

HomeRun: High-efficiency Oblivious Message Retrieval, Unrestricted

Yanxue Jia¹, Varun Madathil², and Aniket Kate^{1,3}

¹Purdue University

²North Carolina State University

³Supra Research

Abstract

In the realm of privacy-preserving blockchain applications such as Zcash, oblivious message retrieval (OMR) enables recipients to privately access messages directed to them on blockchain nodes (or bulletin board servers). OMR prevents servers from linking a message and its corresponding recipient’s address, thereby safeguarding recipient privacy. Several OMR schemes have emerged recently to meet the demands of these privacy-centric blockchains; however, we observe that existing solutions exhibit shortcomings in various critical aspects and may only achieve certain objectives inefficiently, sometimes relying on trusted hardware, thereby impacting their practical utility. This work introduces a novel OMR protocol, HomeRun, that leverages two semi-honest, non-colluding servers to excel in both performance and security attributes as compared to the current state-of-the-art.

HomeRun stands out by providing unlinkability across multiple requests for the same recipient’s address. Moreover, it does not impose a limit on the number of pertinent messages that can be received by a recipient, which thwarts “message balance exhaustion” attacks and enhances system usability. HomeRun also empowers servers to regularly delete the retrieved messages and the associated auxiliary data, which mitigates the constantly increasing computation costs and storage costs incurred by servers. Remarkably, none of the existing solutions offer all of these features collectively. Finally, thanks to its judicious use of highly efficient cryptographic building blocks, HomeRun is highly performant: Specifically, the total runtime of servers in HomeRun is $3830\times$ less than that in the work by Liu et al. (CRYPTO ’22) based on fully-homomorphic encryption, and at least $1459\times$ less than that in the design by Madathil et al. (USENIX Security ’22) based on two semi-honest and non-colluding servers, using a single thread in a WAN setting.

1 Introduction

Safeguarding “metadata”—that is, information concerning who communicated with whom, when, and the extent of their interactions [SG23]—over the Internet poses a formidable challenge. This challenge is notably pronounced in the context of blockchains, where message senders publicly post their messages on a bulletin board (i.e. a blockchain) and recipients retrieve their messages by contacting the blockchain nodes/servers. This process presents an opportunity for various intermediaries to potentially link senders to recipients by tracking these accesses [HHK18]. While senders may employ techniques like anonymous broadcast [LK23, LSD23, NSD22, APY20] to hide their identities, such methods are ineffective for preserving the privacy of the intended message recipients, especially when confronted with adversaries attempting to associate a particular recipient (typically identified by an address or wallet) with a message intended for them.

A straightforward approach to solving the problem is to let full nodes (namely, servers) assist recipients in identifying pertinent messages on the bulletin board. Unfortunately, if recipients share their addresses (namely, `Addr`) with the servers, their privacy would be compromised. On the other hand, recipients can potentially download the bulletin board and read their messages locally, without revealing which messages

are pertinent to them. However, as blockchains grow in size, it is very impractical for recipients to download or maintain a massive bulletin board,¹ merely to read a few pertinent messages. This work focuses on protecting the privacy of blockchain/bulletin-board recipients while allowing efficient and scalable retrieval of pertinent messages.

Hearn and Corallo [HC12] were the first to address the problem by leveraging the false-positive probability of Bloom filters to provide a suitable anonymity set to hide the addresses of recipients: here, the server cannot identify the receiver addresses in the anonymity set. However, subsequent research by Gervais et al. [GCKG14] showed that the Bloom filter-based solution leaks considerable information about recipient addresses. Later, Beck et al. [BLMG21] followed the idea of introducing false-positive probability, but they avoided the information leakage of addresses. While false-positive probability is an effective means to improve privacy, it is not conducive to achieving *full privacy* (i.e., hiding the pertinent messages among all messages) and inherently results in higher communication and computational costs as the privacy level is enhanced. This is because the recipient would receive more messages and identify the pertinent ones from them. Recently, Madathil et al. [MSS⁺22], Liu et al. [LT22] and Jakkamsetti et al. [JLM23] proposed solutions that do not rely on false-positive probability, thus avoiding the above shortcomings. Therefore, their solutions achieve full privacy and constitute the key related work for this paper. Nonetheless, their schemes still have significant drawbacks.

Limitations in the Existing Solutions. The schemes proposed by Madathil et al. [MSS⁺22] and Jakkamsetti et al. [JLM23] rely on a Trusted Execution Environment (TEE), which is a strong trust assumption. Meanwhile, the two-server scheme in [MSS⁺22] and the single-server scheme (that leverages fully homomorphic encryption) introduced by Liu et al. [LT22], both impose significant computational overhead on server(s). Moreover, both schemes in [MSS⁺22] require servers to *update the entire table* whose size is linear in the number of unretrieved messages (denoted as N), while adding a new message. Most recently, the scheme in [JLM23] reduces the addition cost to $O(\log N)$ by leveraging Oblivious RAM (ORAM). In addition, the schemes in [MSS⁺22, LT22] *limit* the number of pertinent messages for a recipient. This limitation not only compromises usage flexibility but also facilitates attacks by adversaries aiming to consume the “message balance” of a recipient.

In this work, we design a new protocol, HomeRun (High-efficiency Oblivious Message Retrieval Unrestricted), that *avoids the shortcomings of the previous work while enjoying their merits*. Let us start with a trivial approach: a server holding a list of message/address pairs $\{(m_i, \text{Addr}_i)\}$ tests whether $\text{Addr}_i = \text{Addr}'$ to determine if message m_i is pertinent to a recipient with address Addr' . Obviously, in this straightforward approach, the server can directly append a new message/address pair to the list, and the number of pertinent to a recipient is unlimited. However, recipients’ privacy is compromised. To achieve full privacy, our key idea is to split each Addr_i and Addr' between two servers such that each server only obtains a share of each address. Then, the two servers perform a matching process without knowing the actual address and the final output. Moreover, HomeRun takes into account the following two highly relevant enhancements.

Allowing Deletion. In the previous works [BLMG21, MSS⁺22, LT22, JLM23], for each message on the bulletin board, the server(s) stores messages and corresponding auxiliary information² to assist recipients with retrieving messages while guaranteeing their privacy. However, since the server(s) do not know which messages have been retrieved, these solutions do not directly support deletion. This would lead to a constantly increased storage cost. Moreover, apart from the scheme in [JLM23], the server runtime in all of the above existing schemes (including ours) is linearly dependent on the number of messages that are stored. Therefore, without deletion, the concrete efficiency of previous schemes may deteriorate very quickly. Whereas, our work for the first time introduces a deletion feature, which ensures precise and regular deletion of retrieved messages, thus avoiding the issue.

¹For example, the size of Bitcoin is 511.84 GB [bit], the size of Ethereum is 1221.43 GB [eth], until September 2023.

²The auxiliary information for the schemes in [MSS⁺22, JLM23] refer to a ciphertext. As for the schemes in [BLMG21, LT22], the auxiliary information is a label. The schemes in [MSS⁺22, JLM23] only provided pertinent indexes to recipients without considering retrieving messages. Therefore, the server(s) in their schemes does not store messages. For simplicity, we do not explicitly differentiate between these two kinds of schemes, and the schemes in [MSS⁺22, JLM23] can be seamlessly integrated with PIR to achieve message retrieval.

Table 1: Comparisons with the previous works.

	FMD1/2 [BLMG21]	PS1 [MSS+22]	PS2 [MSS+22]	OMRp1/2 [LT22]	S-PS [JLM23]	Ours
Setup	1 SH Svr	1 TEE Svr	2 SH Svrs	1 SH Svrs	1 TEE Svr	2 SH Svrs
Addition Method	Append	Update All	Update All	Append	ORAM	Append
Full Privacy	✗	✓	✓	✓	✓	✓
#Pert. Msgs Unlimit	✓	✗	✗	✗	✓	✓
Deletion Support	✗	✗	✗	✗	✗	✓
Request Unlinkability	✗	✓	✗	✓	✓	✓

¹ Setup: “SH Svr” means semi-honest server; “TEE Svr” means a server with a TEE.

² Addition Method: There are three methods to add a new message: (1) updating the entire list, (2) appending to the list and (3) storing the list in an ORAM and adding via ORAM.

³ Full Privacy: A pertinent message is hidden among all unretrieved messages on board.

⁴ #Pert. Msgs Unlimit: The number of pertinent messages to a recipient is unlimited.

⁵ Deletion Support: Retrieved messages can be periodically deleted.

⁶ Request Unlinkability: Server(s) cannot link two retrieval requests from the same address.

Achieving Retrieval Request Unlinkability. Here, we consider an underlying privacy leakage risk: if a recipient requests retrieval twice, the server(s) can link those two requests. We observed that the TEE-based schemes in [MSS+22]³ and [JLM23] does not suffer from this leakage risk, since the server can only access the ciphertexts of retrieval requests under the public key of TEE. However, other solutions [MSS+22, BLMG21, LT22] are all susceptible to this risk; requests from the same recipient will include the (secret) key of the recipient. Therefore, the server(s) can link these requests by identifying the same (secret) keys⁴. Notice that even if recipients communicate with servers over anonymous channels, the requests remain linkable.

Liu et al. [LT22] also observed this leakage risk, and they proposed to resample the key for each retrieval request. Nevertheless, this solution cannot be applied to [MSS+22, BLMG21]. Unlike the scheme in [LT22], the (secret) keys that incur linkability in [MSS+22, BLMG21] are related to addresses. Moreover, senders need to produce the above-discussed auxiliary information based on the recipient address. This means that the recipient needs to frequently update her addresses with all her prospective senders, which affects usability.

In our protocol, for every request, each server receives a random share of an address. As a result, even when a recipient consistently uses the same address, the servers are unable to establish a link between these requests assuming that the recipient communicates with servers over an anonymous channel to avoid network-level leakage. The privacy⁵ achieved by our protocol is captured by the functionality \mathcal{F}_{OMR} in Figure 6. Please see Section 3.2 for more details.

1.1 Our Contributions

In summary, we design an efficient protocol, called HomeRun, that can assist recipients with retrieving pertinent messages from a bulletin board while guaranteeing their privacy. See the summarized comparison with the previous works [BLMG21, MSS+22, LT22, JLM23] in Table 1.

First, HomeRun avoids the shortcomings existing in the previous works [BLMG21, MSS+22, LT22, JLM23], and has the following advantages:

³When a table is too large, it must be stored in the memory of the server. In this case, the server can also link retrieval requests based on access patterns. The problem is solved by [JLM23] using ORAM.

⁴In the two schemes proposed by [BLMG21], a recipient needs to send secret keys to the server. In the two-server solution in [MSS+22], a recipient is required to send her signature verification key to the servers. The retrieval request in the scheme designed by Liu et al. [LT22] includes a BFV public key.

⁵Similar to [MSS+22, BLMG21, LT22, JLM23], the discussion on privacy in this work is conducted without considering the network-level leakage, which is beyond the scope of our paper. In practice, for example, Tor [Tor] can be used to mitigate network-level leakage.




Bulletin Board	Server(s)	Recipient	
			
(1) Add a new message	Request index retrieval	(3) Recover the indexes	
→	←	→	
(2) Response			
PS1:	4.41	0.069	1.52
PS2:	7425.94	0.0011	0.00014
OMDp1:	~ 0	19494.57	0.0054
S-PS:	~ 0.003	~ 0.003n	0.00001n
Ours:	~ 0	5.09	0.0065

Figure 1: Comparison of runtime (in seconds) for addition and index retrieval phases, with the prior schemes PS1 / PS2 [MSS⁺22], OMDp1 [LT22] and S-PS [JLM23]. PS1 and S-PS are based on TEE. The number of pertinent messages is n . Here, for PS1 / PS2 and OMDp1, n is up to 50; for S-PS and ours, n is unlimited. The runtime of S-PS is linear in n , whereas ours is independent on n . The two servers in PS2 communicate in a LAN setting⁶, while those in HomeRun run in a WAN setting. All experiments are evaluated using a single thread.

- We achieve *full privacy* efficiently where a pertinent message is hidden among all messages on the bulletin board. Importantly, we achieve this without relying on trusted hardware or heavy machinery like FHE.
- We allow efficiently adding new messages through *appending*. This is advantageous in scenarios where new messages are frequently generated and a large number of unretrieved messages are on the bulletin board.
- We allow *unlimited* number of pertinent messages for a recipient. This enhances user-friendliness and effectively prevents adversaries from consuming the message balance of a recipient.

We additionally introduce two features into HomeRun. Specifically, *deletion* avoids the constantly increasing computation time and storage cost of servers. *Retrieval request unlinkability* guarantees that servers cannot link multiple retrieval requests derived from the same address.

Finally, we implement HomeRun and provide comprehensive performance comparisons with the prior works [BLMG21, MSS⁺22, LT22, JLM23]. The protocols in [MSS⁺22, JLM23] only focus on retrieving pertinent indexes without considering how to retrieve the pertinent messages, and the scheme in [BLMG21] cannot achieve full privacy. Therefore, in Figure 1, we only show performance comparisons with [MSS⁺22, LT22, JLM23] in addition and index retrieval phases, assuming that the bulletin board contains 2^{19} messages. Please see Section 5.2 for more details.

From Figure 1, we can see that the overall performance of HomeRun is comparable to that of PS1 [MSS⁺22] and significantly surpasses those of PS2 [MSS⁺22] and OMDp1 [LT22]. When the number of pertinent indexes, denoted as n , reaches the scale of several thousand, our performance is comparable to that of S-PS [JLM23]. Moreover, increasing to 16 threads can reduce our response time to 0.96s. In addition, the servers in HomeRun spend 4.47s to delete the retrieved messages in the WAN setting, regardless of the number of messages to be deleted.

Organization. We first introduce our core ideas in Section 2. This followed by preliminaries in Section 3. Our detailed protocol is presented in Section 4. Then, we evaluate our protocol and provide comprehensive comparisons with prior works through experiments in Section 5. In Section 6, we discuss potential extensions based on our protocol. Finally, we provide a recall of related work in Section 7 and a conclusion in Section 8.

