathsf{DBrecip}$ using ORAM. The TEE finally increments $\mathsf{ctr}_{\mathsf{recip}}$ by 1, and returns $(\mathsf{pk}_i, \mathsf{RID}, \mathsf{ctr}_{\mathsf{recip}})$. Note that by the definition of $\mathcal{G}_{\mathrm{att}}$ functionality, all messages that are output by the program are signed by the TEE. This signed output $(\mathsf{pk}_i, \mathsf{RID}, \mathsf{ctr}_{\mathsf{recip}})$ is then sent by the server to the party R_i. The recipient verifies pk_i is indeed the public key it sent to the server signed by the TEE, and then it publishes $\mathsf{pk}, \mathsf{RID}$ [5].

When the counter $\mathsf{ctr}_{\mathsf{recip}}$ is equal to M, the server executes the TEE program to resize the DBkeys to size $2M$ (using resize in Section 4). Again, this can be avoided in practice by setting M to a large enough number at first.

Send. To send a signal to a party R_i with identity RID that a message exists for them at position loc on the public bulletin board, a sender S does the following. It first retrieve the keys of the TEE - epk and mpk. The sender then encrypts $(\mathsf{RID}, \mathsf{loc})$ under epk and posts the resulting ciphertext $\mathsf{ct}_{\mathsf{Signal}}$ to the board (denoted $\mathcal{G}_{\mathsf{ledger}}$). The server reads the board and retrieves the ciphertext $\mathsf{ct}_{\mathsf{Signal}}$ and executes the TEE with the "send" command and inputs $\mathsf{ct}_{\mathsf{Signal}}$.

Upon receiving the "send" command, TEE proceeds as follows. The TEE first decrypts the ciphertext to obtain $(\mathsf{RID}, \mathsf{loc})$ encrypted by the sender. The TEE then reads location RID from $\mathsf{DBrecip}$ to obtain the ciphertext encrypting the last received location for recipient RID, decrypts it, and parses it as $[\mathsf{PrevIdx}, \mathsf{PrevLoc}]$ (which is $[-1,-1]$ if it were the first message) for RID (recall that $\mathsf{PrevIdx}$ is used to iterate all the signals to recipient RID and $\mathsf{PrevLoc}$ is the location that needs to be sent back to the recipient during retrieval). The TEE then writes $[\mathsf{CurIdx}, \mathsf{loc}]$ to $\mathsf{DBrecip}$ (again, encrypted under esk). Note that this correctly updates the last received signal for RID in $\mathsf{DBrecip}$.

Next, the TEE writes encryption of $[\mathsf{PrevIdx}, \mathsf{PrevLoc}]$ at index CurIdx in DB . Finally, the TEE increments CurIdx and returns CurIdx to the server. As above, when $\mathsf{CurIdx} = N$, the server executes the TEE program to resize the DB to $2N$.

Retrieving signals. To retrieve their signals, a party R_i with identity RID first computes a signature on the RID and ctr_i, where ctr_i is the number of times a retrieval request has been made (including this current one). For instance, if it is the first time, $\mathsf{ctr}_i \leftarrow 1$. Recall that the TEE stores the same counter with the encryption and verification keys in DBkeys at location RID. The recipient then encrypts this signed request under epk and sends it to the server. The CPA security of the encryption scheme ensures that the server does not know the requesting party's identity.

The server invokes the TEE with "receive" command with the ciphertext sent by the recipient as input. The TEE decrypts the ciphertext to obtain the RID and ctr_i. Then it queries DBkeys at location RID to retrieve the keys $(\mathsf{pk}_{\mathsf{RID}}, \mathsf{vk}_{\mathsf{RID}})$ and counter value $\mathsf{ctr}_{\mathsf{RID}}$ associated with the RID. It checks that the received signature is valid against $\mathsf{vk}_{\mathsf{RID}}$ and that $\mathsf{ctr}_{\mathsf{RID}} = \mathsf{ctr}_i$. Upon each retrieval, the TEE increments the ctr_i value stored inside DBkeys. This ensures that no malicious party is able to simply replay a previous request and learn signals associated with an honest recipient.

The TEE then initializes an empty output vector $\vec{R}$ of length $\bar{\ell}$. This vector will include the encryptions of the locations pertinent to the recipient. The TEE first retrieves the latest signal received by R_i from $\mathsf{DBrecip}$ at location RID. Recall that this is of the form $[\mathsf{index}, \mathsf{loc}]$ (after decryption using esk). The TEE then writes loc to $\vec{R}$. The TEE then reads the signal stored at position index from DB, which is again of the form $[\mathsf{index}', \mathsf{loc}']$. The TEE then writes loc' to $\vec{R}$. The TEE repeats this

[5] Note that in our construction, we let the TEE to pick a unique RID for the recipient for simplicity (where RID means that the recipient is the RID-th recipient registered by the server). However, for most ORAMs (e.g., all the hierarchical ORAMs) we can also simply let the recipient randomly pick $\mathsf{RID} \in \{0,1\}^\lambda$. This is because most ORAMs allow access to a "keyword-like location, instead of just the indices without any overhead. This does not matter for the single server setting we discuss now, but comes in handy when we discuss the multi-server setting in Section 8.

operation now with index' until filling $\vec{R}$ with $\bar{\ell}$ locations. If a recipient has less than $\bar{\ell}$ signals, perform dummy ORAM reads and get $[\mathsf{index}', \mathsf{loc}'] = [-1, -1]$.

After that, the TEE writes the last $[\mathsf{index}', \mathsf{loc}']$ to DBrecip, maintaining the invariant that the last non-retrieved signal is stored in DBrecip (which is $[-1, -1]$ if there are $\leq \bar{\ell}$ messages). If the last index' retrieved is -1, indicating that all messages of the recipient have been retrieved in $\vec{R}$, the TEE appends -1 to $\vec{R}$. On the other hand, if index' is not -1 indicating that there exist more signals for R_i, the TEE appends -2 to $\vec{R}$, to notify the recipient.

As mentioned above the TEE updates the request counter $\mathsf{ctr}_{\mathsf{RID}}$ that corresponds to R_i and performs a write to DBkeys. Finally, the TEE encrypts the vector $\vec{R}$ under the public key of R_i and sends it to the recipient. Since the encryption scheme is key private, the identity of the recipient is kept private.

Asymptotic analysis. We now analyze the asymptotic costs for the TEE to process a signal upon execution of the "send" command, and the cost of retrieval of locations upon execution of the "receive" command. Recall that each ORAM access costs $O(\log N)$ computation for an ORAM of size N. The "send" command consists of one read to DBrecip and a write to DBrecip and another write to DB. Thus the total computational complexity per "send" invocation is only $O(\log N + \log M)$. Likewise, for "receive" command, the TEE performs one read to DBkeys, $\bar{\ell}$ reads to DB and one write to DBrecip and DBkeys. Thus the total computational complexity per "receive" is only $O(\bar{\ell} \log N + \log M)$.

Theorem 1. *Assume that the signature scheme Σ is existentially unforgeable under chosen message attacks, the encryption scheme Enc is CPA secure and key-private (Definition 2), and the underlying ORAM is correct and private. 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 ($\mathcal{G}_{\mathrm{att}}$, $\mathcal{G}_{\mathsf{ledger}}$) ideal functionalities. The simulator internally simulates the $\mathcal{G}_{\mathrm{att}}$ and the $\mathcal{G}_{\mathsf{ledger}}$ functionalities to the adversary. We consider two cases of corruption here and in both cases, we show that the simulator can simulate without learning the locations of honest recipients. We briefly describe the main idea in the simulation of the aforementioned corruption cases:

- *Sender and server are corrupt*: The simulator receives a ("send", $\mathsf{ct}_{\mathsf{Signal}}$) command via the $\mathcal{G}_{\mathrm{att}}$ 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 (SEND, R_i, loc) to the $\mathcal{F}_{\mathsf{privSignal}}$ ideal functionality on behalf of the adversary.
- *Receiver and server are corrupt*: The simulator receives a ("receive", ctr, σ) command via the $\mathcal{G}_{\mathrm{att}}$ interface from the adversary. The simulator decrypts σ and verifies the signature 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 present our full proofs in Appendix A.3.

