sPAR: (Somewhat) Practical Anonymous Router

Debajyoti Das
debajyoti.das@eit.lth.se
Lund University
Lund, Sweden

Jeongeun Park
jeongeun.park@ntnu.no
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway

Hyewon Sung
hyewonsung@ewha.ac.kr
Ewha Womans University
Seoul, South Korea

Abstract

Anonymous communication is one of the fundamental tools to achieve privacy for communication over the internet. Almost all existing design strategies (e.g., onion routing/Tor, mixnets) for anonymous communication rely on the existence of some honest server/router in the network infrastructure to provide anonymity. A seminal work by Shi and Wu (Eurocrypt 2021) proposes the first cryptographic design for a *non-interactive anonymous router* (NIAR) that can use a single untrusted server or router to permute a set of messages without revealing the permutation to the untrusted router. This work is a really important step towards showing the possibility of designing such protocol from standard cryptographic assumptions. However, the existing constructions are only of theoretical nature and still leaves many open questions towards realizing such systems in practice: (1) the cryptographic building blocks used in those designs are really difficult to implement in practice. (2) Their setup phase takes the permutation as an input to generate the encryption/decryption keys; which means that the messages from the same sender in different *rounds* will be at the same position in the output vector, unless the setup phase is run before every round with a new permutation. (3) It is not known how to realize such a setup procedure, that initializes a random permutation obliviously, without any trusted entities in the system.

In this paper, we propose the first (somewhat) practical design, which we call *sPAR*, that solves the above problems. Our design also relies on a one-time setup phase, however the setup phase does not take any specific permutation as input. Instead, our design can reuse the same setup for many rounds and generates a fresh permutation for every round based on the random values locally generated by the clients. Our design is implementable and deployable in practice; and this presents a new direction for designing anonymous communication systems. Unlike some existing systems like Tor, sPAR does not scale to millions of users, however, we demonstrate with a proof-of-concept implementation that sPAR supports around one hundred users, and show that an optimized variant of our algorithm reduces the latency to a few seconds per message.

1 Introduction

Many anonymous communication systems [1, 11, 32, 34, 72] have been proposed in the last four decades with the goal to protect the privacy for users over the internet. Most design techniques rely on the existence of some honest parties in their infrastructure towards achieving their privacy goal. For instance, popular design techniques such as mixnets [44–47, 64] or Tor [32] route the messages through many servers before delivering them to the intended recipients; and a message needs to go through at least one honest server on its path to be 'mixed' with other messages.

There exist designs based on multi-party-computation approaches where a sender secret-shares their message with the servers, and the servers collaboratively 'shuffle' the received messages without any of the individual servers knowing the permutation [24, 33, 34, 39, 40, 56]. These designs require at least one of the MPC-servers to be honest in order to achieve anonymity.

A recent work by Shi and Wu [69] is the first work to explore the possibility of designing anonymous communication systems without requiring any honest parties/components in the infrastructure. The constructions presented in that work realize *function-hiding* Multi-Client Functional Encryption (MCFE) from a regular MCFE for *selection* function, and use that (in conjunction with correlated pseudorandom functions) to achieve *permutation* on an untrusted server or router, without the server learning the permutation. They call their design paradigm *Non-interactive Anonymous Router* or NIAR. In order to instantiate the function-hiding MCFE, their setup phase takes the *permutation* as an input, and then many rounds of shuffles can be executed using the same setup. We discuss the nuances of their solutions in Section 2.2 in detail.

Although their work takes a major fundamental step towards showing the theoretical possibility of designing such systems, it still leaves many questions towards realizing them in practice unanswered:

(1) How to generate the permutations that the setup phase in NIAR takes as input, without any trusted component? And, how to realize the setup phase in practice so that the permutation is not leaked to anyone?
(2) Is it possible to design an *oblivious permutation* algorithm with cryptographic building blocks that are easier to implement compared to function-hiding MCFE and correlated pseudorandom functions?
(3) How practical can such designs be for real-world application scenarios?

Overall, it still remains unknown if it is possible to provide a practical design for *anonymous routers*.

1.1 Our Construction

We take a positive step towards answering the above questions by designing an anonymous router based on homomorphic encryption techniques. We do not only show how to generate the permutation without any trusted setup, but also completely disassociate it from the setup phase. In our design, each round of message-shuffling uses a fresh permutation based on the random values locally generated by the users.

We use the following strategy to generate fresh permutations in every round: the clients generate random values locally, and send those (encrypted) random values and the encrypted messages to

the untrusted server. The server places the message in a large array, and the position is chosen based on the random values, without the server knowing the values. To avoid collisions without requiring a very large array, we use the trick of choice-of-three from balls-and-bin problem [57]: a client chooses three random indices, and the message is placed in the first available index/bin. We design an oblivious algorithm under fully homomorphic encryption (FHE) to enable placing a message with the choice-of-three trick. This choice-of-three brings down the failure probability (where one message could overwrite another) to 10^{-11}.[1]

As a tradeoff, our protocol requires one additional round of communication involving the clients for the decryption of the output messages; still, the clients communicate only with the server, and do not need to communicate among each other. This is a deviation from the single-round communication model in NIAR. Since our design strategy is completely different from NIAR or any other existing anonymous communication systems, it presents *a new avenue* for designing such systems.

Already existing practical instantiations of FHE schemes [8, 9, 16, 19, 35] allow us to instantiate and implement our design. We provide a proof-of-concept implementation [2] for our protocol, and benchmark the performance for up to 100 clients. For 100 clients in the system, the expected latency per message is only several seconds, which shows promise for applicability in small-scale environments.

Our key contributions can be summarized as:

- a new algorithm to achieve *oblivious shuffle* in the FHE setting based on balls-and-bins technique;
- based on the above algorithm, an instantiation of *anonymous router* that can be realized in practice;
- a proof-of-concept implementation and performance benchmarks;
- overall, a new design paradigm for anonymous communication systems.

1.1.1 Application Scenarios And Limitations. We refer to existing works such as [10, 69] for an extensive discussion on general application scenarios for anonymous routers. Specific to our design, our experimental evaluations up to 100 users demonstrate that sPAR can be used in practice for small-scale applications like mixing transactions in Cryptocurrencies (most existing solutions [7, 37, 67] in that space support only up to a few hundred users), peer-to-peer messaging in a LAN network, etc. We want to highlight that this work presents a first design of its kind, and leaves a lot of scope for future improvements in terms of performance as well as scalability. Additionally, our current work does not consider many possible issues related to real-world deployment scenarios: (i) clients could arbitrarily drop from the systems; (ii) different clients could send (or receive) messages at different rates in practice; (iii) a wide array of active and DoS attacks; and many more. We discuss some of those possible avenues and future works in Section 6.

[1] A failure in a round does not break anonymity, rather forces the users to run the round again with new random choices.

[2] The source code implementation can be found at the GitHub repository linked in Section 5.

2 Technical Overview

2.1 Notations and System Model

We consider a system with a set of senders $\mathcal{S}$ that want to send messages to a set of recipients $\mathcal{R}$ via an intermediate router/server P. Each sender is denoted by u_i where $i \in \{1, \ldots, \eta\}$ and $|\mathcal{S}| = \eta$. Similarly, each recipient is denoted by r_i where $i \in \{1, \ldots, \eta'\}$ and $|\mathcal{R}| = \eta'$. We summarize all the system parameters in Figure 1.

η	Total number of senders
$\mathcal{S}$	Set of all senders u_i, $\forall i \in \{1, \ldots, \eta\}$
$\mathcal{R}$	Set of all recipients r_i, $\forall i \in \{1, \ldots, \eta'\}$
P	untrusted router
δ	the adversarial advantage
$\xleftarrow{\$} [b, c]$	Draw uniformly at random from $[b, c]$

Figure 1: Protocol and system parameters for sPAR

We consider that our protocol works in *rounds*: in every round, the server/router collects all the incoming messages, runs the required computations as well as the decryption mechanism, and then forwards the messages to the intended recipients.

In the real world, the set of senders and the set of recipients could be overlapping or disjoint. For ease of exposition, we assume that all the *clients* act as a sender as well as a recipient in our system, and each client sends exactly one message in every round. This setting is also similar to that of NIAR [69] and many other designs for anonymous communication [5, 25, 28, 36, 46, 75]. However, our design is not limited by this restriction, and can easily be extended to the scenario of disjoint or partially overlapping sets of senders and receivers.

Similar to NIAR, we consider the setting where the clients do not communicate with each other directly after the one-time setup: all the communications go through the router/server. Our system model is illustrated in Figure 2.

Threat Model And Security Goals. We consider an honest-but-curious, probabilistic polynomial-time adversary that can corrupt the server as well as a subset of the clients. We consider that the corrupted clients or the server do not attempt to DoS the system.

Our core design achieves *sender anonymity*, i.e., even a corrupted recipient colluding with the server/router cannot determine the sender of a message. In this setting, since we do not consider the privacy of the recipients, the server retrieves all the (shuffled) messages in plaintext and then delivers them to the intended recipients. We consider that all recipients can be corrupted, the server can be corrupted, and up to $\eta - 2$ senders can be corrupted. All corrupted parties can collude with each other.

We provide further addendum to the protocol (c.f. Section 6.2) for additionally achieving *recipient anonymity*, i.e., even a compromised sender colluding with the server cannot determine the actual identity of the recipient for a message. For recipient anonymity, we consider that all senders can be corrupted, the server can be corrupted, and up to $\eta - 2$ recipients can be corrupted. All corrupted parties can collude with each other.

Note that the goals for sender anonymity and recipient anonymity conflict with each other in the trust assumption. Such a conflict

merely models the possible distrust between senders and recipients in real-world scenarios. We present the formal anonymity definitions in Section 2.5.

2.2 Existing Works And Their Limitations

Most design strategies for anonymous communication systems (e.g., Tor [32], mixnets [31, 45, 46], MPC-based designs [33, 34, 39, 40]) explicitly require at least one honest party in their infrastructure. We skip a detailed discussion about them and refer to some excellent existing works [27, 31, 68, 70] that provide surveys on such systems. Here we focus our discussion on designs that attempt to provide anonymity without additional trusted parties in the system.

Non-interactive Anonymous Router or NIAR. We mainly compare our design with the work of NIAR [69] by Shi and Wu. This was a breakthrough work to show the theoretical possibility of designing *anonymous routers* with non-interactive designs. Their design goals include both sender anonymity and recipient anonymity. This work provides a very crucial insight: such protocols need to realize *oblivious permutation* function on the untrusted server.

However, there seem to be many obstacles towards making their design techniques suitable for practical use: (1) The instantiations of the cryptographic building blocks (function-hiding MCFE and correlated pseudorandom function) are extremely expensive, if at all implementable. (2) The version of their protocol that additionally protects the anonymity for recipients (they call this property *full-insider protection*) requires sub-exponentially secure indistinguishability obfuscation — the practicality of such assumptions is unclear. (3) Additionally, their setup phase takes the *permutation* as an input, and then many rounds of shuffles can be executed using the same setup. All the messages from the same sender in between setup-phases are linkable to each other, even though the server does not know the permutation, unless the setup phase is run before every round with a new permutation. This creates a tradeoff between the *non-interactive* property and *strong* anonymity guarantee.

