Verifiable Self-Mix

Doron Zarchy

APSIA, SnT, University of Luxembourg, Luxembourg
doronz@gmail.com

Abstract. Anonymous communication systems aim to hide which user sent which message, but existing designs sit at two extremes: efficient, centralized mixnets that rely on at least one honest mix server, and fully decentralized protocols such as DC-nets and MPC-based shuffles that avoid trusted mixers at the cost of high communication and fragility under disruption. This paper introduces *verifiable self-mix* (VSM), a new anonymity architecture for privately shuffling messages posted on a public bulletin board. In VSM, users *self-mix* their encrypted messages by writing them to secret locations in a public table. A preprocessing protocol, *Unique Number Selection* (UNS), assigns every user a distinct hidden mix location; an anonymization protocol, *Secure Mapping of Private Permutation* (SMPP), ensures that each such location contains exactly that user's ciphertext, without revealing the mapping from users to locations. Because each user learns their own final location, VSM provides *unconditional individual verifiability*: after an optional decryption step, users can check that their message appears at the correct slot without running any cryptographic checks. We formally define VSM and prove anonymity, integrity, and self-verifiability in the malicious model. We then present several concrete instantiations. Our UNS protocols rely on either trusted hardware or multiple servers running plaintext-equivalence tests and require only constant private memory, making them suitable for constrained trusted execution environments. Our SMPP constructions allow trade-offs between trust and bandwidth, achieving $O(n)$ ciphertext uploads per user (or $O(\log n)$ in theory under FHE, at substantial setup cost), while avoiding the pairwise shared-pad state and interactive disruption-resolution typical of classic DC-net designs.

1 Introduction

Anonymous communication networks (ACNs) aim to protect message senders' identities against a powerful network observer. Mixnets are the most widely used ACN primitive: a set of mix servers repeatedly permute and re-encrypt batches of ciphertexts so that an outside adversary cannot link individual inputs to outputs [13, 30, 46, 52, 1, 56]. While mixnets are efficient and well-understood, they are typically centralized (C-ACNs): anonymity relies on the assumption that at least one mix server in the chain does not collude with the adversary. This assumption is challenging to justify in practice, especially when mixes are operated by a single organization or deployed in low-assurance environments, as illustrated by the Norwegian trial Internet voting system [31].

Trusted hardware (TH), such as Intel SGX, is often proposed as a way to harden a single mixing server: cryptographic state is kept inside an isolated enclave that can be remotely attested. However, current trusted execution environments (TEEs) offer only limited private memory (e.g., SGX enclaves expose about 128 MB of protected memory and incur substantial overhead when paging) [55]. Mixnets conceptually hold and shuffle entire batches of ciphertexts, so they tend to require substantial memory for global permutations. This tension makes it difficult to deploy large-scale mix-based ACNs or e-voting systems directly inside TEEs.

A different line of work removes the trusted mixer entirely and instead builds fully decentralized ACNs (D-ACNs), often via secure multi-party computation (MPC). The classic example is the dining cryptographers network (DC-net) [11], along with many jamming-resistant variants and MPC-based anonymous broadcast systems. While these protocols do not assume a trusted third party, they have well-known drawbacks: the number of rounds often grows with the number of participants or adversaries, honest users may send $\Omega(n)$–$\Omega(n^2)$ messages per round, and even a single malicious participant can disrupt the protocol unless heavy accountability mechanisms and zero-knowledge proofs are added [43]. Multi-party sorting-based D-ACNs can improve round complexity [43], but typically require a strong honest-majority assumption among the participants.

Our goal in this work is to explore a new point in this design space: we seek an ACN architecture that (i) keeps the trusted component small enough to realistically live in a TEE or a small set of threshold servers, (ii) gives users a simple, non-cryptographic way to verify that their messages were included correctly, and (iii) achieves better per-user scalability than DC-net-style protocols, while making more modest trust assumptions than classic mixnets.