□

Finally as noted in [11], it is desirable to not rely on just one server for private signaling. To this end, we describe how to extend our protocol to the multi-server setting in Section 8.

7 Implementation and Evaluation

This section describes the implementation details of our prototype (proof-of-concept) and its evaluation. All our source code is publicly available at [16].

7.1 Implementation details

We use Intel SGX [29] to instantiate the $\mathcal{G}_{\text{att}}$ functionality of Π_{TEE}, and we implement the program Prog (Figure 3) running inside the SGX enclave using Intel SGX Linux SDK [40]. Currently, our prototype implements "initialization", "setup", "send", and "receive" commands of the protocol.

For encryption and authentication, we use OpenSSL [41] on the recipient side and SGX SSL [42] on the enclave side. In the prototype, the recipient uses RSA-OAEP [43] to submit encrypted send/receive requests to the enclave and the enclave uses Elliptic Curve Integrated Encryption Scheme (ECIES) to respond with encrypted replies to the recipient. Using RSA for encrypting responses to the recipients is not storage efficient because it requires storing longer keys (527 byte-keys using 2048-bit RSA modulus) for every recipient. Instead, we use shorter EC keys (65 byte-keys using Prime256v1 curve) for encrypting responses toward the recipients. This reduces the storage required per recipient by an order of magnitude (given each recipient requires two keys, pk and Σ.vk). For authentication, the prototype uses the ECDSA signature scheme with Prime256v1 curve.

Using ZeroTrace library[17]. For ORAMs, the prototype uses ZeroTrace to instantiate two PathORAMs [20] within the enclave (one for messages DB and the other for recipients including DBkeys and DBrecip). ZeroTrace is side-channel resistant (as mentioned in Section 4), and we further ensure that our code implementing Figure 3, using ZeroTrace, also follows oblivious memory primitives. The actual ORAM storage is implemented outside the enclave, within the server's untrusted memory. Note that ZeroTrace implements Recursive PathORAM[6] that has a runtime of $O(\log^2(N))$ for a database with size N. All ORAM storage in the server's untrusted memory is encrypted using AES CTR with a key only known to the enclave. Additionally, ZeroTrace implements a Merkle Hash Tree that cryptographically ties all blocks together to ensure the storage outside the enclave is integrity-protected.

For benchmarking, we deploy the prototype on Azure DC2s v2 (Confidential Compute, 3.7 GHz Intel® Xeon E-2288G Coffee Lake) instance with 2 vCPUs and 8GB RAM. Our prototype runs on Ubuntu 20.04 LTS OS and in total has $\approx 5k$ lines of C++ code ($\approx 1k$ for the enclave Prog, $\approx 3k$ for ZeroTrace lib, and $\approx 1k$ for the recipient application).

7.2 Evaluation

Server Computation. Figures 5a and 5b illustrate the server runtime for to process a "send" request and to process a "receive" request while varying the number of recipients M and number of messages N (as a multiplier of M), with $\bar{\ell} = 1$, implying only one message is retrieved. The plots represent enclave processing time after being invoked by the server during procedure `SEND` and `RECEIVE` respectively. This enclave processing time includes one RSA decryption, three ORAM accesses, and one ECDSA signature computation; additionally an ECDSA verification during `RECEIVE`. As shown in the figures, this server runtime for both procedures increases poly-logarithmically as a function of N and M due to ORAM accesses. Concretely, Π_{TEE} takes at most ~70 ms for processing one signal (during both procedures) for 1 million recipients and 10 million messages.

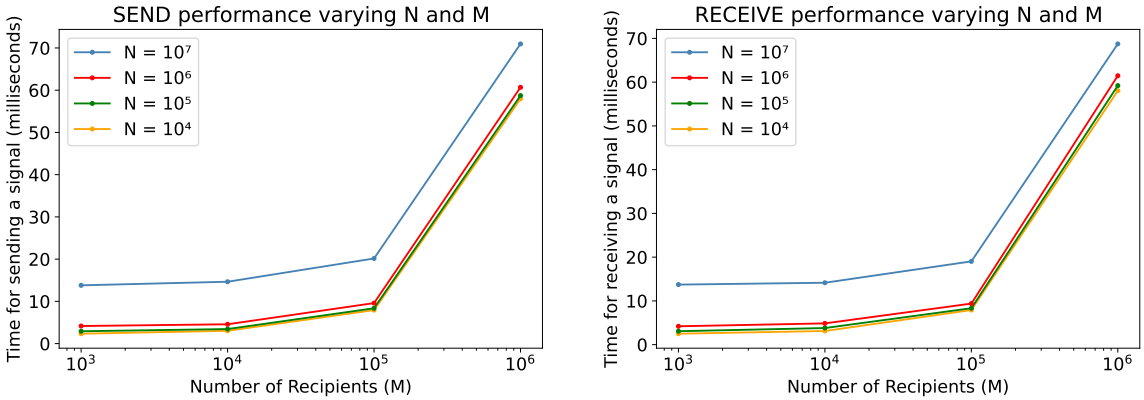
Figure 6 depicts the server runtime while varying the number of messages retrieved $\bar{\ell}$. As $\bar{\ell}$ increases, the time taken to process `RECEIVE` increases linearly, while the time for `SEND` is constant, as expected. This is because, for every message retrieved from the message database DB, the enclave must perform one ORAM Read. Compared to `SEND`, `RECEIVE` performs better when there are multiple signals to be processed, which is due to the fact that during `RECEIVE` the enclave can retrieve all of the signals within the same request, unlike during `SEND` where it has to process them one at a time. That said, the baseline performance for both procedures when $\bar{\ell} = 1$ is almost the same. Concretely, Π_{TEE} takes less than 6s to receive 100 messages when there are 1 million recipients and 10 million messages in the database.

Recipient Computation. The recipient is only required to perform encryption, decryption, signa-

[6] We opt for PathORAM instead of the asymptotically better [21], since concretely, PathORAM is much faster.

	Baseline	Π_{TEE}	PS1 [18]	PS1N [11]	PS2[11]	OMR [12, 14]	FMD [5]
Server computation (per signal per recipient)	NA	**0.011** ms	60 ms	0.228 ms	292 ms	21 ms	0.02 ms
Recipient computation	400 s	< 0.1 ms	< 0.1 ms	< 0.1 ms	< 0.1 ms	5 ms	3.13 s
Signal size	NA	300 bytes	300 bytes	300 bytes	600 bytes	956 bytes	68 bytes
Security	Full	Full	No recipient unlinkability	No recipient unlinkability	No recipient unlinkability	Full	pN-msg-anonymity $p = 2^{-5}$
Environment Assumption	NA	TEE	TEE	TEE	Non-colluding servers	NA	NA

Table 3: Performance and security comparisons between prior work PS1, PS1N, PS2, FMD, OMR, and our protocol Π_{TEE}, assuming a recipient connect once a day. These results are measured for parameters: $N = 500,000$, $M = 500$, and an upper bound of the number of signals retrieved per recipient $\bar{\ell} = 50$. Server computation is calculated using: (one `SEND` time / M recipients) + (one `RECEIVE` time / N signals).



(a) The average time taken by the server to process Π_{TEE} "send".

(b) The average time taken to process Π_{TEE} "receive".