⁶The source code of PS2 does not directly support network simulation.

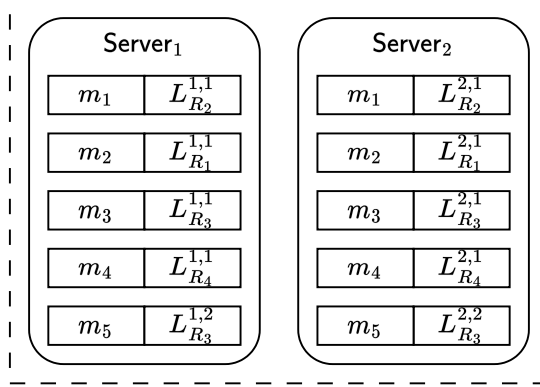


Figure 2: The lists maintained by the two servers. For message m_i pertinent to recipient R_k , Server_j stores it alongside its label $L_{R_k}^{j,t}$, where t denotes it is the t -th message for this recipient. Moreover, $L_{R_k}^{1,t} \oplus L_{R_k}^{2,t} = \text{Addr}_{R_i}$ where Addr_{R_i} is the address of the recipient.

2 Technical Overview

Our work aims to design an efficient and scalable OMR protocol, by leveraging two non-colluding servers, Server_1 and Server_2 . In an OMR protocol, a recipient (namely, R) will first share her address (namely, Addr_R) with prospective senders or all parties (perhaps by posting it to a public repository). Then, a sender submits a message and associated auxiliary data (that is generated based on the recipient’s address) to a bulletin board. Later, a recipient can retrieve her pertinent messages with the assistance of the two servers, without scanning the entire bulletin board. Moreover, the two servers cannot link the recipient’s address to her pertinent messages.

2.1 Core Idea

We start with a naive solution that does not preserve the privacy of recipients. We use a random string as the address (namely, Addr) of a recipient and let a server store a list of message/address pairs $\{(m_i, \text{Addr}_i)\}$. If each message is sent to a different recipient, no additional information is leaked from the list. However, a recipient always receives more than one message. In this case, the server maintaining the pairs of messages and addresses can learn which messages are pertinent to the same recipient. Fortunately, we have two semi-honest, non-colluding servers, denoted as Server_1 and Server_2 . Therefore, we can require each server to store a random share of each address, instead of the actual address. In this way, even if two messages are associated with the same address Addr_R , we can randomly share Addr_R in two different ways, e.g., $\text{Addr}_R = L_R^{1,1} \oplus L_R^{2,1}$ and $\text{Addr}_R = L_R^{1,2} \oplus L_R^{2,2}$. Then, Server_1 holds $L_R^{1,1}$ and $L_R^{1,2}$, while Server_2 holds $L_R^{2,1}$ and $L_R^{2,2}$. Since $L_R^{1,1} \neq L_R^{1,2}$ and $L_R^{2,1} \neq L_R^{2,2}$, neither of the two servers can know that the two messages are sent to the same recipient. We call the share of an address “label”, and show the lists maintained by the two servers in Figure 2. Next, we explain the processes of sending and retrieval.

We suppose that two servers have previously received messages m_1, \dots, m_4 with their associated labels. Now, a sender wants to transmit a message m_5 to recipient R_3 . To do this, the sender generates the shares $L_{R_3}^{1,2}$ and $L_{R_3}^{2,2}$ and sends $(m_5, L_{R_3}^{1,2})$ and $(m_5, L_{R_3}^{2,2})$ to the two servers, respectively. When the recipient R_3 wants to retrieve messages, R_3 generates two shares of her address Addr_{R_3} , denoted as $L_{R_3}^1$ and $L_{R_3}^2$, and sends them to the two servers, respectively. Note that, given that the sender and recipient generate their respective shares randomly, $L_{R_3}^{1,2} \neq L_{R_3}^1$ and $L_{R_3}^{2,2} \neq L_{R_3}^2$ hold true with overwhelming probability. Similarly, for the message m_3 that has been stored by the two servers, $L_{R_3}^{1,1} \neq L_{R_3}^1$ and $L_{R_3}^{2,1} \neq L_{R_3}^2$ also hold with overwhelming probability. Therefore, given $L_{R_3}^1$ (resp. $L_{R_3}^2$), Server_1 (resp. Server_2) cannot know messages m_3 and m_5 are pertinent to recipient R_3 . Next, we explain how the servers help the recipient R_3 retrieve

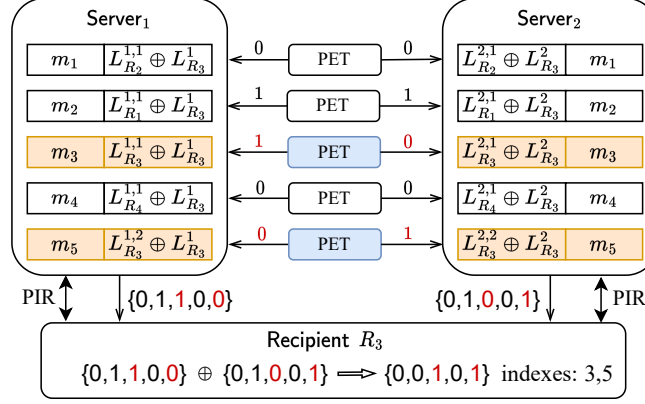


Figure 3: Our initial protocol. Here, we take recipient R_3 as an example. The recipient R_3 generates $L_{R_3}^1$ and $L_{R_3}^2$ such that $\text{Addr}_{R_3} = L_{R_3}^1 \oplus L_{R_3}^2$. After receiving $L_{R_3}^j$ from R_3 , Server_j XORs it with each stored label to obtain “test label”. For each pair of test labels, servers perform a “Private Equality Test (PET)”; equal test labels yield opposite bits, while differing labels produce identical bits. Then, servers send PET outputs (i.e., two bit-vectors) to R_3 . Later, R_3 learns 3 and 5 are pertinent indexes through XOR operation, and retrieves m_3 and m_5 through “Private Information Retrieval (PIR)”.

pertinent messages m_3 and m_5 , and illustrate the process in Figure 3.

The fact that messages m_3 and m_5 are pertinent to recipient R_3 means that $L_{R_3}^{1,1} \oplus L_{R_3}^1 = L_{R_3}^{2,1} \oplus L_{R_3}^2$ and $L_{R_3}^{1,2} \oplus L_{R_3}^1 = L_{R_3}^{2,2} \oplus L_{R_3}^2$. Therefore, Server_1 (resp. Server_2) can perform XOR to $L_{R_3}^1$ (resp. $L_{R_3}^2$) with the label of each message m_i to obtain the corresponding “test label” $L_T^{1,i}$ (resp. $L_T^{2,i}$), and then performs an equality test on each pair $(L_T^{1,i}, L_T^{2,i})$ to check if m_i is pertinent. However, the result of each equality test will leak information about pertinent messages, so we use “Private Equality Test (PET)” [DSZ15, Cou18, CGS22]. Through PET, for each pair, the two servers only obtain random bits b_i^1 and b_i^2 , respectively; if $L_T^{1,i} = L_T^{2,i}$, $b_i^1 \oplus b_i^2 = 1$, otherwise $b_i^1 \oplus b_i^2 = 0$. After obtaining b_i^1 and b_i^2 , the recipient R_3 can learn if m_i is pertinent by computing $b_i^1 \oplus b_i^2$, then obtain m_i using “Private Information Retrieval (PIR)” [BG16, KOR19, HHCG+23].

Nonetheless, the aforementioned initial protocol presents a vulnerability: anyone possessing the address of the recipient R_3 can retrieve messages m_3 and m_5 , as the retrieval process does not require any secret information. To address this, we subsequently introduce a unique secret key for each recipient. This allows the servers to authenticate recipients while preserving their anonymity.

2.2 Achieving Authentication of Recipients

In the previous works [MSS+22, LT22, BLMG21, JLM23], each recipient uses a secret key(s) to generate a retrieval request. Therefore, other users not holding the secret key cannot retrieve the corresponding messages successfully. To achieve the same security property, we replace the random string in the above initial protocol with a public key chosen from a group \mathbb{G} . When initiating a retrieval request, a recipient now needs to prove the knowledge of the corresponding secret key. However, following the above initial protocol where the address is shared between the two servers through the XOR operation, will require the two servers to interact while verifying the proof. To avoid the interaction, we further replace the XOR operation with the operation of the group \mathbb{G} , denoted as “.” here. Accordingly, the servers use the “.” operation and corresponding inverses of the group elements to calculate test labels. Next, we also use the above example to explain the authentication process.

The recipient R_3 generates $L_{R_3}^1 = g^{sk_{R_3,1}}$ and $L_{R_3}^2 = g^{sk_{R_3,2}}$ such that $L_{R_3}^1 \cdot L_{R_3}^2 = \text{Addr}_{R_3} = g^{sk_{R_3}}$ where g is the generator of group \mathbb{G} . Besides $L_{R_3}^1$ and $L_{R_3}^2$, the recipient R_3 additionally sends proofs π_1 and π_2 proving the knowledge of $sk_{R_3,1}$ and $sk_{R_3,2}$, to Server_1 and Server_2 , respectively. Note that only if the recipient knows sk_{R_3} can they compute the proofs π_1 and π_2 correctly. Then, the two servers calculate

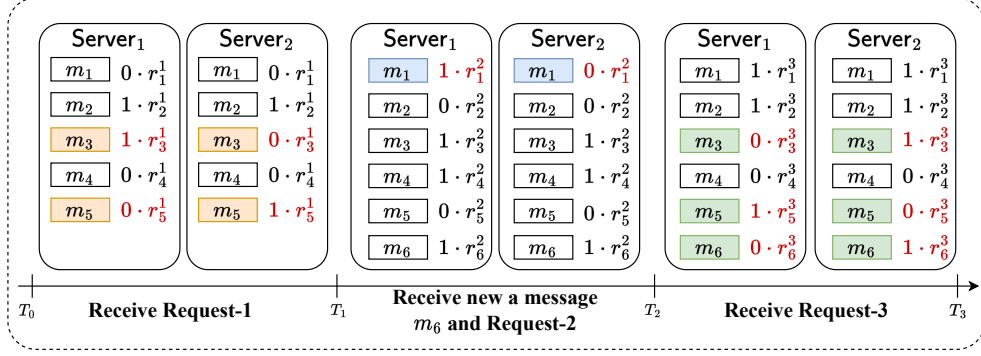


Figure 4: New states triggered by retrieval requests and new messages. (1) $T_0 \sim T_1$: Assuming that m_3 and m_5 are pertinent for Request-1, the 3-rd and 5-th delete label pairs (highlighted in red) are different. This observation is also consistent for Request-2 and Request-3. (2) $T_1 \sim T_2$: If there is a new message m_6 , both servers simply append it to their lists and produce the corresponding delete labels. (3) $T_2 \sim T_3$: Assuming that m_3 , m_5 and m_6 are pertinent to Request-3, the two servers generate delete labels for all three, even though m_3 and m_5 were previously retrieved by Request-1. Note that m_3 and m_5 having been retrieved is not known to the servers.

a pair of test labels for each message using “ \cdot ” operation. For example, the test labels for message m_1 are $L_{R_2}^{1,1} \cdot (L_{R_3}^1)^{-1}$ and $(L_{R_2}^{2,1})^{-1} \cdot L_{R_3}^2$. Likewise, the servers then perform PET on these test label pairs, such that the recipient R_3 can only obtain the messages whose two test labels are equal.

It is worth mentioning that our authentication process includes two steps: (1) verifying the proofs and (2) privately testing equality. The first step guarantees that the recipient knows a secret key sk for an address g^{sk} , and the second step ensures that only the messages whose addresses are equal to g^{sk} will be retrieved by the recipient, thus ensuring that an adversary cannot impersonate an honest recipient.

2.3 Achieving Deletion

Existing schemes [MSS⁺22, LT22, BLMG21] do not directly support deletion. In the approach by [MSS⁺22], the server must keep a vector of length ℓ (e.g., $\ell = 50$) for every registered recipient. Similarly, in the schemes [LT22, BLMG21], the list maintained by the server continually expands unless the server periodically deletes messages that have been stored for more than a specified duration (e.g., one day), regardless of whether they have been retrieved or not. However, in practice, it is desirable to give recipients the flexibility to retrieve messages at any time, while periodically deleting the messages that have indeed been retrieved. To this end, we design a deletion mechanism based on the aforementioned protocol, detailed as follows.

In this work, we assume that after obtaining pertinent indexes from a pair of servers, a recipient will retrieve their pertinent messages from the same pair of servers, within a fixed time Δ . Therefore, the servers can delete messages and labels according to the indexes that are recognized as pertinent. In Section 6.3, we will further discuss how to perform deletion when this assumption does not hold.

In the above protocol, for each message m_i , Server₁ generates a bit b_i^1 while Server₂ generates a bit b_i^2 ; if the message m_i is pertinent, $b_i^1 \oplus b_i^2 = 1$, otherwise, $b_i^1 \oplus b_i^2 = 0$. A natural solution is to delete all the messages corresponding to $b_i^1 \oplus b_i^2 = 1$. We perform a deletion once every specified interval (such as daily).