Our design sPAR relaxes the requirement of being *non-interactive*: it requires only two rounds of client-server communication, and no other communication among the clients. In more detail, one round for sending messages to the router, and the other one for decrypting the received messages. The additional round for decryption follows from our use of FHE with multi-party setup (very similar to [77]), which requires a collaborative decryption step. sPAR relies on the LWE/RLWE assumption and the cryptographic security of hash functions, is easily implementable, and can be suitable for applications with up to a hundred users (e.g., mixing transactions on blockchain, private messaging in a LAN network).

Oblivious Message Retrieval (OMR). OMR [6, 40, 51] is related to the anonymity of the recipient: it hides what messages have been retrieved by a user, and could be used to design systems that can protect the anonymity of recipients [39], but not of senders. There are some recent works in this space that can achieve OMR even with a single server [10, 51, 52]. With OMR designs, each message is encrypted with the encryption key for the intended recipient, and requires the recipient to send a *detection key* before every retrieval operation. And therefore, such design strategies cannot be extended to execute explicit shuffles among the messages (since they are encrypted under different keys), and consequently, are unsuitable to achieve sender anonymity (hiding the sender of a message even from a compromised recipient). Achieving sender anonymity seems to be an inherently difficult problem with the techniques from OMR designs, since the secret-key holder can decrypt all messages.

DC-nets. Dining cryptographers networks or DC-nets are very closely related to our design in terms of scalability and application scenarios. Moreover, peer-to-peer DC-net designs [12, 25, 36, 67] do not require any additional trusted entity in the network, and can achieve strong sender anonymity. Some DC-net designs can even scale for up to a few hundred users. However, their communication model is very different from our scenario: they require multiple rounds of communication and active interaction among all clients to run the protocol.

Other DC-net designs utilize a separate set of MPC servers [1, 5, 24, 26, 28, 34, 53, 75] to avoid the communications among the clients. The communications among the clients are now replaced by communication between the set of servers and the clients.

In essence, our design of sPAR improves upon peer-to-peer DC-nets by replacing all inter-client communication with just two communication rounds between clients and the (untrusted) server.

Designs Based on Trusted Execution Environments (TEEs). There are some recent designs that utilize Trusted hardware (e.g., Intex SGX, RISC-V Keystone, etc.) to design anonymous communication systems [56]. An untrusted server/router uses the trusted hardware component to shuffle the messages without being able to read the messages or being able to decipher the permutation. However, this requires a trust assumption on the hardware component itself, which is known to fail in many cases [42, 50].

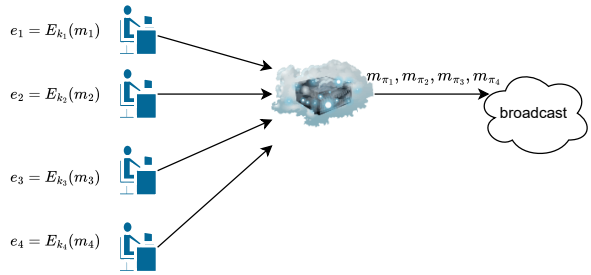
2.3 Challenges And Key Ideas

Setup for Multiple Clients. At the core of our design is the use of FHE in a multi-party setting, which enables multiple clients to compute a common function over their inputs without revealing them, using a jointly generated public key. Importantly, the corresponding secret key is not known to any single party.

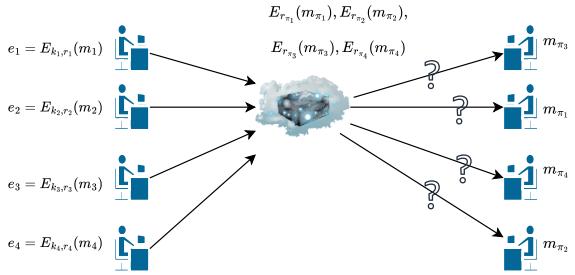
The naive use of FHE in anonymous routing would require giving the clients access to a common decryption key, which would allow a corrupted client to decrypt other clients' ciphertexts and thereby compromise anonymity [69]. We avoid this problem by adopting FHE with a collaborative key-generation setup, where the set of clients jointly generates a public key while keeping the secret key distributed. This ensures input privacy without exposing the global secret key. As a trade-off, decryption must be performed collaboratively, requiring one round of interaction among the clients.

The main advantage of our approach is minimal interaction with the server: the server can perform oblivious shuffling entirely in the encrypted domain without interacting with clients or relying on additional trusted parties (one or many trusted servers). While collaborative decryption introduces a single round of client interaction, the server-side computation remains fully non-interactive.

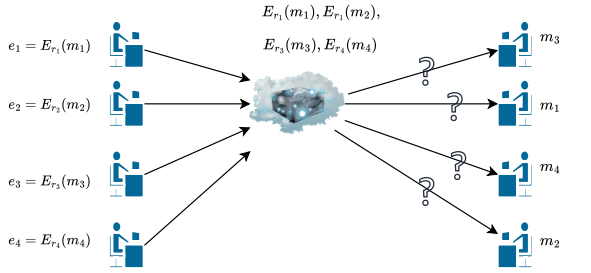
Such a multi-party variant of FHE can be instantiated in two ways. One approach is to use an MPC-based setup which is already deployed in a FHE based blockchain application [76, 77], in which a fixed set of clients jointly generate a public key corresponding to



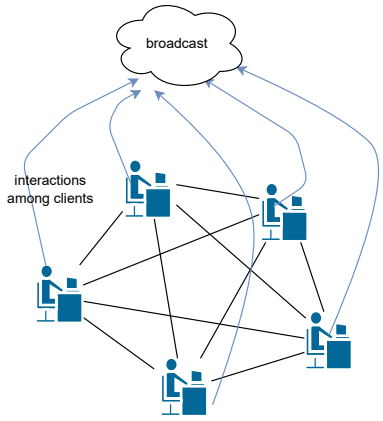
(a) An overview of an anonymous router in the anonymous broadcast setting: each user wants to broadcast her message via an untrusted router/server, without any interactions with other users. They send their encrypted messages to the server, and the server can decrypt the messages after it has run the *shuffle*. Additionally, the permutation π is unknown to the server. This scenario does not consider anonymity for the recipients



(b) An overview of an anonymous router when the system additionally aims to protect the anonymity of the recipients (*full insider protection*, as the NIAR [69] calls it). Here each user encrypts their message with a hybrid key derived from their own key and the public key of the intended recipient. The final result is the encryption of individual messages (in a shuffled order) encrypted under the public keys of the respective recipients, where the shuffle itself is unknown to the server.



(c) An overview of OMR functionality. Similar to anonymous routers, there is only one untrusted server. Each receiver is associated with an address/public-key, and that public-key should not leak the receiver's real identity. A sender wants to send a message m_i to the receiver associated with address r_i. The sender sends m_i in an encrypted format to the server; the server can deliver the message m_i to the receiver without knowing the association between the address and the physical identity of the receiver. The senders and the recipients do not trust each other, and all communications happen through the server/router.



(d) An overview of DC-net functionality in the P2P setting. In the original DC-net [12, 36], Dissent [25], Dicemix [67], and similar protocols, clients interact with each other over point-to-point links to achieve anonymous broadcast among themselves.

Figure 2: Illustrative examples for different design approaches for anonymous communication systems without any trusted entities in the network.

a globally shared secret key. This approach preserves the computational characteristics of the standard single-key FHE setting.

Alternatively, one can adopt a multi-party FHE construction similar to [61], where each client independently generates its own local secret key share and publishes a corresponding public-key distribution in a single round. However, such schemes often incur higher computational overhead in practice. In particular, ciphertext noise grows with the number of participating clients, which limits the depth of supported computations. Moreover, certain operations, such as gate operations [62], become significantly less efficient compared to the standard single-key setting.

For these reasons, we adopt the MPC-based setup to retain the efficiency of single-key FHE while supporting multiple clients in our implementation to demonstrate a practical running time.

Instantiation of Oblivious Permutation. One key technical problem we solve in our work is designing an oblivious shuffling protocol under FHE setting. In our *oblivious shuffle* algorithm, we generate fresh permutations in every round, unlike NIAR that fixes the permutation as part of the setup. Towards generating fresh permutations in every round, we use the following strategy: the clients generate random values locally, independent of all other clients and any past communications; the server can derive the permutation (obliviously) by using those encrypted random values received from the clients. More specifically, let us assume that the server holds an empty database with $\mathfrak{N}$ slots, where $\mathfrak{N} >> \eta$. A client with a message m can choose a random index $i \in \{1, \ldots, \mathfrak{N}\}$ and send encrypted m and i to the server. The server can place m at the position i without knowing either of the values with Homomorphic Placing [21]. After all the clients have placed their messages

the same way, if i values are chosen uniformly at random by the clients, the ordering among the messages in the database (excluding the empty slots) would be a random permutation (assuming there is no collision). Then the server can launch the decryption phase to decrypt the messages.

However, unless $\mathfrak{N} \in \omega(\eta^2)$, there is a high probability of collision, which results in a *failure* in the message delivery. Large $\mathfrak{N}$ would introduce a very high computation overhead for the server. To solve the collision problem without introducing large overhead, we use the trick of choice-of-three from balls-and-bins problem [57]: a client chooses three random indices, and the message is placed in the first available index/bin. We build an oblivious algorithm to enable choice-of-three, placing one of the choices in the least-filled bin. This choice-of-three brings down the failure probability to 10^{-11} with only $\mathfrak{N} = 3\eta$. However, this is an imperfect solution: this allows the adversary a small probability of finding the partial order between two messages in the same bin. To solve that problem, we sort each bin individually. Sorting each bin (with only three elements) does not add significant overhead. These techniques allow us to provide a simple yet effective design for an anonymous router that is practical for several small-scale applications. However, instantiation of FHE in the multi-party setting comes with the requirement of a partial decryption step, and that adds one additional round of communication between the server and the clients. However, the clients still do not need to interact with each other during the protocol run.

2.4 Protocol Overview

Our protocol has one setup phase, and then the server can run polynomially many number of *rounds* using the same setup.

Setup Phase. As part of the setup phase, the clients jointly execute an MPC-based key-generation procedure to generate a common master secret key s without revealing it to any individual client. Instead, each client obtains only a secret share s_i of s. Using these shares, the clients jointly derive the collective encryption key (the common public key) $\hat{\mathsf{pk}}$ together with the required evaluation keys, and send the resulting public material to the server. Any message m encrypted under the common public key as $c = \mathsf{Enc}_{\hat{\mathsf{pk}}}(m)$ can only be decrypted if all clients participate by providing *partial decryptions* using their corresponding secret shares.

Online Phase: Protocol Round. Similar to the setting of NIAR [69], we consider batch-based message processing: each sender sends exactly one message during a protocol *round*. After collecting messages from all the clients, the server starts the decryption procedure for that round. In case the client does not have a message to send, the client can send a random message.