Figure 5: Here Y-axis represents the time taken in milliseconds, while the X-axis represents the number of recipients M in log-10 scale. Each line plot depicts the number of signals N.

ture generation, and verification. In our experiments, these operations took less than 1 ms during `RECEIVE` procedures, even when processing as many as 100 signals during `RECEIVE`. This low latency for the recipient during `RECEIVE` is a consequence of using ECIES encryption instead of RSA.

7.3 Comparison with related work

We compare our protocols with the plain baseline solution and the state-of-the-art related work: PS1, PS1N, PS2, FMD, and OMR in table Table 3. By baseline solution, we mean to simply use trial decryption from [13] to decrypt all the signals on the board to find relevant ones. Hence, for this solution, there are no servers, and thus, there is only recipient computation. For the prior work, we took the results directly from PS, OMR, and FMD papers. For PS1, we generously estimate the results for parameters $\bar{\ell} = 20$, $M = 100$. Note that signal size is based on RSA encryption. From the table, we observe that our results have significantly lower server runtime per signal per recipient, compared to any other scheme. Moreover, all the other schemes scale linearly in N and M, while our scheme scales sublinearly. Compared to OMR[7] (non-TEE-based solution), our construction is *3-4 orders of magnitude faster*. Compared to the prior TEE-based-PS constructions, our construction is still *2-3 orders of magnitude faster*.

[7]OMR [12, 14] retrieves the payloads of the messages (instead of just indices). For just indices, they are about 2x faster.

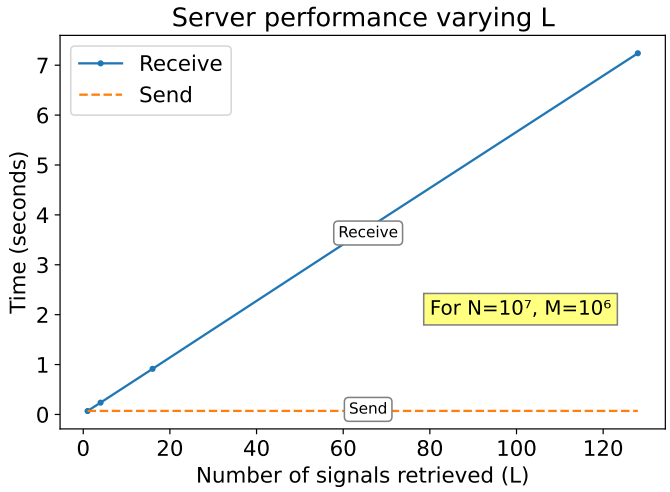


Figure 6: The average time taken to process Π_{TEE} "receive" with respect to the number of signals retrieved $\bar{\ell}$, while keeping its ORAM capacity constant, i.e., $N = 10^7$ and $M = 10^6$. Here Y-axis represents the time taken in seconds and X-axis represents $\bar{\ell}$.

Similar to PS1, PS1N, and PS2, our scheme is designed to retrieve the indices only, which are $\log_2(500000)$ bits, and we use a 4-byte integer to store them. After obtaining the indices, the clients can use techniques like PIR to fetch all the messages from the ledger. Additionally, our scheme can be extended to support direct payload retrieval, akin to OMR, by enlarging the ORAM data block size. This expansion does not significantly affect performance, as it simply involves fetching a larger consecutive memory chunk.

Security-wise, we provide equal guarantees as the baseline scheme and OMR scheme, assuming TEE is trusted. Thus, from the table, it is clear that our scheme performs significantly better than the state-of-the-art even for small parameters. And for larger parameters, such as $N = 10^7$ and $M = 10^6$, it is evident from the figures above that only our scheme can support such scalability.

8 Extending to multiple servers

In real-world applications, especially in applications like cryptocurrencies, it may be desirable to have multiple servers serving the parties. Furthermore, recipients may not like to share to which server they are registered. Therefore, it is non-trivial to design such a scheme with little overhead, while fully preserving the original privacy guarantee together with new privacy requirement.

To achieve full privacy in the multi-server model, we need three additional properties for our PKE scheme used by the TEE: *wrong-key decryption detection*, *public key unlinkability*, and *key-private encryption*. We present the formal definitions in Appendix A.1

Overview. In the multi-server setting, we enable recipients to register with the TEE of their choice, and each recipient announces the TEE public key as well as their public key. Thus if any entity is able to link a signal to a TEE, then the anonymity set for the recipient is reduced to only the set of parties registered with the TEE. To prevent this, we use key-private encryption to encrypt the location and RID. This ensures that any entity observing the signal cannot identify the public key of the TEE to which the ciphertext is associated. In fact, any server running the system cannot tell this either. Therefore every signal on the board will be processed by every TEE. While decrypting the ciphertext, the wrong-key decryption detection property above ensures that a TEE is able to determine correctly when a signal is pertinent to one of the parties that are registered with the TEE.

The TEE does dummy operations if the signal ciphertext is not encrypted under its key, and does the correct operations otherwise.

Two questions remain are: (1) in the single-server case, RID is issued by the TEE, so how should RID be generated for the multi-server case; (2) how can the sender know which TEE public key to use to encrypt the location and the RID.

To answer the first question (1), we simply let RID be $h(\mathsf{pk})$ where pk is the public key of the recipient and h is some collision resistant hash function to $\{0,1\}^\lambda$. Note that ORAM uses RID to access the database, and now RIDs are (random) bits instead of a number in $[N]$. However, this can be done without any overhead for most ORAM constructions [19, 20, 21], and they naturally achieve this property. [8] Thus, issue (1) can be resolved easily.

The second problem (2) is more involved. Naively, the recipient can simply include the *eid* of the TEE inside their public key, such that the sender can use *eid* to find the public key of the TEE. However, this leaks some information: everyone learns at which TEE the recipient is registered. Therefore, to avoid such information leakage, during recipient j's registration, the TEE generates a fresh TEE public key epk_j and sends it to the recipient. The recipient publishes $(\mathsf{pk}_j, \mathsf{epk}_j)$. By the property of public-key-unlinkability, this does not leak at which TEE the recipient j is registered.

These complete our construction for multiple TEEs. Moreover, it is easy to see that the efficiency pretty much remains the same. The only overhead is that now the recipients need to publish epk_j in addition to pk_j (which is still only a very small overhead). Everything else remains exactly the same.

We present the formal updates to the program in Fig 7 and 8. For the unchanged procedures, we omit them for readability, and use purple to highlight the changed lines.

Theorem 2. *Assume that the signature scheme Σ is existentially unforgeable under chosen message attacks, the encryption scheme* Enc *is CPA secure, key private (Definition 2), and satisfies wrong-key-decryption (Definition 3) and public-key-unlinkability (Definition 4), the ORAM scheme is secure and the encryption scheme* Enc_1 *is key-private. 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 $(\mathcal{G}_{\mathrm{att}}, \mathcal{G}_{\mathsf{ledger}})$ ideal functionalities. The simulator internally simulates the $\mathcal{G}_{\mathrm{att}}$ and the $\mathcal{G}_{\mathsf{ledger}}$ functionalities to the adversary. The simulator works similarly to the simulator for the single server case, except that the simulator announces a key-private and public key unlinkable encryption key in the setup. Each time the simulator simulates an honest `SEND`, it generates a new encryption key and encrypts 0 under this key. By the key-privacy property of our encryption, this simulated world is indistinguishable from the real world. Moreover, we require that the simulator will abort, if on behalf of any of the TEEs, the decryption under the wrong key gives a valid plaintext. Since we use the property of wrong-key decryption detection in our scheme, this event will occur with negligible probability. Hence the simulated world and the real world are indistinguishable. □

Analysis: The runtime of sending and fetching a signal remains exactly the same as our single-server construction as benchmarked in Section 7.2. The only difference is that now each recipient needs to publish two public keys as their identity. However, this is relatively cheap: roughly several hundred bytes as shown in Table 3.

[8] Note that the database still remains size $O(M)$ for M registered recipients, instead of $O(2^\lambda)$. Thus, the efficiency remains the same.