More specifically, for message m_i , we denote the bit generated by Server₁ (resp. Server₂) for the j -th request in a particular interval as $b_i^{1,j}$ (resp. $b_i^{2,j}$). If each message is only retrieved once, the two servers can interactively compute $b_i = (b_i^{1,1} \oplus b_i^{2,1}) \oplus \dots \oplus (b_i^{1,k} \oplus b_i^{2,k})$ assuming there are k requests in this interval. Then, the two servers can delete the messages whose $b_i = 1$. However, if a recipient requests twice in an interval, some of her pertinent messages will be retrieved twice, which means that there are j_1 and j_2 such that $(b_i^{1,j_1} \oplus b_i^{2,j_1}) = (b_i^{1,j_2} \oplus b_i^{2,j_2}) = 1$ leading to $b_i = (b_i^{1,1} \oplus b_i^{2,1}) \oplus \dots \oplus (b_i^{1,k} \oplus b_i^{2,k}) = 0$, so the corresponding message m_i will not be deleted successfully. To avoid an even number of 1s being canceled out by the XOR

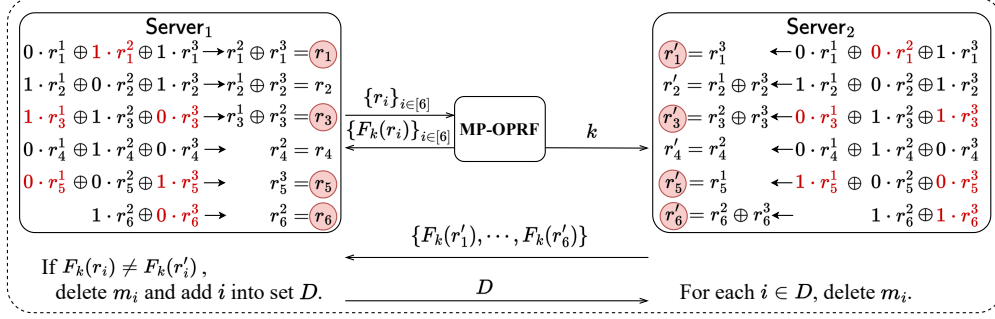


Figure 5: Our core idea for deletion. For each message m_i , the two servers XOR all its delete labels to obtain r_i and r'_i , respectively. In the example shown in Figure 4, messages m_1 , m_3 , m_5 and m_6 are retrieved, so their corresponding r_i and r'_i (marked in red circles) differ. Through “Multi-point OPRF (MP-OPRF)”, Server₁ can learn the inequalities without knowing each r'_i , and then delete the corresponding messages, whose indexes are recorded in set D . Once obtaining D , Server₂ executes the deletion.

operation, we introduce a distinct randomness for each request. To illustrate our deletion solution, we show how to introduce randomnesses in Figure 4 and how to delete retrieved messages in Figure 5, and more details are explained below.

To be precise, for the j -th request, the two servers agree on a randomness⁷ r_i^j and compute “delete labels” $b_i^{1,j} \cdot r_i^j$ and $b_i^{2,j} \cdot r_i^j$ ⁸ for each message m_i , respectively. In Figure 4, we assume that deletion is not executed before time T_3 .

After time T_3 , the two servers need to delete the retrieved messages, i.e., m_1 , m_3 , m_5 and m_6 . In Figure 5, we show how to use the above delete labels to perform the deletion without leaking additional information. More specifically, for each message m_i , Server₁ (resp. Server₂) first generates r_i (resp. r'_i) by performing XOR operations on all delete labels of m_i . Obviously, if m_i has not been retrieved during the period $T_0 \sim T_3$, the corresponding r_i would be equal to r'_i , otherwise $r_i \neq r'_i$. Note that we cannot require Server₂ to send each r'_i to Server₁ directly, since Server₁ can then learn the recipients of retrieved messages. Take the message m_3 as an example: Server₁ knows that $b_3^{1,1} = 1$, $b_3^{1,2} = 1$ and $b_3^{1,3} = 0$. Upon receiving r'_3 , Server₁ can learn that $r'_3 = r_3^2 \oplus r_3^3$ since Server₁ also knows r_3^1 , r_3^2 and r_3^3 . Furthermore, Server₁ can deduce that $b_3^{2,1} = 0$, $b_3^{2,2} = 1$ and $b_3^{2,3} = 1$. Accordingly the Server₁ can compute $b_3^{1,1} \oplus b_3^{2,1} = 1$, $b_3^{1,2} \oplus b_3^{2,2} = 0$ and $b_3^{1,3} \oplus b_3^{2,3} = 1$, and it learns that m_3 is pertinent to the recipient of Request-1 and Request-3.

In order to enhance the privacy of recipients, we leverage “Multi-point Oblivious PRF (MP-OPRF)” to hide each r'_i . Through MP-OPRF, Server₂ receives a key k and Server₁ obtains the PRF value $F_k(r_i)$ for each r_i without knowing k . Then, after obtaining each $F_k(r_i)$, Server₁ deletes message m_i and records index i into a set D if $F_k(r_i) \neq F_k(r'_i)$. Server₂ then deletes messages according to the set D .

In a nutshell, we periodically delete the messages that are retrieved in each interval. For a specific interval, the retrieved messages can be regarded as an anonymity set to break the linkability between messages and requests. However, we admit that if the anonymity set is too small (e.g. 1), the servers will learn some information about the linkability. This leakage is captured in our OMR functionality \mathcal{F}_{OMR} shown in Figure 6. Therefore, in practice, the duration of the period should be configured to an adequate length so that the anonymity set is large enough. This can be enforced since we assume that the two servers do not collude, and at least one server will honestly wait until enough time has elapsed. Nevertheless, we further discuss how to allow the recipients to determine whether to delete in Section 6.2.

⁷It can be derived from a common seed.

⁸When b is a bit and r is a string, if $b = 0$, $b \cdot r = 0$, otherwise $b \cdot r = r$.

Functionality \mathcal{F}_{OMR}

The functionality maintains a table \mathcal{T} indexed by each recipient R_j , which contains the messages pertinent to the corresponding recipient. The functionality also maintains a set D that keeps track of the messages that are retrieved.

Sending a message (SEND):

1. Upon receiving $\langle \text{SEND}, R_j, msg \rangle$ from a sender S_i , send $\langle \text{SEND}, msg \rangle$ to the adversary and server(s);
2. Upon receiving $\langle \text{SEND}, \text{OK} \rangle$ from the adversary and server(s), append msg to $\mathcal{T}[R_j]$;

Retrieving messages (RECEIVE):

1. Upon receiving $\langle \text{RECEIVE} \rangle$ from a recipient R_j , send $\langle \text{RECEIVE} \rangle$ to the adversary and server(s);
2. Upon receiving $\langle \text{RECEIVE}, \text{OK} \rangle$ from the adversary and server(s), send $\langle \text{RECEIVE}, \mathcal{T}[R_j] \rangle$ to the recipient R_j ;
3. Add $\mathcal{T}[R_j]$ to D and update $\mathcal{T}[R_j] = []$;

Deletion (DELETE):

1. Upon receiving $\langle \text{DELETE} \rangle$ from the environment \mathcal{Z} , send $\langle \text{DELETE}, D \rangle$ to the server(s) and $\langle \text{DELETE} \rangle$ to the adversary;
2. Upon receiving $\langle \text{DELETE}, \text{OK} \rangle$ from the adversary and server(s), update $D = \emptyset$.

Figure 6: Server-aided Oblivious Message Retrieval.

3 Preliminaries

Notations. Let κ and λ be the computational and statistical security parameters, respectively. We use $[m]$ to denote the set $\{1, 2, \dots, m\}$. A vector containing $|\vec{X}|$ items is denoted by \vec{X} , and $\vec{X}[i]$ is the i -th item. A table \mathcal{T} indexed by R_j indicates that each row corresponds uniquely to an R_j and $\mathcal{T}[R_j]$ is the row indexed by R_j .

3.1 Threat Model

We assume that all parties (including two servers, multiple senders/recipients, and adversaries) can read from and submit messages to the bulletin board (abstracted as global functionality $\mathcal{G}_{\text{Ledger}}$ shown in Figure 10). In addition, we assume that the adversaries run in polynomial time in the computational security parameter κ . We assume the two servers are semi-honest and non-colluding, which means that they will follow the instructions of our protocol, but try to learn more information from the view of protocol execution without collusion. Senders and recipients may behave maliciously and collude with one of the servers.

3.2 Oblivious Message Retrieval Functionality

We define our OMR functionality \mathcal{F}_{OMR} in Figure 6, based on the definition given in [MSS⁺22]. Recall that our goal is to assist recipients in obliviously retrieving the pertinent messages from a bulletin board, if the recipients can only obtain pertinent indexes, they still need to retrieve the corresponding messages from the bulletin board by themselves. Moreover, recipients may not be able to download a large bulletin board. Therefore, the table \mathcal{T} in our \mathcal{F}_{OMR} contains directly the messages for recipients, rather than indexes. It is worth mentioning that OMR is different from “Private Information Retrieval (PIR)” (please see Definition 3.1); in PIR, a recipient leverages indexes to retrieve messages, whereas, in OMR, a recipient does not even know the indexes and just uses their addresses (i.e., R_j) to request retrievals. In addition, our \mathcal{F}_{OMR} supports deletion, which was not considered in [MSS⁺22]. Note that we focus on server-aided OMR and thus we explicitly show the role of server in \mathcal{F}_{OMR} .

Privacy Implied by \mathcal{F}_{OMR} . For a sending request, the adversary can obtain $\langle \text{SEND}, msg \rangle$, which means that the adversary can only obtain the message but cannot learn anything about the sender and recipient. For a retrieval request, the adversary learns nothing about the recipient and the retrieved messages from $\langle \text{RECEIVE} \rangle$. For the deletion procedure, the server(s) can learn the set D contains all messages retrieved since the last deletion was executed, but the adversary cannot learn the set D .

3.3 Building Blocks

We collect the building blocks here.

3.3.1 Private Equality Test (PET)

Through PET, two parties with items x and y can obtain bits b_1 and b_2 , respectively, such that if $x = y$, $b_1 \oplus b_2 = 1$, otherwise, $b_1 \oplus b_2 = 0$. We show the formal definition in Figure 7. The performance of PET directly affects the latency of our protocol, so we implemented it using the scheme in [CGS22], which enjoys efficient online performance.

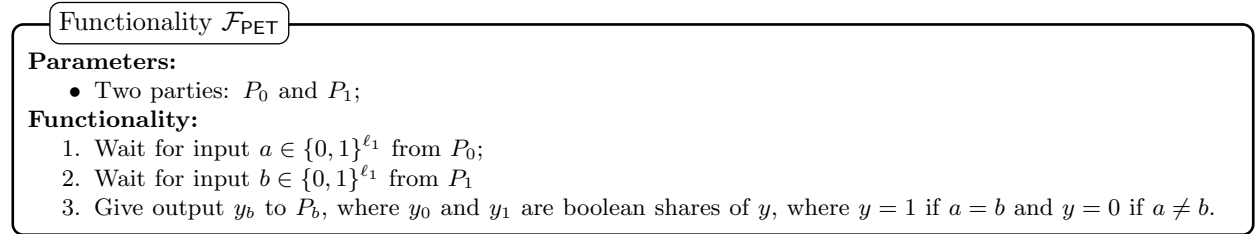


Figure 7: Private Equality Testing Ideal Functionality

3.3.2 Private Information Retrieval (PIR)

PIR allows a client to select an index and retrieve the corresponding message from a server while ensuring the server remains unaware of which message is retrieved. The formal definition is shown in Definition 3.1. Given the persistent arrival of new messages and two-party setting in our application, we implemented PIR leveraging the DPF-based PIR [KOR19, BGI16], which can achieve an efficient performance without needing a setup phase on a fixed database.

Definition 3.1 (Two-Server PIR). A 2-server PIR protocol involving two server S_0 and S_1 holding the same n -bit database z and user consists of three algorithms Q, A, M with query domain D_Q and answer domain D_A and are defined as follows:

- $Q(n, i)$: On input of an index i , client return queries $(q_0, q_1) \in D_Q^2$
- $A(z, q)$: On input of a query q and a database z , server b returns an answer a_b
- $M(i, a_0, a_1)$: On input of an index i and two answers a_0, a_1 recovers and returns the i -th database entry z_i .

Definition 3.2 (Secure Two-Server PIR). The security properties guaranteed by a PIR protocol are correctness and computational secrecy.

Correctness: A 2-server PIR is correct if for every $n \in \mathbb{N}$, every $z \in \{0, 1\}^n$, and every $i \in [n]$ it holds that

$$\Pr[(q_0, q_1) \leftarrow Q(n, i) : M(i, A(j, z, q_j))_{j \in \{0, 1\}} = z_i] = 1$$

Computational Secrecy: Let $D_{b, \lceil \log n \rceil, i}, b \in \{0, 1\}, n \in \mathbb{N}$ and $i \in [n]$ denote the probability distribution on q_b induced by Q . A 2-server PIR scheme provides computational secrecy if there exists a PPT algorithm S_{PIR} such that:

$$\{S_{\text{PIR}}(b, \lceil \log n \rceil)\}_{b \in \{0, 1\}, n \in \mathbb{N}} \approx_c \{D_{b, \lceil \log n \rceil, i}\}_{b \in \{0, 1\}, n \in \mathbb{N}, i \in [n]}.$$

3.3.3 Multi-point Oblivious PRF (MP-OPRF)

A party P_0 with an item x performs OPRF with another party P_1 , such that P_1 obtains a key k without learning x , and P_0 obtains PRF value $F_k(x)$ without knowing k . The term ‘‘Multi-point’’ means that P_0 can input multiple items in an instance. We provide the formal definition in Figure 8. We utilized the scheme in [RS21] to implement MP-OPRF, given its high efficiency in both LAN and WAN settings.

Functionality $\mathcal{F}_{\text{MOPRF}}$

Parameters:

- Two parties: P_0 and P_1 ;
- A PRF $F_k(\cdot) : \{0, 1\}^{\ell_2} \rightarrow \{0, 1\}^{\ell_3}$.

Functionality:

1. Wait for input $\vec{X} = \{x_1, \dots, x_n\}$ from P_0 ;
2. Randomly select a key k for $F_k(\cdot)$;
3. Give output $\{F_k(x_1), F_k(x_2), \dots, F_k(x_n)\}$ to P_0 , and the key k to P_1 .

Figure 8: Multi-Point OPRF functionality.