At the beginning of a round, the server maintains η buckets, each bucket containing three slots, for a total of 3η slots. The client u_j with a message m_j picks three indices a_1, a_2, a_3, each from $\{1, \ldots, \eta\}$ uniformly at random and independent of each other. Then, the client sends encrypted m_j, a_1, a_2, a_3 to the server. The server then homomorphically places the message in the least filled bucket among the buckets corresponding to a_1, a_2, a_3. We call this procedure homomorphic placing, denoted by HomPlacing, and described it in Algorithm 1. Because the clients freshly sample these candidates in every round, the effective placement permutation is refreshed from round to round.

After the server has received and placed messages from all clients, it sorts each bucket individually and initiates the decryption step by sending the encrypted output vector to the clients for partial decryption. After receiving the partial decryptions from all clients, the server aggregates them and recovers the messages. The partial decryption per each client is a vector of $O(\eta)$ ciphertext.

Note that, without the sorting step, there is a small leakage to the adversary. Messages placed in the same bucket would preserve their partial order, allowing the server to distinguish the senders of some message pairs with non-negligible probability. Since each bucket contains only three slots, sorting incurs only a small overhead and is highly parallelizable. In Section 4.4, we show that this protocol provides sender anonymity with negligible adversarial advantage.

Packing Optimization. In order to reduce the communication cost, we can make use of a well-known packing method of FHE ciphertexts. The method packs several individual ciphertexts into one ciphertext by using its underlying algebraic structure in order to reduce communication complexity and cost. Such optimization will be discussed in Section 4.3.

Recipient Anonymity. We also show in Section 6.2 how to extend our design to additionally provide anonymity for the recipients. We mainly utilize existing OMR designs in conjunction with our oblivious shuffling technique.

The recipient anonymity property we consider (c.f. Section 6.2 and Definition 2) is similar to the flavor of onion services (formerly 'hidden services') in Tor [38]. The sender might know a *public-identity* (e.g., onion address in the context of Tor, or a public-key) for the intended recipient, however, does not know the actual/physical identity r_j of the recipient hidden behind that public identity. Recipient anonymity in conjunction with sender anonymity is analogous to the *full-insider protection* notion used in NIAR. The details are explained in Section 6.2.

2.5 Security Definitions

Our security goals are similar to that of NIAR [69]. However, instead of using their definitions, we use indistinguishability-based definitions of *sender anonymity* and *recipient anonymity* that are close to and borrowed from a long line of literature on anonymous communication systems [13, 28, 31, 41, 44–46] and their formalization [4, 29, 30, 43]. This allows us to be comparable with other design paradigms of anonymous communication in terms of anonymity guarantees.

The choice is also because of a weakness in the definitions of NIAR [69]: if a mixing/anonymity protocol uses the same permutation for multiple rounds, a compromised recipient will know that she is receiving all the messages from the same sender, even though she does not know who that sender is. And that leakage might help the adversary build a strong profile about the sender. The definition considered by Shi and Wu does not consider that leakage, and only considers the leakage about the permutation used in the protocol.

Our protocol achieves sender anonymity: which is analogous to the *receiver-insider protection* defined in the NIAR paper. Additionally, we extend our protocol to achieve recipient anonymity (even from corrupted senders colluding with the server) — this

is analogous to their *full-insider protection* notion. Moreover, the definitions we use account for the leakage mentioned above. We present the definitions below.

2.5.1 Sender Anonymity Game. The sender anonymity game, denoted by $\mathcal{G}_{\mathsf{SA}}^{\Pi,\mathcal{A},\eta}(1^\lambda)$, between a challenger Ch and an adversary $\mathcal{A}$ for protocol Π can be described as follows:

- The challenger Ch provides the adversary $\mathcal{A}$ with the description of Π.
- $\mathcal{A}$ statically corrupts all senders in $\mathcal{S}$ except from a pair u_0, u_1, all recipients, and the server P .
- Ch and $\mathcal{A}$ engage in an execution of Π where Ch acts on behalf of u_0 and u_1, while $\mathcal{A}$ controls the corrupted parties and monitors the network traffic as a global passive adversary.
- Before any round of choice the $\mathcal{A}$ can send a pair of challenge messages (u_0, r^*, m_0) and (u_1, r^*, m_1) to Ch. Here r^* is a corrupted recipient, where u is the sender of the message and m is the content.
- In turn, Ch chooses a random bit $b \in \{0, 1\}$ and initiates the challenge transmissions in that round according to the following cases:
  - If $b = 0$, Π transmits the messages (u_0, r^*, m_0) and (u_1, r^*, m_1).
  - If $b = 1$, Π transmits the messages (u_1, r^*, m_0) and (u_0, r^*, m_1).
- $\mathcal{A}$ can terminate the game any time by outputting a bit b^*, as a guess for the challenge bit b.
- The game returns 1 if and only if $b^* = b$ (i.e., $\mathcal{A}$ guesses correctly), otherwise the game returns 0.

Definition 1 (Sender Anonymity). *A protocol Π provides* sender anonymity *for messages up to probability δ for $0 \le \delta < 1$ if, for all probabilistic polynomial time (PPT) adversaries $\mathcal{A}$, and the security parameter λ the following holds:*

$$\left|\Pr\left[\mathcal{G}_{\mathsf{SA}}^{\Pi,\mathcal{A},\eta}(1^\lambda) = 1\right] - \Pr\left[\mathcal{G}_{\mathsf{SA}}^{\Pi,\mathcal{A},\eta}(1^\lambda) = 0\right]\right| \le \delta(\lambda).$$

For our purposes, we want $\delta()$ to be negligible in λ.

2.5.2 Recipient Anonymity Game. The recipient anonymity game $\mathcal{G}_{\mathsf{RA}}^{\Pi,\mathcal{A},\eta'}(1^\lambda)$ between a challenger Ch and an adversary $\mathcal{A}$ for protocol Π can be described as follows:

- The challenger Ch provides the adversary $\mathcal{A}$ with the description of Π.
- $\mathcal{A}$ statically corrupts all recipients in $\mathcal{R}$ except from a pair r_0, r_1, all senders, and the server P .
- Ch and $\mathcal{A}$ engage in an execution of Π where Ch acts on behalf of r_0 and r_1, while $\mathcal{A}$ controls the corrupted parties and monitors the network traffic as a global passive adversary.
- Before any round of choice the $\mathcal{A}$ can send a pair of challenge messages (u^*, r_0, m_0) and (u^*, r_1, m_1) to Ch where r is the intended recipient of the message and m the content.
- In turn, Ch chooses a random bit $b \in \{0, 1\}$ and initiates the challenge transmissions in that round according to the following cases:
  - If $b = 0$, Π transmits the messages (u^*, r_0, m_0) and (u^*, r_1, m_1).
  - If $b = 1$, Π transmits the messages (u^*, r_1, m_0) and (u^*, r_0, m_1).
- $\mathcal{A}$ can terminate the game any time by outputting a bit b^*, as a guess for the challenge bit b.
- The game returns 1 if and only if $b^* = b$ (i.e., $\mathcal{A}$ guesses correctly), otherwise the game returns 0.

Definition 2 (Recipient Anonymity). *A protocol Π provides* recipient anonymity *for messages up to probability δ for $0 \le \delta < 1$ if, for all probabilistic polynomial time (PPT) adversaries $\mathcal{A}$, and the security parameter λ the following holds:*

$$\left|\Pr\left[\mathcal{G}_{\mathsf{RA}}^{\Pi,\mathcal{A},\eta}(1^\lambda) = 1\right] - \Pr\left[\mathcal{G}_{\mathsf{RA}}^{\Pi,\mathcal{A},\eta}(1^\lambda) = 0\right]\right| \le \delta(\lambda).$$

3 Building Blocks

3.1 Preliminaries of Underlying FHE Scheme

Our protocol, introduced in the next section, can be instantiated with any IND-CPA secure FHE scheme that supports the operations presented in this section. However, for concrete instantiation, we chose TFHE[19] which is IND-CPA secure based on the (R)LWE problem [55, 65].

We recall the notion of IND-CPA security for public-key encryption schemes. Informally, an encryption scheme is IND-CPA secure if no probabilistic polynomial-time adversary can distinguish between encryptions of any two messages of its choice, given access to the public key.

This notion ensures that ciphertexts reveal no information about the underlying plaintexts beyond their length. In our protocol, this guarantees that the semi-honest server, which only observes encrypted messages and intermediate ciphertexts, cannot learn any information about the clients' inputs.

Notations. We denote λ as the security parameter of the FHE scheme. We define vectors and matrices in lowercase bold and uppercase bold, respectively. For a vector $\mathbf{x}$, both $\mathbf{x}_i$ and $\mathbf{x}[i]$ denote either the i-th component scalar or the i-th element of an ordered finite set. $\log(\cdot)$ is the logarithm function with base 2. Let R and R_q denote $\mathbb{Z}[X]/(X^N + 1)$ and $(\mathbb{Z}/q\mathbb{Z})[X]/(X^N + 1)$, respectively, for positive integers q and N. We summarize our notations in Table 1.

We introduce ring learning with error (RLWE) problem which we are relying on for the security of an FHE scheme that we need to instantiate our protocol.

Definition 3. *For security parameter λ, let $\Phi_m(X)$ be a cyclotomic polynomial with degree $\phi(m)$ and set $\mathcal{R} = \mathbb{Z}[X]/(\Phi_m(X))$. Let $q = q(\lambda) \ge 2$ be an integer. For a fixed secret random element $s \in \mathcal{R}_q$ and a distribution $\chi = \chi(\lambda)$ over $\mathcal{R}$, the decisional RLWE problem says the distribution outputting $(a, a \cdot s + e)$, where a and e is chosen uniformly at random from $\mathcal{R}_q$ and χ, respectively, and the distribution outputting (a, u) chosen uniformly at random from $\mathcal{R}_q^2$ are computationally indistinguishable.*

The sample $(a, a \cdot s + e) \in \mathcal{R}_q^2$ is called RLWE sample. An RLWE ciphertext of encrypting $m \in R_t$ is defined as $(a, a \cdot s + e + \Delta m)$, where Δ is an encoding factor typically defined as q/t.

For a simple and convenient use like many schemes, we set $\Phi(X) = X^N + 1$, where N is a power of 2.

We also consider the standard learning with errors (LWE) setting [66]. An LWE ciphertext encrypting a message $m \in \mathbb{Z}_q$ under a secret key $\mathbf{s} \in \mathbb{Z}_q^n$ is given by a pair $(\mathbf{a}, b) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where $b = \langle \mathbf{a}, \mathbf{s} \rangle + \Delta m + e$, for a uniformly random vector $\mathbf{a} \in \mathbb{Z}_q^n$ and an error term e sampled from a noise distribution.

Table 1: Parameters related to the FHE scheme

N	The maximum degree of a polynomial
t	Plaintext modulus
q	Ciphertext modulus
ℓ	Decomposition parameter
R	$\mathbb{Z}[X]/(X^N+1), N \in \mathbb{N}$
R_q	$\mathbb{Z}[X]/(X^N+1) \mod q, q, N \in \mathbb{N}$
λ	Security parameter

We note that an LWE ciphertext can be obtained from an RLWE ciphertext via the $\mathsf{SampleExtract}(\cdot)$ procedure[3], which extracts a coefficient and converts it into the LWE form.