On input* ("initialization", pp)

Store $\mathsf{pp} = (N, M, \bar{\ell})$ and locally keep a counter $\mathsf{CurIdx} \leftarrow 0, \mathsf{ctr}_{\mathsf{recip}} \leftarrow 0$.
Initialize an empty database of size $(N+1)$ with ORAM and store its identifier DB.
Initialize an empty database of size $(M+1)$ with ORAM and store its identifier DBkeys.
Initialize an empty database of size $(M+1)$ with ORAM and store its identifier DBrecip.
▷ "+1" is for dummy writes. That slot is never touched for non-dummy reads/writes.

On input* ("setup", $\mathsf{ct}_{\mathsf{keys}}$)

$(\mathsf{pk}, \Sigma.\mathsf{vk}) \leftarrow \mathsf{Dec}(\mathsf{esk}, \mathsf{ct}_{\mathsf{keys}})$, set $\mathsf{RID} \leftarrow h(\mathsf{pk})$ (h is a publicly known hash function).
$\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DBkeys}, \mathsf{RID}, \mathsf{Enc}(\mathsf{esk}, (\Sigma.\mathsf{vk}, \mathsf{pk}, 1)))$
$\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DBrecip}, \mathsf{RID}, \mathsf{Enc}(\mathsf{esk}, [-1, -1]))$
$\mathsf{ctr}_{\mathsf{recip}} \leftarrow \mathsf{ctr}_{\mathsf{recip}} + 1$
$\mathsf{epk}_{\mathsf{RID}} \leftarrow \mathsf{PKGen}(\mathsf{esk})$
return $\mathsf{ct}_{\mathsf{RID}} \leftarrow \mathsf{Enc}(\mathsf{pk}, (\mathsf{pk}, \mathsf{epk}_{\mathsf{RID}}, \mathsf{ctr}_{\mathsf{recip}}))$

On input* ("send", $\mathsf{ct}_{\mathsf{Signal}}$)

$X \leftarrow \mathsf{Dec}(\mathsf{esk}, \mathsf{ct}_{\mathsf{Signal}})$
if $X = \perp$ **then** ▷ Not correct key, do dummy reads and writes.
  $\mathsf{data} \leftarrow \mathsf{ORAM.Read}(\mathsf{DBrecip}, \cdot, \cdot)$
    ▷ Read to whatever value.
  $\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DBrecip}, \cdot, \cdot)$
  $\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DB}, \cdot, \cdot)$
    ▷ Dummy writes can be writing garbage value to some dummy slot.
else
  Parse X as $[\mathsf{RID}, \mathsf{loc}]$
  $\mathsf{data} \leftarrow \mathsf{ORAM.Read}(\mathsf{DBrecip}, \mathsf{RID}, \cdot)$ ▷ $\cdot$ is a dummy data variable
  $[\mathsf{PrevIdx}, \mathsf{PrevLoc}] \leftarrow \mathsf{Dec}(\mathsf{esk}, \mathsf{data})$
  $\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DBrecip}, \mathsf{RID}, \mathsf{Enc}(\mathsf{esk}, [\mathsf{CurIdx}, \mathsf{loc}]))$
  $\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DB}, \mathsf{CurIdx}, \mathsf{Enc}(\mathsf{esk}, [\mathsf{PrevIdx}, \mathsf{PrevLoc}]))$
$\mathsf{CurIdx} \leftarrow \mathsf{CurIdx} + 1$
return CurIdx.

Figure 7: Program Prog run by $\mathcal{G}_{\mathrm{att}}$ for multi-server setting

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 $(\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i})$ to Srv.
4. Await $(eid, \mathsf{ct}_{\mathsf{RID}}, \sigma_T)$ from Srv
5. $(\mathsf{pk}, \mathsf{epk}_{\mathsf{RID}}) \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathsf{RID}})$
6. Assert $\Sigma.\mathsf{Ver}_{\mathsf{mpk}}((eid, (\mathsf{pk}, \mathsf{epk}_{\mathsf{RID}}, \mathsf{ctr}_{\mathsf{recip}})), \sigma_T) = 1$ and publish $(\mathsf{pk}_i, \mathsf{epk}_{\mathsf{RID}})$. Initialize $\mathsf{ctr}_i = 0$.

   ▷ Note that if the returned pk_i is not the pk_i sent, abort.

Srv:

1. Upon receiving $(\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i})$ from R_i, let $((\mathsf{pk}_i), \sigma_T) = \mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i}))$. Upon receiving ORAM request $\mathsf{data} \leftarrow \mathsf{ORAM.Access}(\cdot,\cdot,\cdot,\cdot)$ from TEE when executing the $\mathcal{G}_{\text{att}}.\mathsf{resume}(\cdot,\cdot)$ operation: process and return data to $\mathcal{G}_{\text{att}}$.
2. Send $((eid, \mathsf{ct}_{\mathsf{RID}}), \sigma_T)$ to R_i.
3. Upon return of $\mathsf{ctr}_{\mathsf{recip}}$ from TEE, if $\mathsf{ctr}_{\mathsf{recip}} = M$, call procedure $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“extendrecip”}, 2M)$ and update $M \leftarrow 2M$.

Procedure $(\mathtt{SEND}, R_i, \mathsf{loc})$ Sender S:

1. Sender S obtains $(\mathsf{pk}, \mathsf{epk}_{\mathsf{RID}})$ from recipient R_i (which is published)
2. Sender S computes $\mathsf{ct}_{\mathsf{Signal}} = \mathsf{Enc}(\mathsf{epk}_{\mathsf{RID}}, [h(\mathsf{pk}), \mathsf{loc}])$. S sends $(\mathtt{SUBMIT}, (\mathtt{SEND}, \mathsf{ct}_{\mathsf{Signal}}))$ to $\mathcal{G}_{\mathsf{ledger}}$

Srv:

1. Send $\mathtt{READ}$ to $\mathcal{G}_{\mathsf{ledger}}$
2. Upon receiving $(\mathtt{SEND}, \mathsf{ct}_{\mathsf{Signal}})$: Call $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“send”}, \mathsf{ct}_{\mathsf{Signal}}))$. Upon receiving ORAM requests $\mathsf{data} \leftarrow \mathsf{ORAM.Access}(\cdot,\cdot,\cdot,\cdot)$: process and return data to $\mathcal{G}_{\text{att}}$.
3. Upon return of CurIdx from TEE. If $\mathsf{CurIdx} = N$, call procedure $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“extend”}, 2N)$ and update $N \leftarrow 2N$.

Figure 8: The protocol Π_{TEE} for private signaling in the $\mathcal{G}_{\text{att}}$ hybrid world for multi-server setting.

9 Conclusion

In this work, we have presented a scalable private signaling protocol that is asymptotically superior and concretely orders of magnitude faster compared to prior works. More specifically, to process N sent signals and M retrievals of each retrieving $\bar{\ell}$ signals, the cost of the TEE is $\tilde{O}(N + M\ell)$, which is almost optimal except for some logarithmic factor. Moreover, our construction also enjoys the property of retrieval privacy which was not considered in previous works. Lastly, we show how to extend our construction to allow multiple servers without sacrificing runtime or privacy.

References

[1] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, "Dissent in numbers: Making strong anonymity scale," in *OSDI 12*. USENIX, Oct. 2012, pp. 179–182.

[2] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An anonymous messaging system handling millions of users," in *2015 IEEE S&P*, 2015, pp. 321–338.

[3] J. Lund, "Technology preview: Sealed sender for signal," https://signal.org/blog/sealed-sender/, Oct. 2018.

[4] A. Bittau, Ú. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld, "Prochlo: Strong privacy for analytics in the crowd," in *SOSP*, 2017, pp. 441–459.