3.3.4 Non-interactive Zero-Knowledge Proof (NIZK)

Given an NP language \mathcal{L} and its corresponding efficiently decidable binary relation \mathcal{R} , we say a statement $x \in \mathcal{L}$ if there exists a witness w such that $(x, w) \in \mathcal{R}$. In NIZK, a prover can generate a proof π to prove the knowledge of w satisfying $(x, w) \in \mathcal{R}$. Through verifying π , a verifier can learn if the prover possesses a valid w without knowing the information about w . The formal definition of NIZK is given in Figure 9. In this work, we leveraged the Σ -protocol [Sch90] for discrete logarithm language to enable recipients to prove the knowledge of secret keys.

Functionality $\mathcal{F}_{\text{NIZK}}$

Parameters: The non-interactive zero-knowledge functionality $\mathcal{F}_{\text{NIZK}}^{\mathcal{L}}$ allows proving of statements in an NP language \mathcal{L} . It maintains a set of statement/proof pairs Q , initialized to \emptyset . Let \mathcal{R} be an efficiently decidable binary relation for the NP language \mathcal{L} .

Functionality:

Prove: Upon receiving $\langle \text{PROVE}, \text{sid}, x, w \rangle$:

1. If $(x, w) \notin \mathcal{R}$ then return $\langle \text{PROOF}, \text{sid}, x, \perp \rangle$;
2. Else send $\langle \text{PROVE}, \text{sid}, x \rangle$ to \mathcal{A} and receive the reply $\langle \text{PROOF}, \text{sid}, x, \pi \rangle$. Do $Q = Q \cup \{(x, \pi)\}$ and return $\langle \text{PROOF}, \text{sid}, x, \pi \rangle$;

Verify: Upon receiving $\langle \text{VERIFY}, \text{sid}, x, \pi \rangle$:

1. If $(x, \pi) \notin Q$ then send $\langle \text{VERIFY}, \text{sid}, x, \pi \rangle$ to \mathcal{A} and then receive the reply Res ;
2. If $\text{Res} = \langle \text{WITNESS}, \text{sid}, x, \pi, w \rangle \wedge (x, w) \in \mathcal{R}$ then let $Q = Q \cup (x, \pi)$;
3. Return $\langle \text{VERIFY}, \text{sid}, x, \pi, (x, \pi) \in Q \rangle$.

Figure 9: $\mathcal{F}_{\text{NIZK}}$ functionality

3.3.5 Ledger

A bulletin board can be abstracted as a global ledger, which can be achieved by blockchain techniques. Any party can submit messages to the ledger and read the state of the ledger. The underlying consensus protocols can guarantee that all the parties can read the same state. We follow the global ledger definition in [KZZ16] and give an abridged version in Figure 10.

4 Protocol Description

We break our protocol into four parts: (1) Initialization phase $\Pi_{\text{Initialize}}$ (Figure 11): how the servers and clients initialize; (2) Message sending phase Π_{Send} (Figure 12): how a sender sends a message and servers store it such that it can be obliviously retrieved later; (3) Message retrieval phase Π_{Retrieve} (Figure 13): how

Abridged $\mathcal{G}_{\text{Ledger}}$ functionality

This functionality is globally available to all participants. **Parameters:**

- A predicate **Validate**: verify if a transaction is valid;
- A function **ExtendState**: update **state** according to the latest **buffer** and reset **buffer** = ϵ periodically;
- Variables **state** and **buffer**, both are initialized as ϵ ;

Functionality:

Submit: Upon receiving $\langle \text{SUBMIT}, \text{tx} \rangle$ from a party P_i :

1. Choose a unique transaction ID **txid** and set $\text{BTX} = (\text{tx}, \text{txid}, P_i)$;
2. If $\text{Validate}(\text{BTX}, \text{state}, \text{buffer}) = 1$ then $\text{buffer} = \text{buffer} \cup \text{BTX}$, and execute $\text{ExtendState}(\text{state}, \text{buffer})$;
3. Send $\langle \text{SUBMIT}, \text{BTX} \rangle$ to \mathcal{A} ;

Read: Upon receiving **READ** from a party P_i , send **state** to P_i . If received from \mathcal{A} , send $\langle \text{state}, \text{buffer} \rangle$ to \mathcal{A} .

Figure 10: Abridged $\mathcal{G}_{\text{Ledger}}$ functionality.

Protocol $\Pi_{\text{Initialize}}$

There are two servers, Server_1 and Server_2 , and multiple clients acting as senders or recipients. The bulletin board is abstracted by $\mathcal{G}_{\text{Ledger}}$ (see Figure 10).

Initialization:

Each Server_j ($j \in \{1, 2\}$) does the following:

- Generate key pair $(pk_j, sk_j) \leftarrow \text{PKE.Gen}(1^\lambda)$;
- Send $\langle \text{SUBMIT}, pk_j \rangle$ to $\mathcal{G}_{\text{Ledger}}$;
- Initialize two empty vectors $\vec{M}_j = \emptyset$ and $\vec{X}_j = \emptyset$ (for retrieval), a zero vector $\vec{Y}_j = \vec{0}$ (for deletion);

Each client acting as a recipient R_i does the following:

- Randomly choose $k_{R_i} \leftarrow \mathbb{Z}_p$, and generate an address $\text{Addr}_{R_i} = g^{k_{R_i}} \in \mathbb{G}$; (\mathbb{G} is a cyclic group with prime order p and generator g .)
- Send Addr_{R_i} to parties from which R_i is willing to receive messages;

Figure 11: Protocol for initialization.

a recipient obliviously retrieve messages; (4) Message deletion phase Π_{Delete} (Figure 14): how the servers delete the messages that have been retrieved, as well as the corresponding labels.

4.1 Initialization

In our protocol, there are two servers and multiple clients that can act as senders or recipients. In the initialization phase as shown in Figure 11, servers and clients are required to generate key pairs. Note that new clients can generate key pairs to join the system at any time.

Specifically, each server produces an encryption key pair and then posts the public key on a bulletin board. When a client wants to communicate with a server, the client can encrypt the message using the corresponding public key and publish the ciphertext on the bulletin board. Additionally, if a client will act as a recipient denoted R_i , the client needs to generate an address $\text{Addr}_{R_i} = g^{k_{R_i}}$ where k_{R_i} is the corresponding secret key used for retrieval. To receive messages, R_i must share the address Addr_{R_i} with the prospective sender.

In addition, each server also initializes three vectors. The first one denoted \vec{M} , is to store messages, while the second vector \vec{X} is used to store labels. The two vectors will be updated during the sending and deletion phases, as shown in figures 12 and 14 respectively. The third vector \vec{Y} serves for detecting which messages in \vec{M} and labels in \vec{X} should be deleted. This vector is updated in both the retrieval and deletion phases, as depicted in figures 13 and 14 respectively.

Publishing messages on the bulletin board:

To send a message m to a recipient R_i , the sender does the following:

- Obtain the address Addr_{R_i} of the recipient R_i (from the recipient or the bulletin board);
- Randomly split Addr_{R_i} into $L_{R_i,1}$ and $L_{R_i,2}$, such that $\text{Addr}_{R_i} = L_{R_i,1} \cdot L_{R_i,2}$, and compute $L_{R_i,2}^{-1}$;
- Encrypt $L_{R_i,1}$ and $L_{R_i,2}^{-1}$ as $c_1 = \text{PKE.Enc}_{pk_1}(L_{R_i,1})$ and $c_2 = \text{PKE.Enc}_{pk_2}(L_{R_i,2}^{-1})$ respectively;
- Send $\langle \text{SUBMIT}, (m, c_1, c_2) \rangle$ to $\mathcal{G}_{\text{Ledger}}$.

Preparing for retrievals:

For each new tuple $(m_k, c_{k,1}, c_{k,2})$ on $\mathcal{G}_{\text{Ledger}}$, for $j \in \{1, 2\}$, each Server_j does the following:

- Decrypt $c_{k,j}$ as $t_{k,j}$;
- If $t_{k,j} \in \mathbb{G}$, select an index i based on a predefined rule (e.g., according to the position on bulletin board) shared with Server_{1-j} , otherwise ignore the message;
- Set $\vec{M}_j[i] = m_k$ and $\vec{X}_j[i] = t_{k,j}$.

Figure 12: Protocol for sending.

4.2 Sending Messages

In this section, we describe the procedures undertaken by a sender and two servers when a message m is to be sent to a recipient possessing the address Addr_{R_i} . As shown in Figure 12, the sender publishes the message and auxiliary information on the bulletin board. Subsequently, the two servers read the message and auxiliary information from the bulletin board and prepare for future retrieval.

More specifically, the sender first needs to obtain the address Addr_{R_i} of the recipient R_i . In practice, a recipient may share her address by posting it on the bulletin board for prospective senders. Once obtaining the address Addr_{R_i} , the sender proceeds to randomly split Addr_{R_i} into two shares $L_{R_i,1}$ and $L_{R_i,2}$ such that $\text{Addr}_{R_i} = L_{R_i,1} \cdot L_{R_i,2}$. As mentioned in Section 2.2, Server_2 uses the inversion of a label from the sender to generate the corresponding test label. To reduce the computation cost of Server_2 , we require the sender to generate the inversion of $L_{R_i,2}$ directly. Then, the sender encrypts $L_{R_i,1}$ and $L_{R_i,2}^{-1}$ as ciphertexts c_1 and c_2 using the public keys of Server_1 and Server_2 , respectively. Lastly, the sender submits (m, c_1, c_2) on the bulletin board.

The two servers continuously monitor the bulletin board. Once a new valid message is detected, the servers will add it to their respective lists for subsequent retrieval. In particular, upon observing a new tuple $(m_k, c_{k,1}, c_{k,2})$ on the bulletin board, the two servers decrypt $c_{k,1}$ and $c_{k,2}$ to obtain $t_{k,1}$ and $t_{k,2}$, respectively. Then, if $t_{k,1}$ and $t_{k,2}$ both belong to group \mathbb{G} , the two servers collaboratively determine an index i , and assign $(\vec{M}_1[i], \vec{X}_1[i]) = (m_k, t_{k,1})$ and $(\vec{M}_2[i], \vec{X}_2[i]) = (m_k, t_{k,2})$, respectively.

4.3 Retrieval

We now describe how a recipient, who holds the address Addr_{R_i} and the corresponding secret key k_{R_i} , retrieves the pertinent messages. The recipient begins by requesting the pertinent indexes. Upon obtaining these indexes, the recipient then employs PIR to retrieve the pertinent messages by using these indexes. See Figure 13 for the procedure pseudocode.

For requesting indexes, the recipient first randomly chooses a serial number sn for this request. This serial number serves a dual purpose: it identifies the request and also contributes to generating the randomness utilized in deletion. Then, the recipient splits the secret key k_{R_i} into $k_{R_i,1}$ and $k_{R_i,2}$, and generates the corresponding address shares $L_{R_i,1} = g^{k_{R_i,1}}$ and $L_{R_i,2} = g^{k_{R_i,2}}$. To reduce the computation cost of Server_1 , the recipient also generates the inversion of $L_{R_i,1}$, denoted as $L_{R_i,1}^{-1}$. For authentication purpose, the recipient constructs proof π_1 and π_2 , proving knowledge of $k_{R_i,1}$ and $k_{R_i,2}$. Note that the recipient does not need to prove $k_{R_i,1} + k_{R_i,2} = k_{R_i}$. If this equation does not hold true, it implies that $L_{R_i,1} \cdot L_{R_i,2} \neq \text{Addr}_{R_i}$, leading to an unsuccessful retrieval of messages pertinent to address Addr_{R_i} . Moreover, an adversary cannot

retrieve an honest recipient's message since it cannot compute k_{R_i} from Addr_{R_i} . Finally, the recipient sends $\langle \text{sn}, L_{R_i,1}^{-1}, \pi_1, L_{R_i,1} \rangle$ to Server_1 , and $\langle \text{sn}, L_{R_i,2}, \pi_2 \rangle$ to Server_2 .

Upon receiving the request from a recipient, the two servers first check if the request is valid. More specifically, we represent the message received by Server_1 as $\langle \text{sn}, t_1, \pi_1, t'_1 \rangle$, and the message received by Server_2 as $\langle \text{sn}, t_2, \pi_2 \rangle$. Server_1 needs to check if $t_1, t'_1 \in \mathbb{G}$, $t_1 \cdot t'_1 = 1$ and π_1 is valid, while Server_2 needs to check if $t_2 \in \mathbb{G}$ and π_2 is valid. If both servers verify the request successfully, they will exchange $\langle \text{sn}, \text{OK} \rangle$ with each other. After verifying the request successfully, the two servers proceed to generate the bit vectors through PET. Server_j retrieves each $t_{k,j}$ from \vec{X}_j and computes $a_{k,j} = H_1(t_{k,j} \cdot t_j)$ where H_1 is a hash function mapping a group element to a string. Through PET, the two servers can obtain $b_{k,1}$ and $b_{k,2}$, respectively, such that if $a_{k,1} = a_{k,2}$, $b_{k,1} \oplus b_{k,2} = 1$, otherwise $b_{k,1} \oplus b_{k,2} = 0$. Then, Server_j places the bits obtained from PET sequentially into a vector \vec{B}_j and sends this vector to the recipient. In addition, the servers also need to prepare for subsequent deletions. For each message m_k , the two servers generate the same randomness $H_2(\text{sn}||k)$ by using the serial number sn , and update $\vec{Y}_1[k] = \vec{Y}_1[k] \oplus b_{k,1} \cdot H_2(\text{sn}||k)$ and $\vec{Y}_2[k] = \vec{Y}_2[k] \oplus b_{k,2} \cdot H_2(\text{sn}||k)$, respectively.

After receiving \vec{B}_1 and \vec{B}_2 , the recipient calculates $\vec{B}_1[k] \oplus \vec{B}_2[k]$ for each k . When $\vec{B}_1[k] \oplus \vec{B}_2[k] = 1$, it indicates that k is a pertinent index. The recipient then records it into a set I . At last, through PIR, the recipient can obviously retrieve the pertinent messages indexed by the elements in set I . Furthermore, to prevent the servers from learning the number of messages that the recipient R_i aims to retrieve, the recipient only retrieves one message at a time when performing PIR.