Below is the definition of homomorphic operations that we need for our protocol. TFHE supports the efficient homomorphic operations that we define below.

3.1.1 Homomorphic Operations and Basic Algorithms. Our goal is to support a fixed set of multiple clients in an FHE setting without revealing each client's input or secret key to others. To achieve this, one may consider either multi-party FHE [3, 58] (a.k.a threshold multi-key FHE) or an MPC-based setup that distributes the secret key shares.

The former typically requires a simpler setup, involving only a small number of communication rounds and lightweight homomorphic operations (e.g., additions). However, it incurs a higher cost during the online phase due to expensive multi-party homomorphic evaluations (especially multi-party bootstrapping [62]). In contrast, the latter introduces a more complex setup phase with multiple rounds of communication and heavier preprocessing, but enables more efficient homomorphic operations during the online phase.

While both approaches are compatible with our design, we adopt the latter in our implementation to optimize the online performance, in particular, the shuffling time at the router. We emphasize that, once the setup is completed among a fixed set of clients, the resulting key can be reused for polynomially many rounds of interaction with the router.

We introduce the homomorphic operations used in this paper in an abstract manner, independent of the underlying ciphertext representation.

Let $ct \leftarrow \mathsf{Enc}_{pk}(m)$ denote an encryption of a message m under public key pk.

Homomorphic Addition. Given two ciphertexts $ct_1 = \mathsf{Enc}_{pk}(m_1)$ and $ct_2 = \mathsf{Enc}_{pk}(m_2)$, the homomorphic addition is defined as $ct_+ := \mathsf{Add}(ct_1, ct_2)$, which produces a ciphertext encrypting $m_1 + m_2$.

Homomorphic Multiplication. Given two ciphertexts $ct_1 = \mathsf{Enc}_{pk}(m_1)$ and $ct_2 = \mathsf{Enc}_{pk}(m_2)$, we denote their homomorphic multiplication by $ct_{\mathsf{mul}} \leftarrow \mathsf{Mul}(ct_1, ct_2)$, where ct_{mul} encrypts the product $m_1 m_2$. In our TFHE instantiation, Mul is realized differently depending on the operand types:

- Bootstrapped.AND: When both operands encrypt Boolean values, we evaluate their product using this bootstrapped AND gate [17, 19].
- ExternalProduct: When an encrypted control bit is multiplied by an RLWE ciphertext, we use this, where the control bit is represented in RGSW ciphertext[4] and acts as an encrypted selector for the RLWE ciphertext.

Circuit Bootstrapping. Circuit bootstrapping converts an LWE encryption of a bit into an RGSW encryption of the same bit. More precisely, given $ct = \mathsf{LWE.Enc}(b)$ for $b \in \{0, 1\}$, circuit bootstrapping produces $\widetilde{ct} \leftarrow \mathsf{CircuitBootstrap}(ct)$, where $\widetilde{ct}$ is an RGSW encryption of b. At the same time, the bootstrapping procedure refreshes the ciphertext and reduces the accumulated noise to a controlled level. We refer to [74] for the detail and an efficient realization.

One-round Decryption Protocol. We assume that a ciphertext ct is encrypted under a jointly generated public key $\hat{pk}$, whose corresponding secret key is $s = F(s_1, \ldots, s_\eta)$. No single party knows s, and each client holds only its secret share s_i. Decryption is performed collaboratively among the clients in a single round [60]. The protocol consists of the following two procedures:

- $\mathsf{Partial.Dec}(ct, s_i) \rightarrow d_i$: Each client uses its secret share s_i to compute a partial decryption share d_i.
- $\mathsf{Final.Dec}(ct, \{d_i\}_{i\in[\eta]})$: The final message is recovered by combining all partial decryption shares.

4 Anonymous Router Construction

With this version of protocol, the server maintains η bins/buckets. Each user for each message picks three random indices, and the message is written to the bucket corresponding to one of the indices. The server obliviously chooses the least full bucket among the three to write the message, by operating on the homomorphic ciphertexts.

4.1 Setup

We consider a round-based protocol with a one-time setup phase and many subsequent communication rounds. In the setup phase, the clients jointly execute a distributed key-generation procedure. More precisely, the clients first generate a single global secret key s via an MPC-based setup, without revealing s to any individual party [76]. Instead, each client u_j receives only a local secret share s_j such that $s = F(s_1, \ldots, s_\eta)$, for an appropriate reconstruction function F (e.g., additive reconstruction in the simplest instantiation). Based on this shared representation of s, the parties jointly generate the corresponding public key $\hat{\mathsf{pk}}$ together with all evaluation keys needed by the server.

4.2 Online Phase

We consider that each sender sends exactly one message in every round. The online phase occurs in the following five steps.

Server: Setting up Buckets. If there are η senders, the server creates a matrix L of size $\eta \times 3$, each element in the matrix contains an encryption of 0. The server also creates an auxiliary list I of size $\eta \times 3$, each element is an encryption of 1. The value 1 for an element in I denotes that the corresponding position in L is available to write.

[3] We refer to [19] for the detailed definition of the $\mathsf{SampleExtract}(\cdot)$ procedure.

[4] For the definition and structure of RGSW, see [17, 18, 20].

Client: Encryption. Suppose client u_j holds a message m_j. Using the public parameters generated in the setup phase, the client performs the following steps:

- The client encrypts m_j under a homomorphic encryption scheme and obtains $c_j \leftarrow \mathsf{Enc}_{\hat{\mathsf{pk}}}(m_j)$.
- The client samples indices for three candidate buckets: $a_{j,d} \leftarrow [0, \eta - 1]$, where $d \in \{1, 2, 3\}$.
- For each candidate index $a_{j,d}$ with $d \in \{1, 2, 3\}$, the client constructs the corresponding one-hot plaintext vector $\mathbf{e}_{a_{j,d}} \in \{0, 1\}^{\eta}$ whose only nonzero entry is at position $a_{j,d}$.
- The client encodes each $\mathbf{e}_{a_{j,d}}$ as a plaintext polynomial in the coefficient representation and encrypts it under the common public key $\hat{\mathsf{pk}}$ to obtain $Q_{j,d} \leftarrow \mathsf{Enc}_{\hat{\mathsf{pk}}}(\mathbf{e}_{a_{j,d}})$ for each $d \in \{1, 2, 3\}$.
- Finally, the client sends the tuple $(c_j, Q_{j,1}, Q_{j,2}, Q_{j,3})$ to the server over a public channel.

At the end of this phase, the server receives η tuples $(c_j, Q_{j,1}, Q_{j,2}, Q_{j,3})$ for $j \in \{1, \ldots, \eta\}$, where each tuple is submitted by client u_j.

Server: Write. After the client-encryption phase, the server receives η tuples $(c_j, Q_{j,1}, Q_{j,2}, Q_{j,3})$ for $j \in [\eta]$, where c_j denotes the RLWE encrypted message, and each $Q_{j,d}$ is an RLWE encryption of a one-hot plaintext polynomial corresponding to one candidate bucket. For each tuple, the server performs the following steps:

- For each candidate $d \in \{1, 2, 3\}$ and and each bucket $i \in \{1, \ldots, \eta\}$, the server extracts the i-th coefficient of $Q_{j,d}$ to obtain an LWE ciphertext $z_{i,d}$ which encrypts the selector bit $(\mathbf{e}_{a_{j,d}})_i$. Thus, $z_{i,d}$ encrypts 1 if the d-th candidate of client u_j points to bucket i, and 0 otherwise. Repeating this for all i yields η encrypted selector bits $\{z_{i,d}\}_{i \in [\eta]}$ for each candidate d.
- The server applies key switching to these derived ciphertexts, thereby reducing the LWE dimension and lowering the computation cost of the subsequent homomorphic operations.
- Using the resulting bits together with the current encrypted state (L, I), the server runs HomPlacing to insert c_j into the least-loaded bucket among the three candidates and to update L and I accordingly. The detailed procedure is given in Algorithm 1.
- After all client tuples have been processed, the server obliviously *sorts* [23] the three slots in each bucket to remove any residual within-bucket ordering leakage.

Then, the server broadcasts all elements of L to all clients. We note that the protocol has no overflow with high probability due to the analysis discussed in Section 4.5. Additionally, if there is an error in message delivery because of an overflow, the client can simply retransmit the message without deteriorating anonymity.

Client: Partial Decryption. After receiving the encrypted bucket array L from the server, each client computes a partial decryption share for every component of L using its local secret share s_i. Specifically, each client u_i computes $d_i \leftarrow \mathsf{Partial.Dec}(s_i, L)$ and sends the resulting partial decryption share d_i back to the server. We note that d_i is a vector of dimension $\eta \times 3$, where each component corresponds to one entry of L.

Algorithm 1 Homomorphic placing HomPlacing for three choices

1: **Input:** an HE message ciphertext c, and distinct indices ciphertexts $\{z_{i,d}\}_{i \in [\eta], d \in [3]}$, where $z_{i,d}$ encrypts whether the d-th choice points to bucket i.
2: **State:** arrays $L_{n \times 3}$ and $I_{n \times 3}$, where $L_{i,k}$ stores the current content of slot (i, k) and $I_{i,k}$ encrypts whether that slot is available.
3: **Output:** an encrypted bit hasWritten indicating whether the insertion succeeds.
4: hasWritten $\leftarrow$ Enc(0)
5: **for** $k \leftarrow \{1, 2, 3\}$ **do** // number of slots //
6: **for** $d \leftarrow \{1, 2, 3\}$ **do** // number of choices //
7: **for** $i \leftarrow \{1, \ldots, \eta\}$ **do** // number of buckets //
8: $u_{i,k,d} \leftarrow \mathsf{Mul}(z_{i,d}, I_{i,k})$
9: $\bar{w} \leftarrow \mathsf{NOT}(\mathsf{hasWritten})$
10: $h_{i,k,d} \leftarrow \mathsf{Mul}(u_{i,k,d}, \bar{w})$
11: $L_{i,k} \leftarrow L_{i,k} + \mathsf{Mul}(h_{i,k,d}, c)$
12: $I_{i,k} \leftarrow I_{i,k} - h_{i,k,d}$
13: hasWritten $\leftarrow$ hasWritten $+ h_{i,k,d}$
14: **end for**
15: **end for**
16: **end for**
17: **return** hasWritten

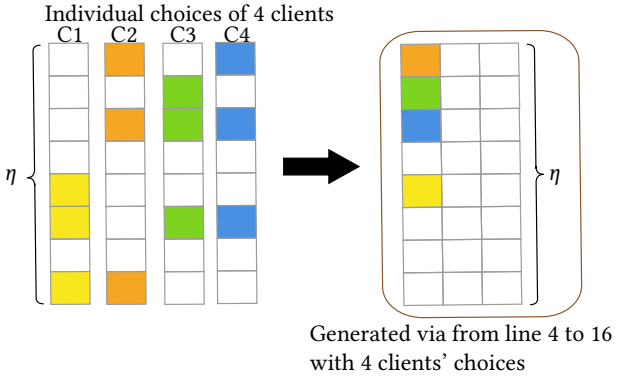


Figure 3: A high-level illustration of Algorithm 1. Each client independently selects three candidate buckets, shown in different colors. The arrays on the left represent the encrypted selector bits derived from the clients' queries, and the matrix on the right shows the resulting $\eta \times 3$ bucket array after four clients have been processed. Each message is inserted into the least-loaded available slot among its three candidate buckets.