[5] G. Beck, J. Len, I. Miers, and M. Green, "Fuzzy message detection," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1507–1528. [Online]. Available: https://doi.org/10.1145/3460120.3484545

[6] N. T. Courtois and R. Mercer, "Stealth address and key management techniques in blockchain systems." *ICISSP*, vol. 2017, pp. 559–566, 2017.

[7] S. Noether, "Ring signature confidential transactions for monero." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1098, 2015.

[8] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE S&P*, 2014, pp. 459–474.

[9] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash Protocol Specification Version 2021.2.14," https://github.com/zcash/zips/blob/master/protocol/protocol.pdf.

[10] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "Zexe: Enabling decentralized private computation," in *2020 IEEE S&P (SP)*, 2020, pp. 947–964.

[11] V. Madathil, A. Scafuro, I. A. Seres, O. Shlomovits, and D. Varlakov, "Private signaling," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3309–3326. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/madathil

[12] Z. Liu and E. Tromer, "Oblivious message retrieval," in *Advances in Cryptology – CRYPTO 2022*, Y. Dodis and T. Shrimpton, Eds. Cham: Springer Nature Switzerland, 2022, pp. 753–783.

[13] J. Grigg and D. Hopwood, "Zcash improvement proposal 307: Light client protocol for payment detection," https://zips.z.cash/zip-0307, Sep. 2018.

[14] Z. Liu, E. Tromer, and Y. Wang, "Group oblivious message retrieval," Cryptology ePrint Archive, Paper 2023/534, 2023, https://eprint.iacr.org/2023/534. [Online]. Available: https://eprint.iacr.org/2023/534

[15] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.

[16] "Our source code," https://anonymous.4open.science/r/sgx-sps-3CF8, 2024.

[17] S. Sasy, S. Gorbunov, and C. W. Fletcher, "Zerotrace : Oblivious memory primitives from intel SGX," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[18] V. Madathil, A. Scafuro, I. A. Seres, O. Shlomovits, and D. Varlakov, "Private signaling, version 20210624:145011," *Cryptology ePrint Archive*, 2021.

[19] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *J. ACM*, vol. 43, no. 3, p. 431–473, may 1996. [Online]. Available: https://doi.org/10.1145/233551.233553

[20] E. Stefanov, M. V. Dijk, E. Shi, T.-H. H. Chan, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: An extremely simple oblivious RAM protocol," *J. ACM*, vol. 65, no. 4, apr 2018. [Online]. Available: https://doi.org/10.1145/3177872

[21] G. Asharov, I. Komargodski, W.-K. Lin, and E. Shi, "Oblivious RAM with worst-case logarithmic overhead," in *Advances in Cryptology – CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part IV*. Berlin, Heidelberg: Springer-Verlag, 2021, p. 610–640. [Online]. Available: https://doi.org/10.1007/978-3-030-84259-8_21

[22] K. Hashimoto, S. Katsumata, and T. Prest, "How to hide metadata in mls-like secure group messaging: Simple, modular, and post-quantum," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2022, p. 1399–1412.

[23] K. Wüst, S. Matetic, M. Schneider, I. Miers, K. Kostiainen, and S. Čapkun, "Zlite: Lightweight clients for shielded zcash transactions using trusted execution," in *Financial Cryptography and Data Security*, I. Goldberg and T. Moore, Eds. Cham: Springer International Publishing, 2019, pp. 179–198.

[24] S. J. Lewis, "Discreet log #1: Anonymity, bandwidth and Fuzzytags," Feb 2021. [Online]. Available: https://openprivacy.ca/discreet-log/01-anonymity-bandwidth-and-fuzzytags/

[25] I. A. Seres, B. Pejó, and P. Burcsi, "The effect of false positives: Why fuzzy message detection leads to fuzzy privacy guarantees?" in *Financial Cryptography and Data Security*, I. Eyal and J. Garay, Eds. Cham: Springer International Publishing, 2022, pp. 123–148.

[26] C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas, "Bitcoin as a transaction ledger: A composable treatment," in *Annual international cryptology conference*. Springer, 2017, pp. 324–356.

[27] R. Pass, E. Shi, and F. Tramer, "Formal abstractions for attested execution secure processors," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 260–289.

[28] AWS, "AWS Nitro Enclaves." [Online]. Available: https://aws.amazon.com/ec2/nitro/nitro-enclaves/

[29] Intel, "Intel Software Guard Extensions (Intel SGX)." [Online]. Available: https://software.intel.com/en-us/sgx

[30] AMD, "AMD Secure Encrypted Virtualization SEV." [Online]. Available: https://www.amd.com/en/developer/sev.html

[31] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 2015, pp. 640–656.

[32] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *11th USENIX Workshop on Offensive Technologies, WOOT 2017, Vancouver, BC, Canada, August 14-15, 2017*, W. Enck and C. Mulliner, Eds. USENIX Association, 2017.

[33] S. Lee, M. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring fine-grained control flow inside SGX enclaves with branch shadowing," in *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, E. Kirda and T. Ristenpart, Eds. USENIX Association, 2017, pp. 557–574.

[34] D. Lee, D. Jung, I. T. Fang, C. Tsai, and R. A. Popa, "An off-chip attack on hardware enclaves via the memory bus," in *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, S. Capkun and F. Roesner, Eds. USENIX Association, 2020, pp. 487–504.

[35] S. Dinesh, G. Garrett-Grossman, and C. W. Fletcher, "Synthct: Towards portable constant-time code," in *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022.

[36] P. Mishra, R. Poddar, J. Chen, A. Chiesa, and R. A. Popa, "Oblix: An efficient oblivious search index," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 279–296.

[37] A. Ahmad, K. Kim, M. I. Sarfaraz, and B. Lee, "OBLIVIATE: A data oblivious filesystem for intel SGX," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[38] T. Moataz, T. Mayberry, E. Blass, and A. H. Chan, "Resizable tree-based oblivious RAM," in *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 8975. Springer, 2015, pp. 147–167.

[39] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The second-generation onion router," 2004, pp. 303–320.

[40] Intel, "Intel Software Guard Extensions for linux* os." [Online]. Available: https://github.com/intel/linux-sgx

[41] OpenSSL, "OpenSSL Cryptography and SSL/TLS Toolkit." [Online]. Available: https://www.openssl.org/

[42] Intel, "Intel Software Guard Extensions SSL." [Online]. Available: https://github.com/intel/intel-sgx-ssl

[43] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern, "Rsa-oaep is secure under the rsa assumption," in *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '01. Berlin, Heidelberg: Springer-Verlag, 2001, p. 260–274.

[44] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval, "Key-privacy in public-key encryption," in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 566–582.

[45] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[46] R. Cramer and V. Shoup, "Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack," *SIAM Journal on Computing*, vol. 33, no. 1, pp. 167–226, 2003. [Online]. Available: https://doi.org/10.1137/S0097539702403773

[47] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, sep 2009. [Online]. Available: https://doi.org/10.1145/1568318.1568324

A Proof

A.1 Extended Preliminaries

Key-private Public Key Encryption scheme. We also require a correct and semantically secure PKE scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ that is also key-private. Correctness and semantic security are both standard. Key-privacy, introduced in [44], basically means that if a ciphertext is encrypted under a public key, one cannot tell which public key was used. This is achieved by most of the commonly used encryption schemes including El Gamal[45], Cramer-Shoup[46], a variant of RSA-OAEP suggested in [44], LWE [47] and so on. We formally capture this property as follows, adapted from [44]:

Definition 2 (Key-privacy)**.** *A PKE scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is key-private if for any PPT adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*, it can only win the following game with probability* $\leq 1/2 + \mathsf{negl}(\lambda)$*:* $(\mathsf{pk}_0, \cdot) \leftarrow \mathsf{KeyGen}(\lambda)$*,* $(\mathsf{pk}_1, \cdot) \leftarrow \mathsf{KeyGen}(\lambda)$*,* $(x, s) \leftarrow \mathcal{A}_0(\mathsf{pk}_0, \mathsf{pk}_1), y \leftarrow \mathsf{Enc}(\mathsf{pk}_b, x)$ *where* $b \leftarrow_{\$} \{0, 1\}$*,* $b' \leftarrow \mathcal{A}_1(y, s)$*, and the adversary wins iff* $b' = b$*.*

PKE scheme with two more properties. For our extension to multiple servers, we require two more properties of the PKE scheme: wrong-key-decryption-detection and public-key-unlinkability.