4.4 Deletion

As mentioned in the previous section, Server_1 and Server_2 maintain vectors \vec{Y}_1 and \vec{Y}_2 for an interval, respectively, such that if m_i has been retrieved in this interval, $\vec{Y}_1[i] \oplus \vec{Y}_2[i] \neq 0^{\ell_2}$, otherwise $\vec{Y}_1[i] \oplus \vec{Y}_2[i] = 0^{\ell_2}$. Obtaining \vec{Y}_2 can let Server_1 learn which indexes are pertinent to each index retrieval request. To solve this problem, the two servers invoke $\mathcal{F}_{\text{MPOPRF}}$ such that Server_1 obtains $F_k(\vec{Y}_1[i])$ where $i \in |\vec{Y}_1|$ and Server_2 obtains the PRF key k . Furthermore, Server_2 can use PRF key k to calculate $F_k(\vec{Y}_2[i])$ where $i \in |\vec{Y}_2|$. Once receiving $F_k(\vec{Y}_2[i])$, Server_1 compares $F_k(\vec{Y}_2[i])$ with $F_k(\vec{Y}_1[i])$. If $F_k(\vec{Y}_1[i]) \neq F_k(\vec{Y}_2[i])$, Server_1 deletes the message $\vec{M}_1[i]$ and the label $\vec{X}_1[i]$, and sends index i to Server_2 . Server_2 also deletes the message $\vec{M}_2[i]$ and the label $\vec{X}_2[i]$. Finally, the two servers reset \vec{Y}_1 and \vec{Y}_2 as zero vectors, respectively, for the next interval.

4.5 Security Proof

Correctness. For an address Addr_{R_i} , the labels generated by the sender are denoted as $L_{R_i}^1$ and $L_{R_i}^2$. A recipient aiming to retrieve messages pertinent to address $\text{Addr}_{R'_i}$ needs to generate labels $L_{R'_i}^1$ and $L_{R'_i}^2$. Obviously, only if $\text{Addr}_{R_i} = \text{Addr}_{R'_i}$, then $L_{R_i}^1 \cdot (L_{R'_i}^1)^{-1} = (L_{R_i}^2)^{-1} \cdot L_{R'_i}^2$ (i.e., $L_{R_i}^1 \cdot L_{R_i}^2 = L_{R'_i}^1 \cdot L_{R'_i}^2$), thus resulting in $H_1(L_{R_i}^1 \cdot (L_{R'_i}^1)^{-1}) = H_1((L_{R_i}^2)^{-1} \cdot L_{R'_i}^2)$. Then, after performing PET with inputs $H_1(L_{R_i}^1 \cdot (L_{R'_i}^1)^{-1})$ and $H_1((L_{R_i}^2)^{-1} \cdot L_{R'_i}^2)$, the two servers will obtain b_1 and b_2 , respectively, such that $b_1 \oplus b_2 = 1$. Upon obtaining b_1 and b_2 , the recipient can know the corresponding index is pertinent, and retrieve the corresponding message through PIR.

We remark that collisions of H_1 would incur a correctness error, i.e., $L_{R_i}^1 \cdot (L_{R'_i}^1)^{-1} \neq (L_{R_i}^2)^{-1} \cdot L_{R'_i}^2$ but $H_1(L_{R_i}^1 \cdot (L_{R'_i}^1)^{-1}) = H_1((L_{R_i}^2)^{-1} \cdot L_{R'_i}^2)$. To ensure that the probability of collisions happening is less than $2^{-\lambda}$, we set the output length ℓ_1 of H_1 to be at least $\lambda + \log N$.

As for the deletion procedure, for each message m_i whose index is identified as pertinent, one of the two servers will XOR randomness $H_2(\text{sn}||i)$ to the corresponding element in vector \vec{Y}_j ($j = 1$ or 2). For each message m_i whose index is not identified as pertinent, either both servers will XOR randomness $H_2(\text{sn}||i)$, or neither does. Obviously, at the end of an interval, if m_i is retrieved, $\vec{Y}_1[i] \neq \vec{Y}_2[i]$ and $F_k(\vec{Y}_1[i]) \neq F_k(\vec{Y}_2[i])$, otherwise, $\vec{Y}_1[i] = \vec{Y}_2[i]$ and $F_k(\vec{Y}_1[i]) = F_k(\vec{Y}_2[i])$.

Protocol Π_{Retrieve}

A recipient holding $(\text{Addr}_{R_i}, k_{R_i})$ wants to retrieve the messages pertinent to her.

Requesting indexes:

The recipient R_i does the following:

- Randomly choose $\text{sn} \leftarrow \{0, 1\}^\ell$, and randomly split k_{R_i} into $k_{R_i,1}$ and $k_{R_i,2}$ such that $k_{R_i} = k_{R_i,1} + k_{R_i,2}$;
- Generate $L_{R_i,1} = g^{k_{R_i,1}}$, $L_{R_i,2} = g^{k_{R_i,2}}$, and $L_{R_i,1}^{-1}$;
- Set $x = (L_{R_i,1}, L_{R_i,2}, g)$ and $w = (k_{R_i,1}, k_{R_i,2})$ and send $\langle \text{PROVE}, x, w \rangle$ to $\mathcal{F}_{\text{NIZK}}$ and receive (π_1, π_2) ;
(The language is $L_{R_i,j} = g^{k_{R_i,j}}$ for $j \in \{1, 2\}$.)
- Send $(\text{sn}, L_{R_i,1}^{-1}, \pi_1, L_{R_i,1})$ to Server_1 , and $(\text{sn}, L_{R_i,2}, \pi_2)$ to Server_2 .

Responding for the request of indexes:

The two servers maintain a set S , that records the serial numbers of valid requests in this time interval and will be cleared after each deletion (as shown in Figure 14).

- Server_1 and Server_2 receive $(\text{sn}, t_1, \pi_1, t'_1)$ and (sn, t_2, π_2) respectively.
- Server_1 : Send $\langle \text{VERIFY}, t'_1, \pi_1 \rangle$ to $\mathcal{F}_{\text{NIZK}}$, if receiving 1 from $\mathcal{F}_{\text{NIZK}}$, $t_1 \cdot t'_1 = 1$ and $\text{sn} \notin S$, then send $\langle \text{sn}, \text{OK} \rangle$ to Server_2 , otherwise, ignore the request;
- Server_2 : Send $\langle \text{VERIFY}, t_2, \pi_2 \rangle$ to $\mathcal{F}_{\text{NIZK}}$, if receiving 1 from $\mathcal{F}_{\text{NIZK}}$ and $\text{sn} \notin S$, then sends $\langle \text{sn}, \text{OK} \rangle$ to Server_1 , otherwise, ignore the request;
- After receiving $\langle \text{sn}, \text{OK} \rangle$ from the other server, Server_j ($j \in \{1, 2\}$) initializes an empty bit vector \vec{B}_j , and does the following for each $k \in [|\vec{X}_j|]$:
 - Server_j computes $a_{k,j} = H_1(t_{k,j} \cdot t_j)$, where $t_{k,j} = \vec{X}_j[k]$, $j \in \{1, 2\}$ and H_1 is a hash function: $\mathbb{G} \rightarrow \{0, 1\}^{\ell_1}$;
 - Server_1 and Server_2 invoke \mathcal{F}_{PET} with inputs $a_{k,1}$ and $a_{k,2}$, and obtain $b_{k,1}$ and $b_{k,2}$ respectively;
 - Server_j ($j \in \{1, 2\}$) appends $b_{k,j}$ into vector \vec{B}_j ;
 - Server_j ($j \in \{1, 2\}$) updates $\vec{Y}_j[k] = \vec{Y}_j[k] \oplus b_{k,j} \cdot H_2(\text{sn}||k)$ where H_2 is a hash function: $\{0, 1\}^{\ell+\ell'} \rightarrow \{0, 1\}^{\ell_2}$;
- Server_j ($j \in \{1, 2\}$) sends \vec{B}_j to the recipient R_i .

Recovering indexes:

The recipient R_i upon receiving \vec{B}_j from Server_j :

- Set $b = \min\{|\vec{B}_1|, |\vec{B}_2|\}$, and initialize a set $I = \emptyset$;
- For each $k \in [b]$, if $\vec{B}_1[k] \oplus \vec{B}_2[k] = 1$, add k into set I .

Retrieving pertinent messages:

For each $i \in I$, the recipient R_i does the following:

- Compute $(q_1^i, q_2^i) \leftarrow \text{PIR}.Q(|\vec{M}_j|, i)$;
- Send q_1^i to Server_1 , and q_2^i to Server_2 ;
- Upon receiving (a_1^i, a_2^i) from the servers, computes $m_i = \text{PIR}.M(i, a_1^i, a_2^i)$;

For each $i \in I$, upon receiving q_j^i , Server_j does the following:

- Compute $a_j^i = \text{PIR}.A(\vec{M}_j, q_j^i)$, and send a_j^i to recipient R_i ;

Figure 13: Protocol for retrieval.

Likewise, H_2 and $F_k(\cdot)$ can also incur collisions, thus leading to correctness error. We also set the output length ℓ_2 of H_2 and the output length of ℓ_3 of $F_k(\cdot)$ to be at least $\lambda + \log N$.

Security. Next, we show the UC security of HomeRun in the $\{\mathcal{G}_{\text{Ledger}}, \mathcal{F}_{\text{PET}}, \mathcal{F}_{\text{MPOPRF}}, \mathcal{F}_{\text{NIZK}}\}$ -hybrid world.

Theorem 1. Assuming the security of CPA-secure encryption, computational secrecy of PIR and hard Discrete Logarithm (DL) problem, the protocol $\Pi_{\text{OMR}} = (\Pi_{\text{Initialize}}, \Pi_{\text{SEND}}, \Pi_{\text{Retrieve}}, \Pi_{\text{Delete}})$ realizes the \mathcal{F}_{OMR}

Protocol Π_{Delete}

The two servers delete the messages that have been retrieved periodically.

Deletion:

- **Server₁** and **Server₂** invoke $\mathcal{F}_{\text{MPOPREF}}$:
 - **Server₁** acts as P_0 with input \vec{Y}_1 ;
 - **Server₁** obtains $J_1 = \{F_k(\vec{Y}_1[i])\}_{i \in [|\vec{Y}_1|]}$, and **Server₂** obtains the key k ;
- **Server₂** computes $J_2 = \{F_k(\vec{Y}_2[i])\}_{i \in [|\vec{Y}_2|]}$ and sends J_2 to **Server₁**;
- **Server₁** does the following:
 - Initializes a set $I = \emptyset$;
 - Set $n = \min\{|J_1|, |J_2|\}$;
 - For each $i \in [n]$, if $J_1[i] \neq J_2[i]$, add i into set I and delete $\vec{M}_1[i]$ and $\vec{X}_1[i]$;
 - Send set I to **Server₂**;
 - Reset vector \vec{Y}_1 as a zero vector;
- For each $i \in I$, **Server₂** deletes $\vec{M}_2[i]$ and $\vec{X}_2[i]$. Then, reset vector \vec{Y}_2 as a zero vector.

Figure 14: Protocol for deletion.

ideal functionality in the $\{\mathcal{G}_{\text{Ledger}}, \mathcal{F}_{\text{PET}}, \mathcal{F}_{\text{MPOPREF}}, \mathcal{F}_{\text{NIZK}}\}$ -hybrid world.

Proof. To prove that Π_{OMR} UC-realizes the \mathcal{F}_{OMR} functionality, we show that there exists a simulator \mathcal{S} interacting with \mathcal{F}_{OMR} functionality that generates a transcript that is indistinguishable from the transcript generated by the real-world adversary \mathcal{A} in the protocol Π_{OMR} .

We consider the following different cases of corruption and define a simulator for each case:

- Simulator \mathcal{S}_N for the case when neither sender nor receiver is corrupt and only one of the servers is corrupt.
- Simulator \mathcal{S}_s for the case when the sender is corrupt and colludes with one of the servers.
- Simulator \mathcal{S}_r for the case when the receiver is corrupt and colludes with one of the servers.
- Simulator \mathcal{S}_{sr} when both the sender and receiver are corrupt and collude with one of the servers.

We discuss these simulators in more detail below.

Case 1: When neither the senders nor the receivers are corrupt. We consider the case when only one of the two servers is corrupt. In this case, the simulator interacts with the corrupt server (the adversary) and the ideal functionality \mathcal{F}_{OMR} and computes a transcript that is indistinguishable from the real-world protocol. The simulator is shown in Figure 15.

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

- **Hyb₀**: The real world protocol.
- **Hyb₁**: This hybrid is the same as the previous hybrid except that the random tape of of the corrupted server is chosen by the simulator. Since the corrupted server is semi-honest, the two hybrids are indistinguishable.
- **Hyb₂**: This hybrid is the same as the previous hybrid except that c_1 is replaced by encryptions to 0, respectively. By the CPA security of the underlying encryption scheme, the two hybrids are indistinguishable.
- **Hyb₃**: This hybrid is the same as the previous hybrid except that the proofs sent to the corrupt server are replaced by simulated proofs. By the zero-knowledge property of the $\mathcal{F}_{\text{NIZK}}$ functionality, these hybrids are indistinguishable.
- **Hyb₄**: This hybrid is the same as the previous hybrid, except that the request of messages via PIR is replaced by the output of S_{PIR} . By the computational secrecy of the underlying PIR scheme, the two hybrids are indistinguishable.
- **Hyb₅**: This hybrid is identical to the previous hybrid except that we use the internally simulated \mathcal{F}_{PET} ideal functionality.
- **Hyb₆**: This hybrid is identical to the previous hybrid except that we use the internally simulated

Simulator \mathcal{S}_N

The simulator \mathcal{S}_N internally simulates the $\mathcal{G}_{\text{Ledger}}$, \mathcal{F}_{PET} and the $\mathcal{F}_{\text{MPOPRF}}$ ideal functionalities towards the adversary.