Verifiable Self-Mix (VSM). We introduce *verifiable self-mix* (VSM), a semi-centralized ACN architecture for privately shuffling messages over a public bulletin board. Instead of having a mix server permute an entire batch, VSM lets each user "self-mix" their encrypted message by writing it to a *secret* location in a public table. Intuitively, the system assigns each user an anonymous "slot" in the table, and users then place their ciphertexts into their own slots without revealing which slot they control.

VSM consists of two generic modules:

- **Unique Number Selection (UNS).** A preprocessing protocol that runs once to assign every user a distinct *mix location* λ_i in an enlarged index space. The value λ_i is hidden from all other parties but learned by user i. Our UNS constructions use either a TEE with a misuse-detection mechanism, or a set of servers that jointly run plaintext-equivalence tests (PETs); in both cases, the trusted component operates only on encrypted indices and needs $O(1)$ private memory, making it suitable for constrained trusted hardware.
- **Secure Mapping of Private Permutation (SMPP).** Given encrypted pairs $(\lambda_i, E(m_i))$ from n users, SMPP produces a public table T such that, for each user i, the entry at position λ_i is an encryption of m_i, and all other entries contain encryptions of dummy values. We present SMPP constructions

based on ElGamal and BGN, using either non-interactive zero-knowledge proofs (NIZKs) or homomorphic checks to ensure that no one learns the mapping from users to locations.

A key property of VSM is that *each user knows their own final location.* If we additionally decrypt the entries of T (SMPP-Dec), every user can *unconditionally* verify that their message appears in the correct slot λ_i simply by checking the published table—no cryptographic tools or hardness assumptions are required on the user side. This transparent plaintext-lookup style of individual verification is related to tracker-based verifiable voting systems such as sElect, Selene, and Hyperion [58, 59, 57]. VSM does not claim receipt-freeness or coercion resistance. Its distinction is architectural: it targets anonymous communication over a bulletin board, separates slot assignment from anonymous placement, avoids a mixnet in the placement path, and makes aggregation a deterministic public function of posted ciphertexts.

Contributions. In summary, this paper makes the following contributions:

- **A formal VSM architecture.** We define verifiable self-mix (VSM) as the composition of UNS and SMPP, and give security definitions and proofs for anonymity, integrity, and self-verifiability in the malicious model.
- **TEE-friendly unique number selection.** We design UNS protocols that assign unique encrypted locations to users while using only constant private memory on the trusted component. We show how UNS can be instantiated either in a TEE with misuse detection, or by multiple servers running PET with a threshold honest-server assumption.
- **Secure mapping of private permutations.** We introduce SMPP as a reusable primitive for anonymously mapping encrypted messages to encrypted locations, and present two concrete instantiations: (i) an ElGamal-based construction that uses NIZK proofs to verify well-formed inputs, and (ii) a BGN-based construction that replaces proofs with homomorphic checks. We further sketch an FHE-based variant, using homomorphic traversal techniques, that reduces per-user bandwidth to $O(\log n)$ in theory, at substantial setup cost.
- **Analysis and comparison with existing ACNs.** We prove that VSM achieves anonymity and integrity under standard cryptographic assumptions, each user can confirm correctness by checking that their message appears at their secret slot, without performing any cryptographic checks or trusting any server. We compare VSM to mixnets, DC-net-style protocols, and anonymous broadcast systems in terms of rounds, bandwidth, and trust. VSM sits between centralized mixnets and fully decentralized MPC-based networks: it uses $O(n)$ per-user communication (or $O(\log n)$ in theory under FHE, at substantial setup cost), versus the $O(n^2)$ cost of classic DC-nets, while keeping the trusted component much smaller than in mix-based designs and relying on standard bulletin-board abstractions.