At a high level, wrong key decryption detection basically means that one can tell whether the key used to decrypt the ciphertext is correct or now.

Definition 3 (Wrong-key Decryption Detection)**.** *A PKE scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is wrong-key-decryption-detectable if any* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$ *and* $\mathsf{pk}', \mathsf{sk}' \leftarrow \mathsf{KeyGen}$ *and any message* x*, it holds that* $\Pr[\mathsf{Dec}(\mathsf{sk}', \mathsf{Enc}(\mathsf{pk}, x)) = \perp] \geq 1 - \mathsf{negl}(\lambda)$*.*

This property are very easy to achieve and are achieved by lots of schemes (e.g., trial decryption with key-exchange used in Zcash [13]), without only a very tiny overhead, if any.

Public key unlinkability means that two public keys generated using the same secret key are indistinguishable from two public keys generated using two different secret keys. We formally capture the property with the following definition.

Definition 4 (Public key unlinkability)**.** *A PKE scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is public-key-unlinkable if it furthermore has an interface* $\mathsf{pk} \leftarrow \mathsf{PKGen}(\mathsf{sk})$ *such that: let* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$*,* $\mathsf{pk}' \leftarrow \mathsf{PKGen}(\mathsf{sk})$*,* $(\mathsf{pk}'', \mathsf{sk}'') \leftarrow \mathsf{KeyGen}$*, it holds that* $(\mathsf{pk}, \mathsf{pk}') \approx_c (\mathsf{pk}, \mathsf{pk}'')$ *(and PKE correctness and semantic security hold with respect to* pk'*).*

This property is is widely achieved by commonly used PKE schemes like Regev [47] El-Gamal [45], and so on.

A.2 Universal Composability

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

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

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

A.3 Proof of Security of Theorem 1

Theorem. *Assume that the signature scheme Σ is existentially unforgeable under chosen message attacks, the encryption scheme* Enc *is CPA secure and key-private, and the underlying ORAM is correct and private. 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. To prove that Π_{TEE} UC-realizes $\mathcal{F}_{\mathsf{privSignal}}$ we show that there exists a simulator $\mathcal{S}$ that interacts with $\mathcal{F}_{\mathsf{privSignal}}$ and the adversary $\mathcal{A}$ to generate a transcript that is indistinguishable from the real world protocol. We consider the following cases of corruption:

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

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

A.3.1 Case 1: Neither S nor R is corrupt and only $\mathcal{S}_N$ is corrupt

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

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

- $\mathbf{Hyb}_0$ The real world protocol.

The simulator $\mathcal{S}$ internally simulates $\mathcal{G}_{\text{att}}$ and $\mathcal{G}_{\mathsf{ledger}}$ towards the adversary $\mathcal{A}$

Registration For each recipient R_i:

1. Compute $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Enc.KeyGen}(1^\lambda)$, $(\Sigma.\mathsf{sk}_i, \Sigma.\mathsf{vk}_i) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$.
2. Set $\mathsf{ct}_{\mathsf{keys},i} = \mathsf{Enc}(\mathsf{epk}_i, (\mathsf{pk}_i, \Sigma.\mathsf{vk}_i))$ and send $(\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i})$ to $\mathcal{A}$ (on behalf of Srv).
3. Upon receiving $(\text{“setup”}, \mathsf{ct}_{\mathsf{keys}})$ on behalf of $\mathcal{G}_{\text{att}}$, send $\mathsf{ORAM.Write}(\mathsf{DBkeys}, \cdot, \cdot, \cdot)$ and $\mathsf{ORAM.Write}(\mathsf{DBrecip}, \cdot, \cdot, \cdot)$ on behalf of $\mathcal{G}_{\text{att}}$ to $\mathcal{A}$.
4. Upon receiving $((eid, \mathsf{pk}_i, \mathsf{RID}), \sigma)$, abort with $\mathsf{sigFailure}_1$ if σ would be validated but the following communication was not recorded: $((\mathsf{pk}_i), \sigma) = \mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i})$.
5. Else publish pk_i and set $\mathsf{index}_i = 0, \mathsf{ctr}_i = 0$ and $\vec{L} = \{0\}_{j=0}^{\bar{\ell}}$.

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

1. Compute $\mathsf{ct}_{\mathsf{Signal}} = \mathsf{Enc}(\mathsf{epk}, 0)$ and send $(\texttt{SUBMIT}, (\texttt{SEND}, \mathsf{ct}_{\mathsf{Signal}}))$ to $\mathcal{G}_{\mathsf{ledger}}$.
2. Upon receiving $(\text{“send”}, \mathsf{ct}_{\mathsf{Signal}})$ on behalf of $\mathcal{G}_{\text{att}}$ from $\mathcal{A}$:
   (a) Send $\mathsf{ORAM.Read}(\mathsf{DBrecip}, \cdot, \cdot)$ to $\mathcal{A}$
   (b) Send $\mathsf{ORAM.Write}(\mathsf{DBrecip}, \cdot, \cdot, \cdot)$ to $\mathcal{A}$
   (c) Send $\mathsf{ORAM.Write}(\mathsf{DB}, \cdot, \cdot, \cdot)$ to $\mathcal{A}$
3. Send $(\texttt{SEND}, \texttt{ok})$ to $\mathcal{F}_{\mathsf{privSignal}}$.

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

1. Compute $\sigma_i = \mathsf{Enc}(\mathsf{epk}, 0)$ and send $(\texttt{RECEIVE}, \sigma_i)$ to $\mathcal{A}$.
2. Upon receiving $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“receive”}, \sigma_i))$ on behalf of $\mathcal{G}_{\text{att}}$:
   (a) Send $\mathsf{ORAM.Read}(\mathsf{DBkeys}, \cdot, \cdot)$ to $\mathcal{A}$
   (b) Send $\mathsf{ORAM.Read}(\mathsf{DBrecip}, \cdot, \cdot)$ to $\mathcal{A}$
   (c) For $k \in [\bar{\ell}]$: Send $\mathsf{ORAM.Read}(\mathsf{DBrecip}, \cdot, \cdot, \cdot)$ to $\mathcal{A}$
   (d) Send $\mathsf{ORAM.Write}(\mathsf{DBrecip}, \cdot, \cdot)$ to $\mathcal{A}$.
   (e) Sample a random pk^* and compute $\vec{\mathsf{ct}}_{\mathsf{loc},i} = \{\mathsf{Enc}(\mathsf{pk}^*, 0)\}_{k \in [\bar{\ell}]}$
   (f) Compute $\sigma_T = \mathsf{Sig}(\mathsf{msk}, (eid, \vec{\mathsf{ct}}_{\mathsf{loc},i}))$
   (g) Return $(\vec{\mathsf{ct}}_{\mathsf{loc},i}, \sigma_T)$ to $\mathcal{A}$
3. Receive $(eid, \vec{\mathsf{ct}}_{\mathsf{loc},i}, \sigma_T)$ from $\mathcal{A}$; if received w/o communication with $\mathcal{G}_{\text{att}}$, abort with $\mathsf{sigFailure}_2$.
4. Else send $(\texttt{RECEIVE}, \texttt{ok})$ to $\mathcal{F}_{\mathsf{privSignal}}$.