Server: Combination and Final Decryption. After receiving the partial decryptions $d_1, \ldots, d_\eta$ from all the clients, the server combines them by running $\mathsf{Final.Dec}(d_1, \ldots, d_\eta)$ entry-wise on L. This yields the shuffled list of encrypted messages. The server discards the 0 entries and forwards the remaining messages.

4.3 Optimizing bandwidth

Our design can utilize the following techniques for bandwidth optimization:

(1) **Packing LWEs to RLWE.** We can optimize the communication complexity and the concrete communication cost by using a packing method. We first recall that the complexity of distributed messages over a public channel after partial decryption is $O(\eta^2)$, which is not ideal in practice. Therefore, we would like each client to pack the partial decryption shares corresponding to all 3η entries into a smaller number of ciphertexts. Then the complexity would become $O(\eta^2/N)$, where typically N is chosen larger than 2048 in many FHE applications [14, 21, 59, 63]. For application scenarios with a small number of clients, the concrete communication cost can therefore be significantly improved.
Let u denote the flattened encrypted output vector obtained from the bucket array L, so that u consists of 3η RLWE ciphertexts. Then the partial decryption output d_i of client u_i also consists of 3η RLWE ciphertexts. To reduce the communication cost, one may transform each RLWE ciphertext into an LWE ciphertext by viewing the RLWE ciphertext as a vector and rearranging its components, and then apply the LWEs-to-RLWE conversion [15, 48] to generate a smaller number of RLWE ciphertexts, each of which packs multiple partial decryption shares into the coefficients of a message polynomial.

(2) **Using homomorphic traversal algorithm.** In order to reduce client's message complexity from $O(\eta)$ to $O(\log \eta)$, we can employ homomorphic de-mux algorithm introduced in [22] as a homomorphic traversal algorithm. With this algorithm, client only sends $\log \eta$ bits (encrypted) representing its chosen index, then the server expands it to an one-hot vector (encrypted). We explain the method in Appendix C. As a trade-off, the algorithm will incur $O(\eta)$ computation complexity for client, and the noise level will increase as well.

4.4 Sender Anonymity Property

In this section, we analyze the sender anonymity property achieved by our protocol sPAR against an honest-but-curious adversary. Recall that, according to our threat model, all recipients are corrupted, the server is corrupted, and all but two senders are corrupted.

Before going into the analysis, we explain how HomPlacing realizes the choice-of-three/balls-and-bins process under encryption.

Plaintext Placement Process Induced by HomPlacing**.** Before proving sender anonymity, we first identify the plaintext process that is realized by Algorithm 1. In the balls-and-bins terminology, each message is a ball, each bucket is a bin, and the three indices sampled by a client are the three choices for that ball. Each bin has capacity three, corresponding to the three slots in each bucket.

Consider a fixed client u_ℓ and ignore encryption for the moment. Let B_d denote the bin selected by the d-th choice of this client, for $d \in \{1, 2, 3\}$. The encrypted selector bit $z_{b,d}$ represents the plaintext indicator of the event $B_d = b$, while $I_{b,k}$ represents whether the k-th slot of bin b is still available. Hence, the product $z_{b,d} \cdot I_{b,k}$ is active exactly when the d-th chosen bin is b and slot (b, k) is available.

The triple loop implements least-loaded selection among the three candidate bins. Under the invariant that slots in each bin are filled in order, the k-th slot is the first available slot of a bin exactly when that bin currently has load $k-1$. Therefore, iterating over the slot index k in the outermost loop scans the candidate bins in increasing order of their current load: $k = 1$ first checks for an empty candidate bin, $k = 2$ checks for a candidate bin with one occupied slot, and $k = 3$ checks for a candidate bin with two occupied slots. For each load level k, the algorithm scans the three candidate choices and all bins using the encrypted selector bits.

The first successful write therefore occurs at the smallest load level among the three chosen bins. Hence, the ball is placed into a least-loaded bin among its three choices, with ties broken by the fixed order of the choices.

The flag hasWritten ensures that the message is inserted at most once. Once a valid slot is found, hasWritten becomes active and all later candidate positions are ignored. If all three chosen bins are already full, no write occurs and hasWritten remains zero, which corresponds to an overflow event. Therefore, in the plaintext view, Algorithm 1 implements the choice-of-three balls-and-bins allocation with bin capacity three.

The following lemma analyzes the plaintext placement process induced by our encrypted protocol.

Lemma 1. *Given a matrix of size $\eta \times 3$ (each element initialized to $\perp$), and a total of η incoming elements, suppose each incoming element is placed in the matrix with the following strategy: Three values a_1, a_2, a_3 are chosen uniformly at random from $\{0, \ldots, \eta - 1\}$, and the element is placed (or overwritten) in the position (i, j) such that $i \in \{a_1, a_2, a_3\}$ and i-th column has the most number of $\perp$ among them, and j is the number of $\perp$ in the i-th column. After all the incoming elements are placed, the elements of each column are sorted (according to their values). Given that all elements have been placed successfully, the relative order (of placement) of two arbitrary elements is uniformly random if the order of their values is uniformly random, independent of their order of arrival.*

Proof. We can show this by induction on the number of elements that have already arrived.

Base Case (Matrix Size $\eta \times 3$, Number of Elements = 2). When the first element arrives, all the slots are empty. Therefore, the first element would be placed in the first row. Let us assume that the index in the first row is x where $x < \eta$. For the second element, it is possible that $a_1 = x$; however, in that case, the element will be placed in $(a_2, 0)$ (assuming a_2 is also not a collision). The probability of $a_1 < a_2$ is 0.5.

If a_2 is also a collision, we can make identical arguments about a_3. If a_3 is also a collision, we are outside our consideration of successful placements of all elements.

Inductive Hypothesis. Let us assume that the lemma holds for any number of elements $< \chi \leq \eta$. We want to show that it holds for χ elements. If either of the chosen two elements is not the χ-th element, the lemma holds (based on the inductive hypothesis).

If one of those two elements is the χ-th element, there is a probability that one of $\{a_1, a_2, a_3\}$ collides with the other element. If they are not placed in the same column, the partial order of the two elements is random (the partial order of two random numbers is random). If they are placed in the same column, the shuffle (of the column) step makes their partial order uniformly random.

We ignore the scenarios where a column could overflow (with a very small probability), since we only consider successful placement of all elements. □

Using the above lemma, we can show that our protocol achieves sender anonymity. Intuitively, the same scramble is executed by the server in the actual protocol, but under FHE. And, since the adversary does not know the inputs from the honest senders, the partial ordering of honest messages is unpredictable to the adversary.

Theorem 1 (Sender Anonymity). *Assuming IND-CPA security of the underlying FHE scheme $\mathcal{E}$, our protocol sPAR provides sender anonymity as defined in Definition 1 with $\delta \in neg(\lambda)$, where λ is the security parameter.*

Proof. We want to show the following: if there exist an adversary $\mathcal{A}_{\mathsf{SA}}$ that can achieve non-negligible δ in the sender anonymity game $\mathcal{G}_{\mathsf{SA}}^{\Pi,\mathcal{A},\eta}(1^\lambda)$, we can construct an adversary $\mathcal{A}_{FHE}$ that can break IND-CPA security of the underlying FHE scheme $\mathcal{E}$.

In our argument, we use an equivalent modified version of the standard IND-CPA game used to define the security of $\mathcal{E}$: on a given challenge message pair (m_0, m_1), the challenger C_{FHE} of the encryption system returns a ciphertext pair $c_0 = \mathsf{Enc}_{\hat{\mathsf{pk}}}(m_b)$ and $c_1 = \mathsf{Enc}_{\hat{\mathsf{pk}}}(m_{1-b})$ instead of just one ciphertext $\mathsf{Enc}_{\hat{\mathsf{pk}}}(m_b)$.[5]

The challenger of the anonymity game C_{SA} uses the encryption oracle from the IND-CPA game for all encryptions — that ensures the same encryption key in both the games. Recall that, in the sender anonymity game, the adversary controls all the clients except u_0 and u_1, as well as the router/server. If $\mathcal{A}_{\mathsf{SA}}$ uses the challenge pair (u_0, r^*, m_0) and (u_1, r^*, m_1), $\mathcal{A}_{FHE}$ uses the challenge message pair (m_0, m_1). Additionally, $\mathcal{A}_{FHE}$ intercepts the encrypted message from u_0 (and u_1 resp.) and replace that with $c_0 = \mathsf{Enc}_{\hat{\mathsf{pk}}}(m_b)$ (and $c_1 = \mathsf{Enc}_{\hat{\mathsf{pk}}}(m_{1-b})$ resp.). Then C_{SA} and $\mathcal{A}_{\mathsf{SA}}$ runs our protocol, including the partial decryption steps (without revealing the private keys of u_0 and u_1 to $\mathcal{A}_{\mathsf{SA}}$ or $\mathcal{A}_{FHE}$), until $\mathcal{A}_{\mathsf{SA}}$ outputs a guess for the challenge bit. If the adversary $\mathcal{A}_{\mathsf{SA}}$ in the anonymity game returns 0, $\mathcal{A}_{FHE}$ also returns 0.

Here we utilize two facts: (1) since the challenger assigns m_0 and m_1 to the challenge users based on the random challenge bit b, the values of m_b and m_{1-b} are ordered randomly; (2) And because of the above, Lemma 1 is applicable here: $\mathcal{A}_{FHE}$ does not learn anything by correlating the order of sending and the final order of messages . That means $\mathcal{A}_{FHE}$ has the advantage of δ over a random coin toss to break IND-CPA game. □

4.5 Failure Probability With Choice of Three

In Algorithm 1, only one of the indices is written out of the three chosen by the client: if the first index cannot be placed in the first segment (each segment of η slots, and 3 segments in total), the second index is placed in the second half. This maps to the following balls-and-bins problem: there are η bins, and each bin has a capacity of 3; three indices are chosen for each ball, and the ball is placed to the bin with the lower ball count. And if the two or more bins have the same ball count, the tie is broken by using the first index chosen by the user. In this scenario, the total number of bins is η, and a total of η balls are placed one after another. An extremely low probability that any of the bins has an overflow directly implies an extremely low probability that a message cannot be placed in the list maintained by the server in our protocol.

	$d = 2$		$d = 3$	
	Prediction	Sim.(1M)	Prediction	Sim.(1M)
$n = 1$	0.7616	0.7616	0.8231	0.8230
$n = 2$	0.2295	0.2295	0.1765	0.1765
$n = 3$	0.0089	0.0089	0.00051	0.00051
$n = 4$	0.000006	0.000007	$< 10^{-11}$	0
$n = 5$	$< 10^{-11}$	0	$< 10^{-11}$	0

Table 2: The concrete probability analysis from [57] for different loads in the bins. $n = i$ in the left column represents the number of balls in a bin, d represents the number of choices for each ball, and value in each cell represents the probability, with #total balls = #total bins.