2 Related Work

Mixnet can be traced back to the pioneering work of Chaum [13] which was based on a decryption mix. Park et al. were the first to propose a construction based on homomorphic properties of re-encryption [47]. It was further developed by [30, 46, 52, 1]. A universally verifiable, based on zero-knowledge shuffle proofs was given by Neff [45] and Groth [34]. Wikstrom proposed a new method for mixnets based on partial decryption and proved its security via universal-composability framework [56]. Adida et al. and Wikstrom proposed methods for public shuffling where the permutation is computed before the actual shuffle [3, 4]. Chaum et al. proposed cMix, a mixnet variant that minimizes real-time public-key operations by moving most expensive work to preprocessing [12].

A major line of work on ACNs starts with DC-nets [11], and subsequent works mostly focus on detecting and preventing cheating parties from jamming the broadcasts [33, 18, 51]. Dunning and Kresman propose a novel technique for hiding messages using Newton identities [24], though the scheme is secure only in the honest but curious setting. DiceMix [51] adapts the Newton-identity technique of Dunning and Kresman [24] to a peer-to-peer mixing setting and achieves security in the malicious model. Many DC-net style protocols, including Dissent and DiceMix, require a number of communication rounds that grows linearly with the number of disrupted messages or misbehaving users in order to identify jammers [18, 51]. More recent systems such as PriFi and OrgAn achieve constant-round anonymity by adding additional trust assumptions (e.g., access points or coordinators) and precomputation phases [7, 22].

Another approach is based multi-party sorting where the sequence of comparisons (for sorting) is set in advance, regardless of the outcome of previous comparisons [5, 43, 41, 10, 35]. These schemes are based on data-oblivious algorithms and verifiable secret-sharing schemes (VSS). These protocols are efficient but assume that the number of adversaries has to be less than one-third of the parties [43, 41, 10].

On evaluation scope. End-to-end latency/throughput numbers are difficult to compare across anonymous communication systems because performance depends strongly on system-level assumptions (batching strategy, cover-traffic rates, network latency, churn and fault model, and deployment architecture) in addition to the cryptographic core [21]. For example, Loopix targets low-latency messaging with cover traffic and Poisson mixing delays [49], while PriFi and OrgAn target organizational networks using a client/relay/server architecture derived from DC-net techniques [7, 22]. In contrast, Riposte and Express target high-latency anonymous broadcast / write-only databases with a small set of non-colluding servers [17, 26], and Atom and Trellis focus on mix-based batch anonymity at scale [38, 39]. Accordingly, we do not attempt a head-to-head evaluation of VSM against full end-to-end systems. Instead, we report an analytic cost breakdown of VSM's modules (UNS and SMPP) in terms of (i) client upload to the bulletin board, (ii) bulletin-board storage, and (iii) dominant cryptographic operations, which is the level at which different VSM instantiations can be compared fairly.

Table 1. High-level comparison of representative ACN families and VSM variants. The table intentionally abstracts away bulletin-board realization, churn, cover traffic, and deployment-specific engineering choices; the goal is to compare the cryptographic core at a coarse level only.

System family	Online client upload	Online interaction rounds	Additional tr
Classic mixnets	compact ciphertext(s)	mix cascade / batching	at least one h
DC-net style systems	protocol-dependent, often $\Omega(n)$	protocol-dependent	depends on disruption
VSM (TEE-UNS)	$O(m)$ for SMPP	1 bulletin-board round after UNS	one trusted TEE (or
VSM (PET-UNS)	$O(m)$ for SMPP	1 bulletin-board round after UNS	at least one honest serv
VSM (FHE-SMPP)	$\tilde{O}(\log m)$ in theory	1 bulletin-board round after UNS	same as the chosen UNS insta

Table 1 should be read only as a high-level orientation; the more detailed module-level cost breakdown for VSM appears later in Table 3.