`READ`: Upon receiving (`READ`) from $\mathcal{A}$, forward (`READ`) to $\mathcal{G}_{\mathsf{ledger}}$ and return whatever is returned.

`ExtendDB`: Upon receiving `ExtendDB` from $\mathcal{A}$, initialize a database DB', repeat N times: $\mathsf{ORAM.Write}(\mathsf{DB}', \cdot, \cdot)$, set $\mathsf{DB} \leftarrow \mathsf{DB}'$ and simply set $\mathsf{pp}.N$ as the received N.

`ExtendRecip`: Upon receiving `ExtendRecip` from $\mathcal{A}$, initialize empty databases $\mathsf{DBrecip}'$ and DBkeys' and call $\mathsf{ORAM.Write}(\mathsf{DBrecip}', \cdot, \cdot)$ and $\mathsf{ORAM.Write}(\mathsf{DBkeys}', \cdot, \cdot)$, N times.

Figure 9: Simulator $\mathcal{S}_N$ for the case of only one corrupt server

The simulator $\mathcal{S}$ internally simulates $\mathcal{G}_{\text{att}}$ towards the adversary $\mathcal{A}$

Setup For each recipient R_i: Same as in Fig 9

`SEND`: Upon receiving $(\texttt{SEND}, S_i)$ from $\mathcal{F}_{\mathsf{privSignal}}$, same as in Fig 9. //`(Honest send)`
Upon receiving $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid_j, (\text{“send”}, \mathsf{ct}_{\mathsf{Signal}}))$ on behalf of $\mathcal{G}_{\text{att}}$ for a $\mathsf{ct}_{\mathsf{Signal}}$ that was not created by the simulator. //`(Malicious send)`

Let $(\mathsf{RID}, \mathsf{loc}) = \mathsf{Dec}(\mathsf{esk}, \mathsf{ct}_{\mathsf{Signal}})$.
if RID corresponds to some R_j **then**
Send $(\texttt{SEND}, R_j, \mathsf{loc})$ to $\mathcal{F}_{\mathsf{privSignal}}$ on behalf of $\mathcal{A}$ and receive $(\texttt{SEND})$ and send back $(\texttt{SEND}, \texttt{ok})$.
Send $\mathsf{data} \leftarrow \mathsf{ORAM.Read}(\mathsf{DBrecip}, \mathsf{RID}, \cdot)$ to $\mathcal{A}$
Send $\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DBrecip}, \cdot, \cdot))$ to $\mathcal{A}$
Send $\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DB}, \cdot, \cdot))$ to $\mathcal{A}$

`RECEIVE` Same as in Fig 9

`SUBMIT` : Upon receiving a `SUBMIT` request from $\mathcal{A}$, forward to $\mathcal{G}_{\mathsf{ledger}}$. //`(Malicious write to` $\mathcal{G}_{\mathsf{ledger}}$`)`

`READ` Same as in Fig 9

Figure 10: Simulator $\mathcal{S}_s$ for the case corrupt server and sender

- $\mathbf{Hyb}_1$ is the same as $\mathbf{Hyb}_0$ except that upon receiving a `SEND` command, the $\mathsf{ct}_{\mathsf{Signal}}$ is replaced with encryption to 0 instead of the actual location. By the CPA security of the underlying encryption scheme the two hybrids are indistinguishable.
- $\mathbf{Hyb}_2$ is the same as $\mathbf{Hyb}_1$ except that all the ORAM calls are now replaced by dummy ORAM calls as in the simulation. By the computational privacy property of the ORAM scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_3$ is the same as $\mathbf{Hyb}_2$ except that in the `RECEIVE` command, the simulator samples a new random public key pk^* and encrypts the locations under this public key. By the key-privacy property of the underlying encryption scheme, these two hybrids are indistinguishable.
- $\mathbf{Hyb}_4$ is the same as $\mathbf{Hyb}_5$ except that in the `RECEIVE` command, the simulator returns encryptions of 0 as $\mathsf{ct}_{\mathsf{loc},i}$ to the server on behalf of $\mathcal{G}_{\text{att}}$. By the CPA security of the underlying encryption scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_5$ is the same as $\mathbf{Hyb}_4$ except that in the **Setup** procedure, the simulator aborts with $\mathsf{sigFailure}_1$. Since we use EUF-CMA signatures, this occurs with negligible probability.
- $\mathbf{Hyb}_6$ is the same as $\mathbf{Hyb}_5$ except that in the `RECEIVE` command, the simulator may abort with $\mathsf{sigFailure}_2$. This occurs with negligible probability since we use EUC-CMA signatures.

Note that $\mathbf{Hyb}_6$ is exactly the same as the ideal world. Therefore through a sequence of hybrids we have proved that the real and the ideal worlds are indistinguishable.

A.3.2 Case 2: S and Srv are corrupt

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

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

- $\mathbf{Hyb}_0$ is the real world.

- $\mathbf{Hyb}_1$ is the same as $\mathbf{Hyb}_0$ except that for an honest sender, encryptions of 0 are sent instead of the actual locations in the `SEND` command. By the CPA security of the underlying encryption scheme, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are indistinguishable.
- $\mathbf{Hyb}_2$ is the same as $\mathbf{Hyb}_1$ except that all the ORAM calls are now replaced by dummy ORAM calls as in the simulation. By the computational privacy property of the ORAM scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_3$ is the same as $\mathbf{Hyb}_2$ except that for `RECEIVE` command, the simulator returns encryptions of 0 to the server as $\mathcal{G}_{\text{att}}$. By the CPA security of the underlying encryption scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_4$ is the same as $\mathbf{Hyb}_3$ except that the "setup" of **Setup** procedure is done as in the simulation and the simulator might abort with $\mathsf{sigFailure}_1$. By the EUF-CMA property of the signature scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_5$ is the same as $\mathbf{Hyb}_4$ except that the `RECEIVE` is done as in the simulation and the simulator might abort with $\mathsf{sigFailure}_2$. By the EUF-CMA property of the signature scheme, the two hybrids are indistinguishable.

A.3.3 Case 3: Srv and R are corrupt

Simulator overview In this case, the simulator needs to simulate a view towards a malicious recipient that is indistinguishable from the real world interaction. For malicious recipients, the simulator is notified of their `RECEIVE` request since the Srv must invoke the $\mathcal{G}_{\text{att}}$ functionality to get . Next the simulator needs to to send the correct locations to the recipient, even though the simulated $\vec{L}$ are just an encryption of 0s. To this end, the simulator simply sends the `RECEIVE` command to the $\mathcal{F}_{\mathsf{privSignal}}$ functionality and learns the locations that correspond to the malicious recipient. It then simulates the $\mathcal{G}_{\text{att}}$ functionality to compute an encryption of the locations and sends it back to the Srv.

Proof by hybrids
- $\mathbf{Hyb}_0$ The real world protocol.
- $\mathbf{Hyb}_1$ is the same as $\mathbf{Hyb}_0$ except that upon receiving a `SEND` command, the $\mathsf{ct}_{\mathsf{Signal}}$ is replaced with an encryption to 0 instead of the actual location. By the CPA security of the underlying encryption scheme the two hybrids are indistinguishable.
- $\mathbf{Hyb}_2$ is the same as $\mathbf{Hyb}_1$ except that all the ORAM calls are now replaced by dummy ORAM calls as in the simulation. By the computational privacy property of the ORAM scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_3$ is the same as $\mathbf{Hyb}_2$ except that in the `RECEIVE` command for an honest recipient, the simulator returns encryptions of 0 as $\mathsf{ct}_{\mathsf{loc},i}$ to the server on behalf of $\mathcal{G}_{\text{att}}$. By the CPA security of the underlying encryption scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_4$ is the same as $\mathbf{Hyb}_3$, except that for malicious recipients, the simulator may abort with sigFailure. Since we assume EUF-CMA signatures these two hybrids are indistinguishable.
- $\mathbf{Hyb}_5$ is the same as $\mathbf{Hyb}_4$ except that in the **Setup** procedure, the simulator aborts with $\mathsf{sigFailure}_1$. Because of EUF-CMA signatures this occurs with negl probability.
- $\mathbf{Hyb}_6$ is the same as $\mathbf{Hyb}_5$ except that in the `RECEIVE` command, the simulator may abort with $\mathsf{sigFailure}_2$. Because of EUF-CMA signatures this occurs with negl probability.