Based on the analysis from [57], we know that the max load for a bin in such ball-and-bins problem is bounded by $\log\log\eta/\log d + O(1)$ for η bins and d choices for every ball. We refer to Figure 4.1 of [57] for the concrete probability analysis for different loads in the bins. We present a summary in Table 2. In the table, n in the left column represents the number of balls in a bin, d represents the number of choices for each ball, and each cell represents the probability of having n balls in a bin when the balls are distributed with choice-of-d, with total #balls = total #bins = η. From their analysis, with the choice $d = 3$ and the probability of having 4 balls or more in a bin ($n \geq 4$) is extremely unlikely ($< 10^{-11}$), which directly translates to the server maintaining 3 slots for each bucket in our protocol.

5 Implementation and Evaluations

In this section, we present the empirical performance of sPAR based on a concrete implementation. In particular, we evaluate the execution time of each protocol step, including client-side query generation, server-side homomorphic placing, oblivious sorting, and partial decryption, for varying numbers of clients.

Experimental Setup. We implemented sPAR in C++ on top of a custom 64-bit TFHE backend [20] that uses the `SPQLIOS` FFT library [71] for negacyclic polynomial multiplication. All experiments were run on Ubuntu 20.04.6 LTS under WSL2 on a Windows 10 Pro host. The host machine was equipped with an Intel Core i9-12900K (16 cores / 24 threads, 3.20 GHz) and 32 GB RAM; the CPU supports AVX/AVX2 with FMA. The WSL2 guest was configured with 4 vCPUs and 14 GiB RAM (8 GiB swap).[6]

Parameters. The concrete cryptographic parameters used throughout our experiments are summarized in Table 3. We adopt the standard TFHE setting $(N, k) = (2048, 1)$ with noise standard deviation $\sigma = 2^{-54}$, which is used both for bootstrapping keys and client-side encryption. For the gate bootstrapping path we use a smaller LWE dimension $n = 1024$ in order to reduce the online computation cost. Since reducing the LWE dimension alone would weaken the underlying LWE assumption, we compensate for this by increasing the noise of the key-switching key used to map ciphertexts from

[5] By defining an appropriate hybrid, this modified game can be easily shown to be equivalent to the standard CPA-security encryption game.

[6] We compiled with gcc/g++ 9.4.0 (glibc 2.31) using `-O3` and `-march=native`, and enabled OpenMP via `-fopenmp`. All experiments were executed with 4 OpenMP threads.

dimension N to n, setting its standard deviation to $\sigma_{\mathsf{ksk}} = 2^{-26}$. We verified all parameter choices using the lattice estimator [2], and every LWE/RLWE instance arising in our implementation provides at least 117 bits of estimated security.

The remaining decomposition parameters in Table 3 are used for circuit-bootstrapping subroutines [74]. We use finer decompositions than for standard gate bootstrapping [17]. Each partial-decryption share is additionally masked with fresh Gaussian noise of standard deviation $\sigma_{\mathsf{pd}} = 2^{-54}$. The two refresh periods specify how often the encrypted state variables hasWritten and I are refreshed. Since both are repeatedly updated, we refresh them periodically to prevent excessive noise growth.

Our implementation code can be found at the provided GitHub URL.[7]

Table 3: Implementation Parameters

Parameter	Value
Global parameters	
polynomial degree N	2048
LWE dimension n	1024
GLWE rank k	1
plaintext modulus t	2^4
ciphertext modulus q	2^{64}
Noise parameters	
encryption σ	2^{-54}
key switching σ_{ksk}	2^{-26}
partial-decryption smudging σ_{pd}	2^{-54}
Gadget decomposition parameters (β, ℓ) *in bootstrapping*	
gate bootstrapping	$(2^9, 3)$
key switching	$(2^8, 8)$
blind rotation	$(2^9, 5)$
private key switching	$(2^{16}, 4)$
homomorphic trace algorithm	$(2^9, 6)$
scheme switching	$(2^9, 6)$
Others	
refresh period of hasWritten	18
refresh period of I	8

Implementation details. To instantiate our proof-of-concept sPAR and to manage noise growth effectively, we make several specific design choices in our implementation. Since we use TFHE as our backend, we rely on its standard homomorphic operations and refer the reader to [17, 19] for their details.

First, the server extracts encrypted selector bits from the uploaded one-hot RLWE queries. For each query, applying SampleExtract to its relevant coefficients yields LWE encryptions of the corresponding entries of the underlying one-hot plaintext. Repeating this procedure gives the encrypted selector bits used in the placement phase. We then apply key switching to reduce the LWE dimension from N to n, thereby lowering the cost of the subsequent homomorphic operations.

[7] https://github.com/HyewonSung/sPAR

Second, the multiplications in Algorithm 1 are instantiated differently according to the operand types. The products in lines 8 and 10 involve two encrypted control bits and are evaluated using TFHE bootstrapped Boolean gates. In contrast, line 11 multiplies an encrypted control bit by an RLWE-encrypted message and is implemented using ExternalProduct. For this purpose, we circuit-bootstrap $h_{i,k,d}$ from LWE to RGSW. We employ the state-of-the-art circuit-bootstrapping technique of [74], which consists of multi-value functional bootstrapping[8], homomorphic trace, and scheme switching. For the trace subroutine, we further adopt the noise-efficient algorithm of [48], which avoids the constant-factor noise amplification of the conventional trace operation.

Third, the state variables I and hasWritten are updated using subtraction and addition, respectively, so their noise accumulates over successive placements. We therefore refresh them periodically using bootstrapping and the corresponding refresh periods are given in Table 3[9].

Finally, the oblivious sorting is performed entirely under HE using a fixed sorting circuit within each bucket, while keeping both the comparison outcomes and the resulting data movement hidden. To obtain encrypted comparison results that can be used for conditional data movement, we employ circuit bootstrapping to convert the comparison output into the required ciphertext form. Since each bucket contains only three slots, the sorting circuit is small, and different buckets can be processed independently in parallel. So the practical overhead of the circuit-bootstrapping operations is relatively small.

5.1 Performance Evaluation

Table 4 reports the per-message running time of sPAR for $\eta \in \{20, 40, 60, 80, 100\}$ clients, evaluated using four OpenMP threads, while Table 5 reports the corresponding end-to-end latency per round. We leverage parallelization wherever possible : client-side query generation, the server-side conversion of uploaded queries into lower-dimensional LWE ciphertexts, within-bucket oblivious sorting, and partial decryption are executed in parallel, as these operations are independent across clients or buckets. The homomorphic placing phase is sequential across clients, since each insertion updates the encrypted state (L, I) required by the next one. Within each insertion, however, only the control logic that tracks the placement state is inherently sequential. We therefore reorganize the computation so that state-independent operations are evaluated in parallel, while only this small control chain remains sequential. In particular, the expensive circuit-bootstrapping and external-product operations are deferred until the control bits are determined and are then executed in parallel. This reordering preserves the output of Algorithm 1 while significantly reducing the sequential computation.

Latency. Table 4 reports the computation time of each protocol phase on a per-message basis. As shown there, **Client:Send** is essentially constant at 0.002 seconds regardless of η. **Partial Dec.** grows approximately linearly with η, from 0.015 seconds at $\eta = 20$

[8] This directly produces the multi-valued intermediate representation required by circuit bootstrapping, reducing additional conversion and bootstrapping overhead.
[9] Specifically, we refresh the ciphertext by evaluating a bootstrapped AND gate with an encryption of 1, which preserves the plaintext bit while resetting the accumulated noise.

to 0.076 seconds at $\eta = 100$, while **Sorting** remains nearly constant at about 0.21 seconds. The dominant cost is **Server:Write**, which scales approximately linearly with η per message, increasing from 12.1 seconds at $\eta = 20$ to 58.7 seconds at $\eta = 100$. This behavior reflects the linear number of homomorphic placement operations, in particular the bootstrapping operations, performed for each message.

Since the server processes the η messages sequentially, this linear per-message cost results in an overall round latency that grows approximately quadratically with η. The end-to-end figures are summarized in Table 5, ranging from about 4.2 minutes at $\eta = 20$ to about 101.5 minutes at $\eta = 100$. Although these numbers are still far from a fully optimized deployment target, we stress that our protocol is a proof-of-concept implementation of the first HE-based anonymous router over an untrusted server. We therefore view the present measurements as a feasibility baseline for FHE-based anonymous routing, rather than as the final performance limit of the protocol.

Table 4: Computation times (in seconds) required by different steps of sPAR with $\eta \in \{20, 40, 60, 80, 100\}$ clients, one message each.

η	**Client:Send** per message	**Server:Write** per message	**Partial Dec.** 3η ciphertexts	**Sorting** 3 slots
20	0.002 s	12.1 s	0.015 s	0.210 s
40	0.002 s	23.7 s	0.030 s	0.219 s
60	0.002 s	34.8 s	0.045 s	0.212 s
80	0.002 s	46.5 s	0.060 s	0.211 s
100	0.002 s	58.7 s	0.076 s	0.211 s

Bottleneck and Outlook. Profiling shows that the dominant cost of **Server:Write** comes from TFHE bootstrapping operations, including both gate bootstrapping and circuit bootstrapping. This suggests that the protocol can directly benefit from improvements in the underlying TFHE backend without changing the higher-level routing design. In particular, this is encouraging because our protocol can directly benefit from improvements in TFHE implementations without requiring any change to the protocol design.

Indeed, recent optimized TFHE backends already demonstrate substantially faster bootstrapping in practice. For example, Zama reports a boolean bootstrap latency of 796 μs (microseconds) on GPU [49]. To illustrate the potential benefit of a more optimized backend, we project the latency by applying the corresponding GPU speedup to the bootstrapping operations while keeping the remaining costs unchanged. For circuit bootstrapping, we conservatively apply the same relative speedup factor as for gate bootstrapping. Under this projection, the end-to-end latency at $\eta = 100$ decreases from 101.5 minutes to 5.3 minutes. Across the tested parameter range, this corresponds to an approximately 19× reduction in end-to-end latency, as summarized in Table 5.

Since our protocol is intended to run on a heavyweight untrusted server, such as a cloud-provisioned machine, this setting is naturally compatible with GPU-based implementations and other dedicated hardware accelerators. We therefore expect the practical latency of our protocol to decrease substantially as optimized TFHE implementations continue to mature.

Table 5: End-to-end latency (in seconds) per round of sPAR. The right column projects the estimated latency when the bootstrapping backend is replaced by Zama's GPU implementation [49].

η	Current backend	Zama-projected[†]
20	251.7 s	13.1 s
40	979.6 s	50.7 s
60	2167.3 s	112.7 s
80	3854.2 s	200.5 s
100	6087.9 s	318.9 s

[†] Each gate bootstrap (27η per message, 20 ms in our backend) is replaced by Zama's 0.796 ms bootstrap, and each circuit bootstrap (9η per message, 59.6 ms) is scaled by the same GPU factor to 2.37 ms. Non-bootstrap costs are kept at the measured values.