Multi-server anonymous broadcast and metadata-hiding. Several recent systems implement anonymous broadcast or metadata-hiding communication by having clients write into a shared database maintained by multiple servers. Riposte uses distributed point functions to realize an anonymous write-only database supporting millions of users [17]. Atom is a horizontally scalable mix-based system that combines anytrust[1] server groups with multi-stage routing and layered mixing to resist traffic analysis [38]. Express reduces client-side costs for metadata-hiding messaging by using two non-colluding servers and compact DPF-based writes [26]. Trellis [39] is a mix-net based anonymous broadcast system in which servers iteratively mix messages and post them to a public bulletin board. Blinder [2] studies anonymous committed broadcast and proposes an MPC-based protocol with robustness/availability against active adversaries. Clarion builds anonymous communication from generic multiparty shuffling protocols [25]. Compared to these systems, VSM also assumes an authenticated, append-only bulletin board and a small set of servers, but differs in that (i) users "self-mix" into secret slots rather than writing secret shares into a shared database, and (ii) each user learns their own final location, yielding unconditional individual verifiability.

Low-latency mixnets and DC-net-style group communication. As concrete examples of the deployment regimes mentioned above, Loopix [49] is a low-latency mixnet that uses cover traffic and Poisson mixing delays to resist traffic analysis. Dissent and Verdict provide accountable anonymous *group* messaging built on DC-nets (with Verdict using proactively verifiable DC-nets) [18, 19]. PriFi and OrgAn adapt DC-net-style anonymous communication to organizational networks using a client/relay/server architecture [7, 22].

[1] An *anytrust* assumption means that security holds as long as at least one server in a designated set behaves honestly; this is the same assumption made by VSM's multi-server UNS instantiation.

3 System Overview

VSM is a new protocol that provides anonymity to the users.

Suppose that n users have private input and wish to write on the public database while preserving their anonymity. In contrast to mixnet, where the server (or mix) shuffles the messages, here each user "shuffles" its own message according to a simple idea: each user selects a distinct location in a k-entry table and writes the message in this distinct location. We say that a user *self-mixes* when it autonomously selects and occupies a slot in the output table, as opposed to a server-controlled mix where the server assigns and populates slots on the user's behalf. There are two challenges regarding this approach. First, there may be collisions, i.e., when more than one user selects the same location. Second, the users may lose their anonymity when performing the actual writing on the bulletin board. Practical instantiations of the bulletin board include public ledgers, server-maintained append-only logs with digital signatures, or transparency-log infrastructure; the cost of maintaining this abstraction is outside our scope. VSM addressed these two challenges.

The general architecture of VSM is built on two decentralized modules (i.e., sub-protocols): unique number selection (UNS) and Secure Mapping of Private Permutation (SMPP).

1. UNS is a preprocessing phase executed once per *epoch* before the users send a batch of messages. The goal of UNS is to assign a unique number to each user. These numbers are called mix-locations and should be kept private. One mode of UNS is suitable for a trusted execution environment and the other is designed to be executed by multiple parties.
2. SMPP is an anonymization module that allows users to "transfer" the encrypted message into a specific entry, which is indexed by the mix location obtained by UNS, to a public database (abstracted as a bulletin board) without being detected. We provide two instantiations based on two encryption schemes. The first is ElGamal-based, which does not depend on any trusted setup beyond the bulletin board and threshold decryption, though it requires relatively more expensive zero-knowledge proofs; the second is BGN-based, whose privacy depends on an honest-majority setup MPC, though it avoids per-user NIZKs.

If the same mix-locations are reused across multiple message batches, the resulting outputs remain anonymous with respect to user identity but become linkable to the same pseudonymous slot. Formal unlinkability across epochs therefore requires rerunning UNS for each epoch.

4 Preliminaries

We use the notation $[n] = \{1, \ldots, n\}$ and write $\hat{x} := E(x)$ for an encryption of x under the relevant public key. We extend this notation entry-wise to vectors, i.e., for $v = (v_1, \ldots, v_m)$ we write $\hat{v} := (E(v_1), \ldots, E(v_m))$. All communication

to the bulletin board is authenticated, and the bulletin board is assumed public and append-only.

We write $\tilde{O}(f)$ to suppress polylogarithmic factors in f and in the security parameter.