Simulating $\Pi_{\text{Initialize}}$:

1. Initialization:

- w.l.o.g. assuming Server_2 is corrupted, choose a uniformly distributed random tape for the corrupted Server_2 (semi-honest, \mathcal{A}), and run \mathcal{A} 's code to generate (pk_2, sk_2) .
- On behalf of honest Server_1 generate (pk_1, sk_1) , and send $\langle \text{SUBMIT}, pk_1 \rangle$ to the simulated $\mathcal{G}_{\text{Ledger}}$. (Note that the simulator \mathcal{S}_s knows the key pairs of the honest server and the corrupted server.)
- On behalf of R_i , sample $k_{R_i} \leftarrow \mathbb{Z}_p$ and compute $\text{Addr}_{R_i} = g^{k_{R_i}}$ and send Addr_{R_i} to \mathcal{A} .

Simulating Π_{Send} :

1. **Publishing to the board:** Upon receiving $\langle \text{SEND}, msg \rangle$ from the \mathcal{F}_{OMR} functionality, compute $c_1 = \text{Enc}(pk_1, 0)$ and $c_2 = \text{Enc}(pk_2, R)$, where $R \leftarrow \mathbb{G}$. Then send $\langle \text{SUBMIT}, msg, c_1, c_2 \rangle$ to the simulated $\mathcal{G}_{\text{Ledger}}$ functionality. Then send $\langle \text{SEND}, \text{OK} \rangle$ to \mathcal{F}_{OMR} .
2. **Preparing for retrievals:** Upon receiving READ from \mathcal{A} (Server_1 or Server_2) on behalf of the $\mathcal{G}_{\text{Ledger}}$ functionality, the simulator responds with $\langle \text{state}, \text{buffer} \rangle$ that includes all (msg, c_1, c_2) tuples. The simulator assigns an index i for each msg according to a predefined rule, and sets $\vec{M}_j[i] = msg$ on behalf of honest Server_j .

Simulating Π_{Retrieve} :

Upon receiving $\langle \text{RECEIVE} \rangle$ from the \mathcal{F}_{OMR} functionality,

1. Requesting indices:

- If Server_1 is corrupt: sample random $sn \leftarrow \{0, 1\}^\ell$ and sample $L \leftarrow \mathbb{G}$ and compute L^{-1} . Internally simulate $\mathcal{F}_{\text{NIZK}}$ and receive a proof π and send $\langle sn, L, \pi, L^{-1} \rangle$ to Server_1 (\mathcal{A}) and internally store (L, π) .
- If Server_2 is corrupt: sample random $sn \leftarrow \{0, 1\}^\ell$ and sample $L \leftarrow \mathbb{G}$. Internally simulate $\mathcal{F}_{\text{NIZK}}$ and receive a proof π and send $\langle sn, L, \pi \rangle$ to Server_2 (\mathcal{A}) and internally store (L, π) .

2. Response to request:

- If Server_1 is corrupt: receive $\langle \text{VERIFY}, t, \pi \rangle$ from \mathcal{A} on behalf of $\mathcal{F}_{\text{NIZK}}$, if (t, π) is stored, respond with 1 else respond with 0. Upon receiving $\langle sn, \text{OK} \rangle$ from \mathcal{A} , send $\langle sn, \text{OK} \rangle$ back to \mathcal{A} .
- If Server_2 is corrupt: send $\langle sn, \text{OK} \rangle$ to \mathcal{A} . Upon receiving $\langle \text{VERIFY}, t, \pi \rangle$ from \mathcal{A} on behalf of $\mathcal{F}_{\text{NIZK}}$, if (t, π) is stored, respond with 1 else respond with 0. Wait to receive $\langle sn, \text{OK} \rangle$ from \mathcal{A} .
- For each invocation of \mathcal{F}_{PET} for $k \in [|\vec{M}_j|]$, sample a bit $b_k \in \{0, 1\}$ and send back b_k to \mathcal{A} .

3. Retrieving pertinent messages:

- Invoke $S_{\text{PIR}}(1, \lceil \log n \rceil)$ and receive q_1^i, q_2^i for a random index $i \in [|\vec{M}_j|]$.
- Send q_j^i to \mathcal{A} (i.e., Server_j) and receive back a_j^i .
- Send $\langle \text{RECEIVE}, \text{OK} \rangle$ to \mathcal{F}_{OMR} .

Simulating Π_{Deletion} :

Upon receiving $\langle \text{DELETE}, D \rangle$ from the \mathcal{F}_{OMR} functionality,

1. Deletion:

- If Server_1 is corrupt:
 - Upon receiving \vec{Y}_1 from \mathcal{A} , internally simulate $\mathcal{F}_{\text{MPOPRF}}$ and randomly sample $(x_1, \dots, x_{|\vec{Y}_1|})$ and send $(x_1, \dots, x_{|\vec{Y}_1|})$ to the \mathcal{A} .
 - For each $\vec{M}_2[i] \in D$, send corresponding x_i to \mathcal{A} .
 - Receive a set I from \mathcal{A} , and send $\langle \text{DELETE}, \text{OK} \rangle$ to \mathcal{F}_{OMR} .
- If Server_2 is corrupt:
 - Internally simulate $\mathcal{F}_{\text{MPOPRF}}$ and sample a random key k . Send k to \mathcal{A} .
 - Upon receiving J_2 from \mathcal{A} , for each $\vec{M}_1[i] \in D$, add i to I .
 - Send I to \mathcal{A} , and send $\langle \text{DELETE}, \text{OK} \rangle$ to \mathcal{F}_{OMR} .

Figure 15: Simulator when only one of the servers is corrupt

$\mathcal{F}_{\text{MPOPRF}}$ ideal functionality. Since this hybrid is identical to the ideal world, we show that the real and ideal worlds are indistinguishable.

Simulator \mathcal{S}_s

The simulator \mathcal{S}_s internally simulates the $\mathcal{G}_{\text{Ledger}}$, \mathcal{F}_{PET} and the $\mathcal{F}_{\text{MPOPRF}}$ ideal functionalities towards the adversary.

Simulating $\Pi_{\text{Initialize}}$: same as in Figure 15.

Simulating Π_{Send} :

- Upon receiving $\langle \text{SUBMIT}, msg, c_1, c_2 \rangle$ on behalf of the simulated $\mathcal{G}_{\text{Ledger}}$ functionality, decrypt c_1 to $R_{(1)}$ using sk_1 and decrypt c_2 to $R_{(2)}$ using sk_2 , and compute $R = R_{(1)} \cdot R_{(2)}$. If $R_{(1)} \in \mathbb{G}$ and $R_{(2)} \in \mathbb{G}$, send $\langle \text{SEND}, R, msg \rangle$ to \mathcal{F}_{OMR} .

Simulating $\Pi_{\text{Retrieval}}$: same as in Figure 15.

Simulating Π_{Deletion} : same as in Figure 15.

Figure 16: Simulator when only one of the servers and the sender are corrupt

Case 2: When one of the servers and the sender are corrupt. We construct the simulator in Figure 16, and then present proof by hybrids to show that the simulated world and the real world are indistinguishable.

- **Hyb₀**: The real world protocol.
- **Hyb₁**: This hybrid is the same as the previous hybrid except that the random tape of of the corrupted server is chosen by the simulator. Since the corrupted server is semi-honest, the two hybrids are indistinguishable.
- **Hyb₂**: This hybrid is the same as the previous hybrid except that the proofs sent by the receiver are replaced by simulated proofs. By the zero-knowledge property of the $\mathcal{F}_{\text{NIZK}}$ functionality, these hybrids are indistinguishable.
- **Hyb₃**: This hybrid is the same as the previous hybrid, except that the request of messages via PIR is replaced by the output of S_{PIR} . By the computational secrecy of the underlying PIR scheme, the two hybrids are indistinguishable.
- **Hyb₄**: This hybrid is identical to the previous hybrid except that we use the internally simulated \mathcal{F}_{PET} ideal functionality.
- **Hyb₅**: This hybrid is identical to the previous hybrid except that we use the internally simulated $\mathcal{F}_{\text{MPOPRF}}$ ideal functionality. Since this hybrid is identical to the ideal world, we show that the real and ideal worlds are indistinguishable.

Case 3: When one of the servers and the receiver are corrupt. We construct the simulator in Figure 17, and present a proof by hybrids to show that the real world and the simulated worlds are indistinguishable.

- **Hyb₀**: The real world protocol.
- **Hyb₁**: This hybrid is the same as the previous hybrid except that the random tape of of the corrupted server is chosen by the simulator. Since the corrupted server is semi-honest, the two hybrids are indistinguishable.
- **Hyb₂**: This hybrid is the same as the previous hybrid except that c_1 is replaced by encryptions to 0, respectively. By the CPA security of the underlying encryption scheme, the two hybrids are indistinguishable.
- **Hyb₃**: This hybrid is the same as the previous hybrid, except that the simulator may abort upon receiving a retrieval request from the adversary such that $t_1 \cdot t_2$ corresponds to that of an honest party. By the discrete log assumption, this occurs with negligible probability and therefore the two hybrids are indistinguishable.
- **Hyb₄**: This hybrid is the same as the previous hybrid except that the proofs sent by the receiver are replaced by simulated proofs. By the zero-knowledge property of the $\mathcal{F}_{\text{NIZK}}$ functionality, these hybrids are indistinguishable. The adversary extracts the Addr_R of the receiver from the witness.
- **Hyb₅**: This hybrid is identical to the previous hybrid except that we use the internally simulated \mathcal{F}_{PET} ideal functionality and the bit string sent to the malicious receiver is constructed as in the simulator.

Simulator \mathcal{S}_r

The simulator \mathcal{S}_s internally simulates the $\mathcal{G}_{\text{Ledger}}$, \mathcal{F}_{PET} and the $\mathcal{F}_{\text{MPOPRF}}$ ideal functionalities towards the adversary.

Simulating $\Pi_{\text{Initialize}}$: same as in Figure 15.

Simulating Π_{Send} : same as in Figure 15.

Simulating $\Pi_{\text{Retrieval}}$:

- **Requesting indices:** Upon receiving $(\text{PROVE}, (t_1, t_2, g), (k_1, k_2))$ from \mathcal{A} ,
 - If $t_1 \cdot t_2$ corresponds to an honest recipient's Addr_R , then abort.
 - Else forward to the internally simulated $\mathcal{F}_{\text{NIZK}}$ functionality and return (π_1, π_2) to \mathcal{A} . Store $((t_1, t_2, g), (k_1, k_2), \pi_1, \pi_2)$
- **Response to request:**
 - Upon receiving $(\text{sn}, L_{R_{i,1}}^{-1}, \pi_1, L_{R_{i,1}})$ and $(\text{sn}, L_{R_{i,2}}, \pi_2)$ from \mathcal{A} , check if $(\text{sn}, L_{R_{i,1}}^{-1}, \pi_1, L_{R_{i,1}})$ and $(\text{sn}, L_{R_{i,2}}, \pi_2)$ exist in the list of stored proofs.
 - Compute $\text{Addr}_R = t_1 \cdot t_2$.
 - Send $(\text{RETRIEVE}, R)$ to the \mathcal{F}_{OMR} functionality and receive the set I .
 - Simulating \mathcal{F}_{PET} : For $\vec{M}_j[i] \in I$, set $\vec{b}_{1-j}[i] = 1 \oplus \vec{b}_j[i]$, output \vec{b}_{1-j} to \mathcal{A} (Server_{1-j}).
 - Send \vec{b}_j to the \mathcal{A} .
- **Retrieving pertinent messages:** Upon receiving q_j^i from \mathcal{A} (corrupt receiver), compute $a_j^i = \text{PIR}.A(\vec{M}_j, q_j^i)$ and send a_j^i back to \mathcal{A} .

Simulating Π_{Deletion} : same as in Figure 15.

Figure 17: Simulator when only one of the servers and the receiver are corrupt

- **Hyb₆**: This hybrid is identical to the previous hybrid except that we use the internally simulated $\mathcal{F}_{\text{MPOPRF}}$ ideal functionality. Since this hybrid is identical to the ideal world, we show that the real and ideal worlds are indistinguishable.

Case 4: When one of the servers, the receiver and sender are corrupt. We construct the simulator in Figure 18, and present a proof by hybrids to show that the real world and the simulated worlds are indistinguishable.

- **Hyb₀**: The real world protocol.
- **Hyb₁**: This hybrid is the same as the previous hybrid except that the random tape of the corrupted server is chosen by the simulator. Since the corrupted server is semi-honest, the two hybrids are indistinguishable.
- **Hyb₂**: This hybrid is the same as the previous hybrid except that c_1 is replaced by encryptions to 0. By the CPA security of the underlying encryption scheme, the two hybrids are indistinguishable.
- **Hyb₃**: This hybrid is the same as the previous one except that for a malicious sender, the proofs are simulated via the $\mathcal{F}_{\text{NIZK}}$ functionality and the simulator extracts the Addr_R of the receiver.
- **Hyb₄**: This hybrid is the same as the previous hybrid, except that the simulator may abort upon receiving a retrieval request from the adversary such that $t_1 \cdot t_2$ corresponds to that of an honest party. By the discrete log assumption, this occurs with negligible probability and therefore the two hybrids are indistinguishable.
- **Hyb₅**: This hybrid is the same as the previous hybrid except that the proofs sent by the receiver are replaced by simulated proofs. By the zero-knowledge property of the $\mathcal{F}_{\text{NIZK}}$ functionality, these hybrids are indistinguishable. The adversary extracts the Addr_R of the receiver from the witness.
- **Hyb₆**: This hybrid is identical to the previous hybrid except that we use the internally simulated \mathcal{F}_{PET} ideal functionality and the bit string sent to the malicious receiver is constructed as in the simulator.
- **Hyb₇**: This hybrid is identical to the previous hybrid except that we use the internally simulated $\mathcal{F}_{\text{MPOPRF}}$ ideal functionality. Since this hybrid is identical to the ideal world, we show that the real and ideal worlds are indistinguishable. □

Simulator \mathcal{S}_{sr}

The simulator \mathcal{S}_s internally simulates the $\mathcal{G}_{\text{Ledger}}$, \mathcal{F}_{PET} and the $\mathcal{F}_{\text{MPOPRF}}$ ideal functionalities towards the adversary.