6 Extensions And Discussions

6.1 Client Churn And Threshold Version

The protocol specification in Section 4 does not consider the scenario where some clients might become inactive/unavailable in some rounds. If such a scenario occurs, the parties will be unable to retrieve the messages, since each client needs to provide their partial decryption. Without any major modification in the protocol, it is possible to use *Threshold Homomorphic Encryption* [3]: as long as a threshold number of clients are available, the decryption step would be possible. They can even decide to decrypt even when not all the clients send their messages.

Note that such a threshold setting also impacts the overall trust assumption for the system: if the number of corrupted clients is more than the threshold, the adversary will be able to decrypt all messages and break the anonymity guarantees of the system. Additionally, we consider the specific nuances (how to detect an inactive client, the quantification of anonymity when some clients do not send messages, and possible performance optimization) of the threshold version as future work.

6.2 Recipient Anonymity

In order to achieve recipient anonymity, our design can integrate existing OMR protocols (e.g., PerfOMR [52]) as is, with minor modifications in our design. Each message can be encrypted with another layer of encryption (that is compatible with the relevant OMR protocol) under the recipient's public key.

Each recipient $r_i \in \mathcal{R}$ creates a public-key $\mathsf{pk}_i^{\text{recipient}}$ associated with them, and only r_i knows the corresponding secret key $\mathsf{sk}_i^{\text{recipient}}$. Similar to *onion services* in Tor, this public key $\mathsf{pk}_i^{\text{recipient}}$ is publicly known but the corresponding physical identity r_i is not known to anyone. [10] Suppose the sender u_i wants to send a message m_i to the recipient with public-key $\mathsf{pk}_j^{\text{recipient}}$. Then u_i first encrypts the message m_i with $\mathsf{pk}_j^{\text{recipient}}$, to generate m_i'.[11] Then u_i computes $c_i = \mathsf{Enc}_{\hat{\mathsf{pk}}}(m_i')$ and sends it to the server.

[10] This could be achieved by using our protocol in Section 4.2 as a sender, and send $\mathsf{pk}_i^{\text{recipient}}$ to all the desired parties.

[11] m_i' could also include additional details, besides the encryption of m_i, required by the OMR protocol without impacting the end-to-end design.

Then the server runs the protocol identical to the one Section 4.2. However, after the decryption phase, the server has a set of output messages $\{m'_i\}_{i\in\{1,\ldots,\eta\}}$. Then the server runs an additional step with the OMR protocol for the recipients, such that only the intended recipient r_j can decrypt m'_i to retrieve the message m_i.

The anonymity guarantee for the recipients directly follows from the security guarantees of the underlying OMR scheme. We skip a detailed security analysis corresponding to the application of a specific OMR scheme here. We consider the possible optimizations and improvements specific to integrating OMR with our design as interesting directions for future work.

6.3 Other Future Works

Our work is a first design of its kind and leaves a lot of scope for future improvements. Some of those future directions include:

- We do not consider any active attacks in our current design. However, it is possible for the server to tamper with messages, combined with equivocation towards the goal of breaking anonymity; or some malicious clients could send malformed messages to disrupt the protocol. We envision that such active attacks could be prevented by techniques based on Verifiable FHE [73]. It is an important future direction to investigate possible attacks and defenses.
- Our communication model explicitly assumes that each client sends exactly one message in every round. However, in a real-world scenario, different clients could send messages at different frequencies, and could be active/inactive at different times of the day. Such differences could allow deanonymization [54, 64]). The effective strategies for allowing such variations with the current design strategy are interesting future work.

7 Conclusion

We consider our work as a second step towards designing anonymous messaging systems without any trusted infrastructure. Our work has followed the footsteps of Shi and Wu [69] that showed the theoretical possibility of such systems. We do not attempt to achieve high scalability of Tor [32] or Mixnets [31, 64], nor the performance of some MPC-based systems [33, 34]. Still, our proof-of-concept implementation demonstrates that our design is implementable, and suitable for small-scale applications with up to a hundred users, where the application can tolerate several seconds of delay per message. Our design strategy shows a new direction for designing anonymous messaging systems using FHE techniques.

Acknowledgment

This work has been partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] Ittai Abraham, Benny Pinkas, and Avishay Yanai. 2020. Blinder – Scalable, Robust Anonymous Committed Broadcast. In *Proceedings of the 2020 ACM SIGSAC CCS*. 1233–1252.

[2] Martin R. Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology* 9, 3 (2015), 169–203.

[3] Gilad Asharov, Abhishek Jain, and Daniel Wichs. 2011. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. Cryptology ePrint Archive, Paper 2011/613. https://eprint.iacr.org/2011/613

[4] Michael Backes, Aniket Kate, Praveen Manoharan, Sebastian Meiser, and Esfandiar Mohammadi. 2013. AnoA: A Framework for Analyzing Anonymous Communication Protocols. In *2013 IEEE 26th Computer Security Foundations Symposium*. IEEE Computer Society, New Orleans, LA, USA, 163–178. doi:10.1109/CSF.2013.18

[5] Ludovic Barman, Italo Dacosta, Mahdi Zamani, Ennan Zhai, Apostolos Pyrgelis, Bryan Ford, Joan Feigenbaum, and Jean-Pierre Hubaux. 2020. PriFi: Low-Latency Anonymity for Organizational Networks. *Proc. Priv. Enhancing Technol.* 2020, 4 (2020), 24–47.

[6] Gabrielle Beck, Julia Len, Ian Miers, and Matthew Green. 2021. Fuzzy Message Detection. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 1507–1528. doi:10.1145/3460120.3484545

[7] Joseph Bonneau, Jeremy Clark, Joshua A. Kroll, Andrew Miller, and Arvind Narayanan. 2014. Mixcoin: Anonymity for Bitcoin with accountable mixes. In *Proceedings of Financial Cryptography and Data Security (FC'14)*.

[8] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. 2022. FINAL: Faster FHE Instantiated with NTRU and LWE. 188–215. doi:10.1007/978-3-031-22966-4_7

[9] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS '12*.

[10] Paul Bunn, Eyal Kushilevitz, and Rafail Ostrovsky. 2023. Anonymous Permutation Routing. In *Theory of Cryptography: 21st International Conference, TCC 2023, Taipei, Taiwan, November 29–December 2, 2023, Proceedings, Part III* (Taipei, Taiwan). Springer-Verlag, Berlin, Heidelberg, 33–61. doi:10.1007/978-3-031-48621-0_2

[11] David Chaum. 1988. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. 1, 1 (Jan. 1988), 65–75. doi:10.1007/BF00206326

[12] David Chaum. 1988. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology* 1, 1 (1988), 65–75.

[13] David Chaum, Farid Javani, Aniket Kate, Anna Krasnova, Joeri Ruiter, Alan T Sherman, and Debajyoti Das. 2016. *cMix: Anonymization by high-performance scalable mixing*. Technical Report. Technical report. http://www.cs.bham.ac.uk/~deruitej/papers/cmix.pdf

[14] Hao Chen, Ilaria Chillotti, and Ling Ren. 2019. Onion Ring ORAM: Efficient Constant Bandwidth Oblivious RAM from (Leveled) TFHE. 345–360. doi:10.1145/3319535.3354226

[15] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2021. Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts. 460–479. doi:10.1007/978-3-030-78372-3_18

[16] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. 409–437. doi:10.1007/978-3-319-70694-8_15

[17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2016. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. 3–33. doi:10.1007/978-3-662-53887-6_1

[18] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2017. Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE. 377–408. doi:10.1007/978-3-319-70694-8_14

[19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. 33, 1 (Jan. 2020), 34–91. doi:10.1007/s00145-019-09319-x

[20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *J. Cryptol.* 33, 1 (Jan. 2020), 34–91. doi:10.1007/s00145-019-09319-x

[21] Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder V.L. Pereira. 2022. SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 563–577. doi:10.1145/3548606.3560702

[22] Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder V. L. Pereira. 2022. SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering. 563–577. doi:10.1145/3548606.3560702

[23] Kelong Cong, Robin Geelen, Jiayi Kang, and Jeongeun Park. 2025. Revisiting Oblivious Top-k Selection with Applications to Secure k-NN Classification. In *Selected Areas in Cryptography – SAC 2024: 31st International Conference, Revised Selected Papers, Part I*. 3–25. doi:10.1007/978-3-031-82852-2_1

[24] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. 2015. Riposte: An anonymous messaging system handling millions of users. In *IEEE Symposium on Security and Privacy*. 321–338.

[25] Henry Corrigan-Gibbs and Bryan Ford. 2010. Dissent: Accountable Anonymous Group Messaging. In *Proc. ACM SIGSAC CCS*. 340–350.

[26] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. 2013. Proactively Accountable Anonymous Messaging in Verdict. In *Proc. 22nd USENIX Security Symposium*. 147–162.

[27] George Danezis and Claudia Diaz. 2008. *A Survey of Anonymous Communication Channels*. Technical Report MSR-TR-2008-35. 46 pages. https://www.microsoft.com/en-us/research/publication/a-survey-of-anonymous-communication-channels/

[28] Debajyoti Das, Easwar Mangipudi, and Aniket Kate. 2022. OrgAn: Organizational Anonymity with Low Latency. *Proceedings on Privacy Enhancing Technologies* 2022 (07 2022), 582–605. doi:10.56553/popets-2022-0087

[29] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2018. Anonymity Trilemma: Strong Anonymity, Low Bandwidth Overhead, Low Latency - Choose Two. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, San Francisco, California, USA, 108–126.

[30] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2020. Comprehensive Anonymity Trilemma: User Coordination is not enough. *Proceedings on Privacy Enhancing Technologies* 2020, 3 (2020), 356–383. doi:10.2478/popets-2020-0056

[31] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2021. Divide and Funnel: a Scaling Technique for Mix-Networks. Cryptology ePrint Archive, Paper 2021/1685. https://eprint.iacr.org/2021/1685 https://eprint.iacr.org/2021/1685.

[32] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The Second-Generation Onion Router. 303–320. doi:10.21236/ada465464

[33] Saba Eskandarian and Dan Boneh. 2021. Clarion: Anonymous communication from multiparty shuffling protocols. *Cryptology ePrint Archive* (2021).

[34] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. 2021. Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic Privacy. 1775–1792.

[35] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144. https://eprint.iacr.org/2012/144.

[36] Philippe Golle and Ari Juels. 2004. Dining Cryptographers Revisited. In *Proc. of Eurocrypt 2004*.

[37] Ethan Heilman, Leen AlShenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. 2017. TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub. doi:10.14722/ndss.2017.23086

[38] Diana L. Huete Trujillo and Antonio Ruiz-Martínez. 2021. Tor Hidden Services: A Systematic Literature Review. *Journal of Cybersecurity and Privacy* 1, 3 (2021), 496–518. doi:10.3390/jcp1030025

[39] Yanxue Jia, Debajyoti Das, Wenhao Zhang, and Aniket Kate. 2025. Kerblam — Anonymous Messaging System Protecting Both Senders and Recipients. Cryptology ePrint Archive, Paper 2025/997. https://eprint.iacr.org/2025/997

[40] Yanxue Jia, Varun Madathil, and Aniket Kate. 2024. HomeRun: High-efficiency Oblivious Message Retrieval, Unrestricted. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*.