A.3.4 Case 4: Corrupt Srv, S and R

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

Proof by hybrids

The simulator $\mathcal{S}$ internally simulates $\mathcal{G}_{\text{att}}$ towards the adversary $\mathcal{A}$

Setup For each honest recipient R_i: Same as in Fig 9
//(`Malicious receiver`):
Upon receiving $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“setup”}, \mathsf{ct}_{\mathsf{keys},i})$ from Srv on behalf of $\mathcal{G}_{\text{att}}$:
1. Compute $(\mathsf{pk}_i, \Sigma.\mathsf{vk}_i) = \mathsf{Dec}(\mathsf{esk}, \mathsf{ct}_{\mathsf{keys},i})$ and compute pk_i from sk_i.
2. Set $\vec{L} = \{0\}_{j=0}^{\bar{\ell}}$
3. Set $\mathsf{index} = 0$ and $\mathsf{ctr} = 0$
4. Compute $\sigma = \Sigma.\mathsf{Sig}(\mathsf{msk}, (eid, (\mathsf{pk})))$ and send (pk, σ) to $\mathcal{A}$ and store $(\mathsf{pk}_i, \Sigma.\mathsf{vk}_i, \mathsf{index}_i, \mathsf{ctr}_i)$.

`SEND`: Same as in Fig 9

`RECEIVE` For honest recipients, same as in Fig 9.
// `(For malicious recipients)`
1. Receive $\mathcal{G}_{\text{att}}.\mathsf{resume}(eid, (\text{“receive”}, \mathsf{ctr}_i, \sigma_i))$ from Srv on behalf of $\mathcal{G}_{\text{att}}$. If $\mathsf{Sig.Ver}(\Sigma.\mathsf{vk}_i, \mathsf{ctr}_i, \sigma_i) = 0$, return $\perp$. Else if i corresponds to that of an honest recipient, abort with $\mathsf{sigFailure}$. Else:
2. Send $(\texttt{RECEIVE}, R_i)$ to $\mathcal{F}_{\mathsf{privSignal}}$ on behalf of R_i and get back $(\texttt{RECEIVE})$ from $\mathcal{F}_{\mathsf{privSignal}}$. Send $(\texttt{RECEIVE}, \texttt{ok})$ to $\mathcal{F}_{\mathsf{privSignal}}$ and get back $\vec{R} = [\mathsf{loc}_1 \dots \mathsf{loc}_k]$.
3. Send $\mathsf{ORAM.Read}(\mathsf{DBrecip}, \mathsf{RID}, \cdot)$ to the adversary.
4. Send $\bar{\ell}$ number of calls of $\mathsf{ORAM.Read}(\mathsf{DB}, \mathsf{RID}, \cdot)$ to the server.
5. If $k > \bar{\ell}$: append -2 to $\vec{R}$, else if $k = \bar{\ell}$, append -1 to $\vec{R}$, else for $j \in [k+1, \bar{\ell}]$, write 0 to $\vec{R}[j]$.
6. Run $\cdot \leftarrow \mathsf{ORAM.Write}(\mathsf{DBkeys}, \mathsf{RID}, (\Sigma.\mathsf{vk}, \mathsf{pk}, \mathsf{ctr}_{\mathsf{RID}}))$
7. Compute $\mathsf{ct}_{\mathsf{loc},i} = (\mathsf{Enc}(\mathsf{pk}_i, \mathsf{loc}_1) \dots \mathsf{Enc}(\mathsf{pk}_i, \mathsf{loc}_{\bar{\ell}}))$
8. Compute $\sigma_T = \Sigma.\mathsf{Sig}(\mathsf{msk}, (\mathsf{ct}_{\mathsf{loc},i}))$ and send $(\mathsf{ct}_{\mathsf{loc},i}, \sigma_T)$ to $\mathcal{A}$.

`READ` Same as in Fig 9

Figure 11: Simulator $\mathcal{S}_r$ for the case corrupt server and recipients

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

Setup
For each recipient R_i: Same as in Fig 11

`SUBMIT` : Same as in Fig 10

`SEND`: Same as in Fig 10

`RECEIVE` Same as in Fig 11

`READ` Same as in Fig 9

Figure 12: Simulator $\mathcal{S}_{sr}$ for the case corrupt server, senders and recipients

- $\mathbf{Hyb}_0$ is the real world.
- $\mathbf{Hyb}_1$ is the same as $\mathbf{Hyb}_0$ except that for an honest sender, encryptions of 0 are sent instead of the actual locations in the `SEND` command. By the CPA security of the underlying encryption scheme, $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_0$ are indistinguishable.
- $\mathbf{Hyb}_2$ is the same as $\mathbf{Hyb}_1$ except that all the ORAM calls are now replaced by dummy ORAM calls as in the simulation. By the computational privacy property of the ORAM scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_3$ is the same as $\mathbf{Hyb}_2$ except that for `RECEIVE` command, the simulator returns encryptions of 0 to the server as $\mathcal{G}_{\text{att}}$. By the CPA security of the underlying encryption scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_4$ is the same as $\mathbf{Hyb}_3$, except that for malicious recipients, the simulator may abort with $\mathsf{sigFailure}$. Since we assume EUF-CMA signatures these two hybrids are indistinguishable.
- $\mathbf{Hyb}_5$ is the same as $\mathbf{Hyb}_4$ except that the "setup" of **Setup** procedure is done as in the simulation and the simulator might abort with $\mathsf{sigFailure}_1$. By the EUF-CMA property of the signature scheme, the two hybrids are indistinguishable.
- $\mathbf{Hyb}_6$ is the same as $\mathbf{Hyb}_5$ except that the `RECEIVE` is done as in the simulation and the simulator might abort with $\mathsf{sigFailure}_2$. By the EUF-CMA property of the signature scheme, the two hybrids are indistinguishable.

B Hybrid Functionalities

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

Figure 13: Abridged $\mathcal{G}_{\mathsf{ledger}}$ 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 in Fig. 13. For the complete description of the functionality, we refer the reader to Pages 339-340 of [26].

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

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

Enclave operations

//local interface – install an enclave.
On receive* install(idx, Prog) from some $\mathcal{P} \in \mathsf{reg}$:
- if $\mathcal{P}$ is honest, assert $idx = sid$
- generate nonce $eid \in \{0,1\}^\lambda$, store $T[eid, \mathcal{P}] := (idx, \mathsf{Prog}, \vec{0})$, send eid to $\mathcal{P}$.

//local interface – resume an enclave.
On receive* resume(eid, inp) from some $\mathcal{P} \in \mathsf{reg}$:
- let $(idx, \mathsf{Prog}, \mathsf{mem}) := T(eid, \mathcal{P})$, abort if not found
- let $(\mathsf{outp}, \mathsf{mem}) := \mathsf{Prog}(\mathsf{inp}, \mathsf{mem})$, update $T[eid, \mathcal{P}] := (idx, \mathsf{Prog}, \mathsf{mem})$
- let $\sigma := \Sigma.\mathsf{Sig}_{\mathsf{msk}}(idx, eid, \mathsf{Prog}, \mathsf{outp})$ and send (outp, σ) to $\mathcal{P}$.

Figure 14: Global functionality modeling a TEE [27]