Simulating $\Pi_{\text{Initialize}}$: same as in Figure 15.

Simulating Π_{Send} : same as in Figure 16.

Simulating $\Pi_{\text{Retrieval}}$: same as in Figure 17.

Simulating Π_{Deletion} : same as in Figure 15.

Figure 18: Simulator when only one of the servers and the sender and receiver are corrupt

5 Performance Evaluation

In this section, we experimentally evaluate HomeRun and compare with the previous works.

5.1 Implementation

Benchmarking Environment. We implement HomeRun in C++, which is available on GitHub: <https://github.com/yanxue820/twoPartyOMR>. Our experiments are conducted on a server equipped with two Intel Xeon Silver 4116 CPUs (2.10GHz) and 128GB RAM, running Ubuntu. We evaluate HomeRun in two network settings, LAN network with 10Gbps bandwidth and 0.02 ms RTT and WAN network with 400Mbps and 100ms RTT, which are emulated using Linux tc command. The garbled-circuit-based scheme in [MSS⁺22] is also a two-server scheme, but we only evaluate it in the LAN network as their code does not support simulating networks.

The cyclic group \mathbb{G} is realized using the elliptic curve secp256r1 provided in OpenSSL [Ope]. HomeRun leverages the Private Equality Test (PET), Multi-point Oblivious PRF (MP-OPRF) and Private Information Retrieval (PIR) as building blocks. We implement PET based on the design in [CGS22], setting the length of a leaf node as 1-bit. The MP-OPRF protocol is instantiated by using the scheme in [RS21]. We leverage the two-party PIR designed in [KOR19], which is based on Distributed Point Function (DPF) [BGI16].

Parameters. We set the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$. The bit-length ℓ_1 of the input of PET is set to 64. We set both the bit-length ℓ_2 of the element in vector \vec{Y}_j and the bit-length ℓ_3 of the PRF $F_k(\cdot)$ output as 128.

5.2 Performance Comparison

In this section, we provide a thorough performance comparison of HomeRun against prior works [MSS⁺22, LT22, BLMG21, JLM23]⁹. These schemes can be divided into two categories: (1) Detection schemes focus on detecting the indexes of pertinent messages without being dedicated to studying how the recipient obtains the pertinent messages using these indexes; (2) Retrieval schemes allow the recipient to obtain the pertinent messages. HomeRun integrates a detection scheme with a two-party PIR for message retrieval. We first present the performance of the detection component, followed by an analysis of the added overhead introduced by the PIR.

Next, assuming that there are 2^{19} messages (approximate number of Bitcoin transactions per day as mentioned in [MSS⁺22, LT22]) on the bulletin board, we compare the computation and communication costs and summarize the results in Table 2.

Number of pertinent messages. A main advantage of HomeRun is that the number of pertinent messages for a recipient is unlimited. The prior schemes [BLMG21, JLM23] also enjoy the feature. Therefore, in

⁹FMD1 [BLMG21] is more efficient than FMD2 [BLMG21]. So, we only show the performance of FMD1 here. We could not re-evaluate the work of [JLM23] in our environment as their current codebase is not side-channel resistant. We will conduct a comprehensive comparison once a secure implementation of [JLM23] becomes available. For now, we include the experimental results presented in their paper as is.

Table 2, we use n to denote the number of pertinent messages for our work and [BLMG21, JLM23]. In contrast, other schemes only allow a recipient to receive up to a pre-determined number of pertinent messages. For the sake of fair comparisons, we set the pre-determined number at 50 for these schemes. There are 2^{19} messages on the bulletin board and each recipient can receive at most 50 messages, so we set the number of rows in PS1/PS2 [MSS+22] as 10485.

While we do not limit the number of pertinent messages for a recipient, the extra overhead from the PIR is dependent on the number of pertinent messages. Therefore, to ensure fair comparisons, we detail the additional overhead for retrieving 50 messages in the last column of Table 2.

Server(s) total computation time. The server(s) is responsible for helping the recipient to identify pertinent messages. Roughly speaking, if a sender wants to send a message to a recipient, the server(s) needs to do two steps: (1) process the label from the sender *before* receiving a request from the recipient; (2) generate a response *after* receiving a request from the recipient. Except for the schemes designed in [MSS+22], other schemes in the first step only need to add the label (and the message) from the sender into a list, so the total computation time refers to the runtime of the second step. As for PS1/PS2 [MSS+22], the total computation time involves the two steps. We denote the runtimes of the two steps as T_5^a and T_5^r (“a” means “addition” and “r” means “response”), respectively. From Table 2, we can see that the total computation time of the servers in HomeRun is significantly shorter than that in prior works that do not rely on TEE. More specifically, when using a single thread, HomeRun achieves an online runtime of 4.21s in the LAN setting and 5.09s in the WAN setting, which is comparable with the performance of TEE-based PS1. Furthermore, by scaling to 16 threads, the online runtime is further optimized to 0.49s in the LAN setting and 0.96s in the WAN setting.

Recipient reconstruction time. After receiving the response from the server(s), the recipient needs to reconstruct the pertinent indexes (or messages). We can see that the reconstruction times in these schemes, except for PS1 [MSS+22] and FMD1 [BLMG21], are much less than 1 second. The recipient in PS1 needs to decrypt 50 ciphertexts. The goal of [BLMG21] is to reduce the number of messages that the recipient needs to test, rather than give the exact pertinent messages. More specifically, the server sends $N' = (2^{19} - n)p + n$ messages to the recipient, where p is a false positive probability. Then, the recipient identifies the n pertinent messages from the N' messages. The test time for each message, denoted as T , is determined by the particular application¹⁰. Therefore, we only give the expression, rather than the precise value.

Latency. The latency refers to the duration between the recipient initiating a request and receiving the relevant indexes or messages. For HomeRun and the schemes in [LT22, BLMG21], the latency is the sum of the server(s) computation time and the recipient reconstruction time. Whereas, when computing the latency of PS1/PS2 [MSS+22] and S-PS [JLM23], we need to subtract the runtime before receiving the request from the total computation time of the server. It is worth mentioning that the latency of HomeRun is independent of the number of pertinent messages. More specifically, regardless of how many pertinent messages the recipient can receive, the latency of HomeRun is only 0.96s using 16 threads in the WAN setting, which is greater than that of PS2 and S-PS, comparable with that of TEE-based PS1 and significantly less than that of other schemes. Note that the latency of S-PS is linear in n . When n is greater than 1000, the latency of S-PS is comparable with ours.

Deletion time. In these schemes, only HomeRun can support deletion. To prevent the servers from knowing which messages are pertinent to a recipient, we require the two servers to batch-delete all retrieved messages and corresponding labels at regular intervals (e.g., every day). No matter how many messages need to be deleted, HomeRun takes only 4.47s (resp. 2.51s) using a single thread in the WAN (resp. LAN) setting to delete these messages.

Label size for each message. The server(s) needs to maintain a label for each message to identify if the message is pertinent. In HomeRun, the label is an elliptic curve point, which occupies a size of 33 bytes using compressed representation. As indicated in Table 2, our label size is the most compact.

Recipient → Sender. The recipient needs to send an “address” to a sender, such that the sender can send messages to her. In HomeRun, the address is an elliptic curve point of size 33 bytes, which is much smaller

¹⁰For instance, in cryptocurrencies, the recipient needs to use a private key to test whether a transaction belongs to her.

#Msgs on bulletin board (N)	2^{16}	2^{18}	2^{20}	2^{22}
Detector total runtime (online, sec)	1.07	3.02	10.31	40.34
Deletion time (sec)	2.73	3.57	5.93	14.45
Server \leftrightarrow Server (online, byte)	1.97M	7.87M	31.50M	126M
Digest size (byte)	16K	64K	256K	1024K

Table 3: Costs under different N (the number of messages on bulletin board) in WAN (400Mbps bandwidth, 100ms RTT) setting, using a single thread.

than that in the schemes by [BLMG21, LT22] and comparable with that in the designs of [MSS+22].

Recipient \rightarrow Server(s). The recipient needs to send a request to the server(s). In HomeRun, the recipient needs to send a share of address (33 bytes) and a proof (65 bytes) to each server. In order to reduce the computation cost of Server_2 , we require the recipient to also send an inversion (33 bytes) of a share to Server_2 . The total communication cost that the recipient sends to the two servers is 229 bytes, which is much smaller than that in the schemes by [LT22] and comparable with that in the designs of [BLMG21, MSS+22, JLM23]. It is worth mentioning that the communication cost in [BLMG21] varies with the false positive probability p . As p increases, the communication overhead correspondingly increases.

Server \leftrightarrow Server. Only HomeRun and PS2 in [MSS+22] rely on two servers. The total communication cost between servers of HomeRun is about $6.4\times$ less than that of PS2. Moreover, our online communication cost is only 15.75M bytes.

Digest size. The server(s) responds with a digest to the recipient, such that the recipient can recover the pertinent indexes or messages. In HomeRun, for each message on the bulletin board, each server needs to send 1 bit to the recipient. Therefore, the digest size is $2N$ bits, i.e., 128K bytes when $N = 2^{19}$, which is much smaller than that in the designs of [BLMG21, LT22]¹¹. Although HomeRun has a larger digest size than the schemes presented by [MSS+22, JLM23], it is worth noting that HomeRun supports an unlimited number of pertinent indexes, while PS1/PS2 limit it up to 50 and the digest size of S-PS is linear in n .

Combining with PIR. After obtaining the pertinent indexes, the recipient can employ PIR to retrieve the corresponding messages. More specifically, we assume that the message size is 640 bytes. For each pertinent index, the recipient takes 0.008ms to generate two DPF keys, each of size 249 bytes, and sends them to the two servers, respectively. Each server spends 27.44ms to generate a 640-byte digest, which is subsequently sent back to the recipient. Finally, the recipient recovers the pertinent message by applying the XOR operation to the two digests.

In order to give a fair comparison, we assume a scenario with 50 pertinent messages. The overhead of the recipient is an extra 0.4ms for DPF key generation and an additional transfer of 12,450 bytes to each server. On the server side, generating the digests requires 1.37s in a single thread. We show the extra overhead of retrieving 50 messages through PIR in the last column of Table 2.

5.3 Evaluation under Different N

In Table 3, we show the experimental results for different numbers (denoted as N) of messages on the bulletin board. Similarly to previous work [BLMG21, MSS+22, LT22], our detector total runtime (online) is $O(N)$. HomeRun requires each server to send N bits to the recipient, leading to the digest size of $O(N)$. Note that although the digest size in the previous work [BLMG21, MSS+22, LT22] is sublinear in N , their solutions limit the number of pertinent messages for a recipient. In contrast, HomeRun allows an unlimited number of pertinent messages for each recipient. In addition, our deletion time and online communication cost between the two servers are both $O(N)$.

While the experimental results demonstrate that HomeRun is sufficiently efficient for practical use, we discuss how to further improve performance through horizontal scaling in Section 6.4.

¹¹For FMD1, $|M|$ is the sum of the sizes (in bytes) of a message and its label.

6 Extensions

6.1 Authentication of Recipients

In this work, the two non-colluding and semi-honest servers need to separately verify a NIZK proof to guarantee that only the holder of a secret key can retrieve the corresponding messages (see Section 2.2 or Section 4.3). However, if one of two servers and a receiver collude and the server does not verify the proof, the receiver can choose the address of an honest party Addr_h and obtain the messages pertinent to Addr_h . While the attack cannot be successful assuming the two servers are semi-honest, the malicious behavior is not easily detectable¹² in practice. Therefore, we still discuss how to prevent this attack below.

The key idea is that the sender generates a new one-time address for the recipient each time, so that the adversary cannot know the actual addresses of a recipient. To avoid out-of-band communications between senders and receivers for updating addresses, we leverage the notion of stealth address [CM17] to ensure that a recipient can only use the secret key of a permanent address to retrieve her messages.

Specifically, a recipient has a permanent address $A = g^a$ where a is the secret key. To send a message m to the recipient, the sender generates a new one-time address $E = g^{H(A^r)}$ together with $R = g^r$, where r is a randomness and H is a hash function, and generates shares L_1 and L_2 such that $L_1 \cdot L_2 = E$. Later, the two servers obtain L_1 and L_2 , respectively, as before. In addition, R can be publicly published on the board, so both servers can obtain R . When the recipient requests a retrieval, she generates random shares a_1 and a_2 where $a_1 + a_2 = a$ and sends a_i to Server_i (note that the two servers are non-colluding). Then, Server_2 sends R^{a_2} to Server_1 such that Server_1 can compute $s = H(R^{a_1} \cdot R^{a_2})$ and $E' = g^s$. It is easy to see that if $E = E'$, then $E'/L_1 = L_2$. Therefore, the two servers can use E'/L_1 and L_2 respectively, to perform the subsequent process as in HomeRun. In this way, as long as the recipient cannot provide valid a_1 and a_2 , Server_1 cannot compute the matched one-time address E' . Moreover, the address is one-time, so leaking it to Server_1 does not break the recipient’s privacy. Furthermore, if a malicious sender generates $R = g$ to leak the permanent address of a recipient to Server_1 , the malicious behavior can be easily detected. Therefore, the message can be prevented from being published on the board, or the honest server can reject processing the message.