[41] Aniket Kate, Greg M Zaverucha, and Ian Goldberg. 2010. Pairing-based onion routing with improved forward secrecy. *ACM Transactions on Information and System Security (TISSEC)* 13, 4 (2010), 1–32.

[42] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security and Privacy (SP)*. 1–19. doi:10.1109/SP.2019.00002

[43] Christiane Kuhn, Martin Beck, and Thorsten Strufe. 2020. Breaking and (Partially) Fixing Provably Secure Onion Routing. In *Proceedings of the 41st IEEE Symposium on Security and Privacy*. IEEE, 168–185.

[44] Albert Kwon, David Lu, and Srinivas Devadas. 2020. XRD: Scalable Messaging System with Cryptographic Privacy. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, Ranjita Bhagwan and George Porter (Eds.). USENIX Association, Santa Clara, CA, USA, 759–776. https://www.usenix.org/conference/nsdi20/presentation/kwon

[45] Simon Langowski, Sacha Servan-Schreiber, and Srinivas Devadas. 2022. Trellis: Robust and Scalable Metadata-private Anonymous Broadcast. Cryptology ePrint Archive, Paper 2022/1548.

[46] David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2018. Karaoke: Distributed private messaging immune to passive traffic analysis. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 711–725.

[47] David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2019. Yodel: Strong Metadata Security for Real-Time Voice Calls. In *12th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2019)*. https://petsymposium.org/2019/files/hotpets/yodel-final.pdf

[48] Kang Hoon Lee and Ji Won Yoon. 2026. Homomorphic Field Trace Revisited: Breaking the Cubic Noise Barrier. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2026, 1 (Jan. 2026), 105–134.

[49] Agnes Leroy. 2025. *Bootstrapping TFHE ciphertexts in less than one millisecond*. Zama. https://www.zama.org/post/bootstrapping-tfhe-ciphertexts-in-less-than-one-millisecond

[50] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 973–990. https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[51] Zeyu Liu and Eran Tromer. 2022. Oblivious Message Retrieval. In *Advances in Cryptology – CRYPTO 2022*. 753–783. doi:10.1007/978-3-031-15802-5_26

[52] Zeyu Liu, Eran Tromer, and Yunhao Wang. 2024. PerfOMR: Oblivious Message Retrieval with Reduced Communication and Computation. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 3169–3186. https://www.usenix.org/conference/usenixsecurity24/presentation/liu-zeyu

[53] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. 2019. Honeybadgermpc and Asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *Proceedings of the 2019 ACM SIGSAC CCS*. 887–903.

[54] Zhongtang Luo, Adithya Bhat, Kartik Nayak, and Aniket Kate. 2024. Attacking and Improving the Tor Directory Protocol. 3221–3237. doi:10.1109/SP54263.2024.00083

[55] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors over Rings. 1–23. doi:10.1007/978-3-642-13190-5_1

[56] Varun Madathil, Alessandra Scafuro, István András Seres, Omer Shlomovits, and Denis Varlakov. 2022. Private Signaling. In *31st USENIX Security Symposium (USENIX Security 22)*.

[57] M. Mitzenmacher. 2001. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems* 12, 10 (2001), 1094–1104. doi:10.1109/71.963420

[58] Christian Mouchet, Juan Ramón Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. 2021. Multiparty Homomorphic Encryption from Ring-Learning-with-Errors. 2021, 4 (Oct. 2021), 291–311. doi:10.2478/popets-2021-0071

[59] Muhammad Haris Mughees, Hao Chen, and Ling Ren. 2021. OnionPIR: Response Efficient Single-Server PIR. 2292–2306. doi:10.1145/3460120.3485381

[60] Pratyay Mukherjee and Daniel Wichs. 2016. Two Round Multiparty Computation via Multi-key FHE. 735–763. doi:10.1007/978-3-662-49896-5_26

[61] Jeongeun Park. 2021. Homomorphic Encryption for Multiple Users With Less Communications. *IEEE Access* 9 (2021), 135915–135926. doi:10.1109/ACCESS.2021.3117029

[62] Jeongeun Park and Sergi Rovira. 2023. Efficient TFHE Bootstrapping in the Multiparty Setting. *IEEE Access* 11 (2023), 118625–118638. doi:10.1109/ACCESS.2023.3326237

[63] Jeongeun Park and Mehdi Tibouchi. 2020. SHECS-PIR: Somewhat Homomorphic Encryption-Based Compact and Scalable Private Information Retrieval. 86–106. doi:10.1007/978-3-030-59013-0_5

[64] Ania M Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The loopix anonymity system. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 1199–1216.

[65] Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)* 56, 6 (2009), 34.

[66] Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM* 56, 6, Article 34 (Sept. 2009), 40 pages. doi:10.1145/1568318.1568324

[67] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. 2017. P2P Mixing and Unlinkable Bitcoin Transactions.

[68] Sajin Sasy and Ian Goldberg. 2024. SoK: Metadata-protecting communication systems. *Proceedings on Privacy Enhancing Technologies* (2024).

[69] Elaine Shi and Ke Wu. 2021. Non-Interactive Anonymous Router. 489–520. doi:10.1007/978-3-030-77883-5_17

[70] Fatemeh Shirazi, Milivoj Simeonovski, Muhammad Rizwan Asghar, Michael Backes, and Claudia Diaz. 2018. A Survey on Routing in Anonymous Communication Protocols. *ACM Comput. Surv.* 51, 3, Article 51 (June 2018), 39 pages. doi:10.1145/3182658

[71] TFHE developers. 2026. spqlios-arithmetic. GitHub repository.

[72] Adithya Vadapalli, Kyle Storrier, and Ryan Henry. 2022. Sabre: Sender-Anonymous Messaging with Fast Audits. 1953–1970. doi:10.1109/SP46214.2022.9833601

[73] Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. 2023. Verifiable Fully Homomorphic Encryption. arXiv:2301.07041 [cs.CR] https://arxiv.org/abs/2301.07041

[74] Ruida Wang, Yundi Wen, Zhihao Li, Xianhui Lu, Benqiang Wei, Kun Liu, and Kunpeng Wang. 2024. Circuit Bootstrapping: Faster and Smaller. 342–372. doi:10.1007/978-3-031-58723-8_12

[75] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. 2012. Dissent in Numbers: Making Strong Anonymity Scale. In *10th USENIX OSDI'12*. 179–182.

[76] Zama. 2026. FHEVM: A cross-chain protocol for confidential smart contract. https://github.com/zama-ai/fhevm/blob/main/fhevm-whitepaper.pdf.

[77] Zama. 2026. Zama Protocol - The Key Management Service (KMS). Design doc (webpage). https://docs.zama.org/protocol/protocol/overview/kms

A Open Science

We provide the implementation artifact for evaluating the core experimental contributions of this paper. The artifact consists of

Algorithm 2 Homomorphic placing HomPlacing for one slot

Input: An encrypted value V_r, and $\log \eta$ encrypted bits $\{c_0, \ldots, c_{L-1}\}$, where $L = \log \eta$.
Output: a vector $\mathbf{l}$ consisting of η values of leaves: $z_0, \ldots, z_{\eta-1}$, where $\mathbf{l}[i] = z_i$ for $i \in \{0, \ldots \eta - 1\}$
$\mathbf{l} := \mathbf{0}$
Initialize root: $b_0 = V_r$
Initialize the node values: $b_i := 0$ for $i \in \{1, \ldots, 2\eta - 2\}$
for $i \leftarrow 0, \ldots, L-1$ **do**
    for $j \leftarrow 0, \ldots, 2^i - 1$ **do**
        $b_{2^{i+1}+2j-1} := b_{2^i-1+j} - b_{2^i-1+j} \cdot c_i$
        $b_{2^{i+1}+2j} := b_{2^i-1+j} \cdot c_i$
    end for
end for
for $i \leftarrow 0, \ldots, \eta - 1$ **do**
    $z_i := b_{2^{L+1}-1+i}$
    $\mathbf{l}[i] = z_i$
end for
return $\mathbf{l}$

the proof-of-concept implementation of sPAR, including the source code, header files, test driver, and documentation needed to compile and run the experiments. All sources and documentation are available at the following GitHub repository:

https://github.com/HyewonSung/sPAR

The README file in the repository contains detailed instructions for how to run one round of sPAR for various numbers of clients $\eta \in \{20, 40, 60, 80, 100\}$ in a Linux environment. Running these commands allows readers to reproduce the experimental results reported in Tables 4 and 5.

B Ethical Considerations

The paper improves anonymous communication systems. Although there are debates about the ethical use of anonymous communication systems, academic research on anonymous communication itself is not considered by the community to have any ethical issues. The work does not involve any human subjects or user data. The work does not analyze/expose vulnerabilities of already deployed systems. To the best of our knowledge, this work does not raise any ethical concerns.

C Additional details

C.1 Homomorphic Placing

To reduce the clients' communication cost, each client sends a single RLWE ciphertext per message, encoding the index in its bit representation. With this approach, the upload complexity per client is reduced from $O(\eta)$ to $O(\log \eta)$.

The server then executes a homomorphic expansion algorithm to transform the ciphertext into an encrypted array, where only the designated position contains the corresponding message. This additional step incurs $O(\eta^2)$ homomorphic computations (or $O(\eta)$ time with η parallel cores).

We adopt homomorphic traversal algorithm introduced in [21], to place an item to the designated component of the output vector homomorphically, while the rest components are encryptions of 0. We introduce the algorithm adapted to our protocol (See Algorithm 2).

The algorithm starts with the root of an empty but complete tree of depth $L := \log \eta$ with η leaves. It takes an encrypted value V_r and $\log \eta$ encrypted bits $\{c_0, \ldots, c_{s-1}\}$. Then V_r is put to the root of the tree, while other nodes are filled with 0 (remaining empty).

The encrypted $\log \eta$ bits indicates the address of the position where the other input value (denoted by V_r) will be travelled to. For example, if the input bits are a bit representation of 3, then the input value V_r will be copied to the third leaf from the left (we count from 0). To do this, each node should evaluate an operation to keep all the values oblivious to the computing party, therefore the computation complexity is $O(\eta)$. Each computation of each left/right child node is performed via homomorphic operation, therefore the $\cdot$ denotes homomorphic multiplication. In our instantiation, we use external product to keep practical computation cost. We will use Algorithm 2 as a part of our anonymous router construction in the next section. When it is applied to our construction, we let clients choose multiple slots (indices) for their message to be placed to avoid collision. If the number of slots is the same as the number of clients, there is a high change for at least two clients to choose the same slot since they do not communicate to choose the slots. Therefore, we increase the size of slots by introducing additional parameter, so that we can guarantee the possibility of having collision between any two parties is negligible.

As a result, in our adapted algorithm, clients prepare multiple indices for such slots, there are additional loops for placing the value to the only one slot (see Algorithm 1).