6.2 Option to Not Delete

We provide a deletion feature that helps the servers to avoid continuously increasing storage consumption. However, the two servers can learn which messages are retrieved (and thus deleted) within a certain time frame, which may raise concerns about privacy for the recipients. In order to accommodate the interests of the recipients, in practice, we can allow them the option not to delete their pertinent messages. More specifically, the recipient can inform the two servers if she agrees to deletion, and the two servers reach a consensus on the recipient’s decision. If the recipient chooses not to delete, the two servers will set all the “delete labels” (see Section 2.3) as 0, such that this retrieval will not affect the OPRF values.

6.3 Timing Issues with Deletion

Recall that in our protocol the recipients retrieve their messages (via PIR) after first obtaining the relevant indices. If, for any reason (such as network instability, or messages being stored at different servers), the messages are deleted before the receiver retrieves them, it could result in permanent loss of the pertinent messages. Thus the process of deletion should be tied to the retrieval of messages itself, rather than relying on the index retrieval. In our work, we leverage DPF-based PIR [KOR19, BGI16] to achieve message retrieval. For a request to retrieve messages indexed by i , the two servers performing DPF-based PIR will generate two string vectors, \vec{V}_1 and \vec{V}_2 , respectively; The elements of the two vectors are identical elsewhere, except at the i -th position. It can be easily observed that the elements in the two vectors share identical features with the “delete labels” mentioned in Section 2.3. Therefore, the two servers can follow the same idea described in Section 2.3 to identify the messages that are indeed retrieved in an interval, by using the vectors.

¹²The malicious behaviors that compromise correctness can be detected, as all the messages are publicly available on the blockchain. All the previous works do not consider the adversary that aims to break correctness.

6.4 Horizontal Scaling

To further improve performance, we can use multiple pairs of servers to horizontally scale. Specifically, we can allocate each pair of servers to serve a designated set of recipients. To send a message to a recipient, the sender leverages the public keys of the corresponding servers to encrypt the auxiliary information. Then, only the designated servers can open the auxiliary information and assist the recipient with retrieval. But note that anyone observing the bulletin board can see which messages correspond to which server, thereby reducing the anonymity set of the recipient of that message to the parties that are serviced by the two servers. We can mitigate this by using key-private encryption so that one cannot tell the set of servers that can process this ciphertext, thereby achieving full anonymity for that recipient.

6.5 Group Setting

Next, we explore a potential extension of our protocol - group retrieval. Recently, Liu et al. [LTW23] extended the topic to a group setting where a message may be addressed to more than one recipient with two cases: (1) Ad-hoc Group where a group is chosen by the sender; (2) Fixed Group where a group is pre-formed by its members. The main difference between the two cases is that for a fixed group, the sender can treat it as a single entity, while for an ad-hoc group, the sender still needs to treat each member individually. Here, we discuss how to extend our protocol to support the two group settings.

(1) Ad-hoc Group. The key idea is to allow each server to recover the label for each member, and then leverage the label to assist the member with retrieval as before. We assume that a sender chooses k recipients with public keys $\{pk_1, \dots, pk_k\}$ to form a group, and randomly splits each public key pk_i to generate the labels $\{L_i^1, L_i^2\}$ as before. Then, the sender chooses a unique identifier id_i for each member, and interpolates a polynomial f_1 (resp. f_2) over points $\{(id_i, L_i^1)\}_{i \in [k]}$ (resp. $\{(id_i, L_i^2)\}_{i \in [k]}$). Last, the sender sends polynomials f_1 and f_2 to $Server_1$ and $Server_2$, respectively. When requesting retrieval, the j -th member still splits pk_j as before and sends the labels and id_j to the servers. The servers use id_j to recover the labels and perform the subsequent procedure as before.

(2) Fixed Group. The key idea is to enable all members to collaboratively perform the role of the recipient in our original protocol. We set the public key of the group as $pk_G = \prod_{i=1}^m pk_i$ and the corresponding secret key as $sk_G = \prod_{i=1}^m sk_i$. The sender splits the group public key to labels L_G^1 and L_G^2 and sends them to the servers, respectively. To retrieve messages, each member in this group splits his secret key sk_i into sk_i^1 and sk_i^2 and generates the corresponding $pk_i^1 = g^{sk_i^1}$ and $pk_i^2 = g^{sk_i^2}$. Then, the members collaboratively generate two multi-signature σ_1 and σ_2 under public lists $\{pk_i^1\}_{i \in [k]}$ and $\{pk_i^2\}_{i \in [k]}$. Last, the leader of the group sends $\langle \sigma_1, \{pk_i^1\}_{i \in [k]}, \prod_{i=1}^k pk_i^1 \rangle$ and $\langle \sigma_2, \{pk_i^2\}_{i \in [k]}, \prod_{i=1}^k pk_i^2 \rangle$ to the two servers, respectively. After verifying the multi-signatures successfully, the two servers can use $\prod_{i=1}^k pk_i^1$ and $\prod_{i=1}^k pk_i^2$ to generate test labels and perform the subsequent as before. A limitation of this extension is that members within a group cannot retrieve messages independently. We defer this issue to future work.

6.6 Denial-of-Service Attack

The previous works [MSS⁺22, LT22] limit the number of pertinent messages of a recipient, so it is inevitable that an adversary can send many messages to a recipient to induce an overflow, as mentioned in [LT22]. Our HomeRun and [BLMG21, JLM23] eliminate this limitation, thus avoiding this DoS attack. In addition, Liu et al. [LT22] pointed out an amplified DoS attack: the adversary maliciously crafts labels such that the corresponding messages are misperceived as pertinent by almost all recipients. Liu et al. [LT22] proved that their schemes are resistant to this DoS attack, but the solution by [BLMG21] is not. HomeRun essentially uses the address of a recipient to detect her pertinent messages, and the addresses of honest recipients are mutually distinct. Therefore, the amplified DoS attack cannot be applied to HomeRun. The existing works, including HomeRun, do not consider the DoS attacks that use arbitrarily chosen addresses and/or spam messages to consume resources. But we note that senders could prove that the recipient of a transaction

belongs to the set of all parties to mitigate this attacks. This can be efficiently accomplished using any proof-of-membership techniques (Merkle Trees[Mer87], vector commitments[CF13] etc.)

7 Related Work

Hearn et al. [HC12] require a recipient to insert her addresses into a Bloom filter, which can be used to check if an element is in a set with a predetermined false-positive probability p . Then, the Bloom filter will be sent to a server that later checks if an address appearing in the bulletin board is in the Bloom filter; if so, the server sends the corresponding message to the recipient. The drawback here is the server can identify the messages pertinent to the recipient with the false-positive probability p . Moreover, according to the research by Gervais et al. [GCKG14], this solution also leaks considerable information about addresses.

Later, Beck et al. [BLMG21] followed the idea of introducing false-positive probability while avoiding the leakage of addresses. Moreover, they first formalized the problem as “Fuzzy Message Detection (FMD)” where “fuzzy” means that the detection is with a false-positive probability. In their work, two schemes, FMD1 and FMD2, are proposed. FMD1 can only support restricted false-positive probabilities p of the form $\frac{1}{2^7}$, whereas FMD2 can support more fine-grained false-positive probabilities p of the form $\frac{m}{2^7}$. Most recently, a concurrent work by Pu et al. [PTDH23] also followed a similar idea by designing post-quantum fuzzy stealth signatures, but achieved better performance, which is comparable to ours. In these schemes, the recipient still needs to pick up her pertinent messages from the messages received from the server, which is avoided in our protocol. Moreover, our protocol can achieve full privacy. Note that the schemes in [BLMG21, PTDH23] have to set $p = 1$ to achieve full privacy, but the performance would be almost equivalent to that of directly scanning the data on the chain.

Liu et al. [LT22] achieved full privacy more efficiently, and the recipients in their schemes can obtain pertinent messages without re-testing by themselves. Nevertheless, due to the inefficiency of fully homomorphic encryption, their schemes still lack efficiency when a large number of messages are published on the bulletin board.

Madathil et al. [MSS+22] only focused on index retrieval without considering message retrieval. Their key idea is to require the server to maintain a table, wherein each row corresponds to a recipient and records the pertinent indexes for the recipient. When a recipient requests an index retrieval, the server responds with the corresponding row. To protect privacy, they proposed to use TEE or a two-party garbled circuit to maintain the table, and update the entire table upon the addition of a new index; only the TEE-based scheme is efficient. Most recently, Jakkamsetti et al. [JLM23] also presented a TEE-based solution. To avoid limiting the number of pertinent messages for a recipient, Jakkamsetti et al. [JLM23] changed the table used in [MSS+22] to a linked list. Moreover, by using ORAM, they achieved sublinear cost. Our protocol does not rely on TEE but achieves a performance comparable to that of [MSS+22], and the same privacy as that of [JLM23].

8 Conclusion

In this work, we design an Oblivious Message Retrieval protocol, HomeRun, leveraging two non-colluding, semi-honest servers. Through our protocol, recipients can efficiently retrieve messages from a bulletin board without compromising privacy. The comprehensive experimental results demonstrate the high efficiency of our protocol. Particularly, assuming there are 2^{19} unretrieved messages on the bulletin board, our protocol allows a recipient to retrieve a message within 1 second using 16 threads in a WAN setting.

Compared with previous works, our protocol not only offers performance advantages but also presents significant improvements in both functionality and security. Specifically, as for functionality, our protocol supports unlimited pertinent messages for a recipient, periodic deletion, and appending-based addition. Regarding security, our protocol achieves full privacy and guarantees request unlinkability.

Acknowledgements

We thank Zhongtang Luo for his assistance in running the experiments. This work was supported in part by the National Science Foundation (NSF) under grant CNS1846316, and by Protocol Labs and Supra Research.

References

- [APY20] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder - scalable, robust anonymous committed broadcast. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1233–1252. ACM Press, November 2020.
- [BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.
- [bit] Bitcoin blockchain size. https://ycharts.com/indicators/bitcoin_blockchain_size.
- [BLMG21] Gabrielle Beck, Julia Len, Ian Miers, and Matthew Green. Fuzzy message detection. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1507–1528. ACM Press, November 2021.
- [CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16*, pages 55–72. Springer, 2013.
- [CGS22] Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-PSI with linear complexity via relaxed batch OPRF. *PoPETs*, 2022(1):353–372, January 2022.
- [CM17] Nicolas T Courtois and Rebekah Mercer. Stealth address and key management techniques in blockchain systems. In *ICISSP 2017-Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, pages 559–566, 2017.
- [Cou18] Geoffroy Couteau. New protocols for secure equality test and comparison. In *International Conference on Applied Cryptography and Network Security*, pages 303–320. Springer, 2018.
- [DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS 2015*. The Internet Society, February 2015.
- [eth] Ethereum chain full sync data size. https://ycharts.com/indicators/ethereum_chain_full_sync_data_size.
- [GCKG14] Arthur Gervais, Srdjan Capkun, Ghassan O. Karame, and Damian Gruber. On the privacy provisions of bloom filters in lightweight bitcoin clients. In *Proceedings of the 30th Annual Computer Security Applications Conference, ACSAC '14*, page 326–335, New York, NY, USA, 2014. Association for Computing Machinery.
- [HC12] Mike Hearn and Matt Corallo. Connection bloom filtering. <https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki>, 2012.
- [HHCG⁺23] Alexandra Henzinger, Matthew M Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In *Usenix Security*, volume 23, 2023.

- [HHK18] Ryan Henry, Amir Herzberg, and Aniket Kate. Blockchain access privacy: Challenges and directions. *IEEE Security & Privacy*, 16(4):38–45, 2018.
- [JLM23] Sashidhar Jakkamsetti, Zeyu Liu, and Varun Madathil. Scalable private signaling. Cryptology ePrint Archive, Paper 2023/572, 2023. <https://eprint.iacr.org/2023/572>.
- [KOR19] Daniel Kales, Olamide Omolola, and Sebastian Ramacher. Revisiting user privacy for certificate transparency. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 432–447. IEEE, 2019.
- [KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 705–734. Springer, Heidelberg, May 2016.
- [LK23] Donghang Lu and Aniket Kate. RPM: robust anonymity at scale. *Proc. Priv. Enhancing Technol.*, 2023(2):347–360, 2023.
- [LSD23] Simon Langowski, Sacha Servan-Schreiber, and Srinivas Devadas. Trellis: Robust and scalable metadata-private anonymous broadcast. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023.
- [LT22] Zeyu Liu and Eran Tromer. Oblivious message retrieval. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 753–783. Springer, Heidelberg, August 2022.
- [LTW23] Zeyu Liu, Eran Tromer, and Yunhao Wang. Group oblivious message retrieval. Cryptology ePrint Archive, Paper 2023/534, 2023. <https://eprint.iacr.org/2023/534>.
- [Mer87] Ralph C. Merkle. A digital signature based on a conventional encryption function. pages 369–378, 1987.
- [MSS+22] Varun Madathil, Alessandra Scafuro, István András Seres, Omer Shlomovits, and Denis Varlakov. Private signaling. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 3309–3326. USENIX Association, August 2022.
- [NSD22] Zachary Newman, Sacha Servan-Schreiber, and Srinivas Devadas. Spectrum: High-bandwidth anonymous broadcast. In Amar Phanishayee and Vyas Sekar, editors, *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, pages 229–248. USENIX Association, 2022.
- [Ope] Openssl. <https://www.openssl.org/source/gitrepo.html>.
- [PTDH23] Sihang Pu, Sri AravindaKrishnan Thyagarajan, Nico Döttling, and Lucjan Hanzlik. Post quantum fuzzy stealth signatures and applications. Cryptology ePrint Archive, Paper 2023/1148, 2023. <https://eprint.iacr.org/2023/1148>.
- [RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 901–930. Springer, Heidelberg, October 2021.
- [Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO’89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.
- [SG23] Sajin Sasy and Ian Goldberg. Sok: Metadata-protecting communication systems. Cryptology ePrint Archive, Paper 2023/313, 2023. <https://eprint.iacr.org/2023/313>.
- [Tor] The Tor Project. <https://metrics.torproject.org/>.