Our instantiations rely on standard public-key primitives: (threshold) ElGamal encryption for additively homomorphic operations and rerandomization (Appendix B.1 and Appendix E.1), the Boneh–Goh–Nissim (BGN) cryptosystem for one multiplication via pairings (Appendix C), and non-interactive zero-knowledge proofs for well-formedness of ciphertext vectors (Appendix D). When required, the servers also execute a plaintext-equivalence test (PET) over threshold ElGamal ciphertexts (Appendix E).

5 Unique Number Selection

Unique Number Selection (UNS) is the first phase of VSM, which assigns unique numbers (also called a mix-integer) to the users. To improve efficiency and reduce collisions, the location space size is set to be larger than the number of users , namely $m = a \cdot n$ for some pre-selected $a > 1$, from which the users select mix-integers. To this end, each user i selects $k = O(log\ n)$ integers and encrypts them with the server public key pk (where in case of multi servers, pk can be generated by trusted hardware, or by some distributed key generation protocol, and shares of the associated private key is distributed amongst the servers), with signed ElGamal (which is CCA encryption scheme, [53]) that result in the input vector $\hat{P}_i := (E_{pk}(l_{i0}), E_{pk}(l_{i1}), \ldots, E_{pk}(l_{ik}))$. Therefore, UNS receives as input $\hat{P} := (\hat{P}_1, ..., \hat{P}_n)$, where $\hat{P}_i$ is a list of encryption of integers of user i, and outputs $(E_{pk}(\lambda_1), ..., E_{pk}(\lambda_n))$ where $\lambda_1, ..., \lambda_n$ are integers assigned to the users, such that $\lambda_i \neq \lambda_j$, for any $i \neq j$. Each surviving user i recovers λ_i in plaintext as the candidate ℓ_{i,t_i} it originally submitted, identified by observing which of its posted ciphertexts appears in the public UNS output.

At the time of execution, UNS keeps a record of the *current* "candidate" of integers, which are the items in the list that are being tested for collisions. UNS sequentially picks pairs of two current integer candidates and tests if there is a collision, i.e., if they are both encryptions of the same plaintext. If there is a collision, then UNS updates the current candidate to the subsequent encrypted integer in the list. If UNS "consumes" a user's preference list, then this user is removed (since there is a high chance that the user is malicious, see Lemma 2). This process repeats until no collision occurs or until all dishonest users, who may intentionally attempt to incur collisions, are removed. We present here UNS that applies both to trusted hardware and to multiple untrusted parties that cooperatively execute Plaintext Equivalence Tests [15].

We first formalize UNS. Let $\hat{P}_i$ be a list of encryption of integers (field elements) of user i. UNS is a protocol that assigns distinct locations. It receives as input $\hat{P} := (\hat{P}_1, \ldots, \hat{P}_n)$, where pk is a public key generated by a trusted party (e.g., by trusted hardware or by a distributed key-generation protocol), and outputs $(E_{pk}(\lambda_1), \ldots, E_{pk}(\lambda_n))$ where $\lambda_1, ..., \lambda_n$ are integers assigned to the users,

such that $\lambda_i \neq \lambda_j$, for any $i \neq j$. $E(\lambda_i)$ is an ElGamal encryption of λ_i (details are in Appendix B.1). That is, UNS receives as input a list of ElGamal encryptions from each user and outputs to each user an encryption of an integer. To this end, each user i selects k integers and encrypts them with the server public key (for which its associated private key is distributed amongst the servers, or maintained by a trusted hardware), with a CCA encryption scheme (signed ElGamal, see Appendix B.2) which result in the vector $\hat{P}_i := (E(l_{i0}), E(l_{i1}), ..., E(l_{ik}))$.

UNS parses the preference list of all users and tests each time a pair of elements in the list for collision. We assume that the servers which execute UNS verify the users' signatures. At the time of execution, UNS keeps a record of the *current* "candidate" of integers, which is the item in the list that is being tested. UNS sequentially picks pairs of two current integer candidates and tests if there is a collision, i.e., if they are both encryptions of the same plaintext. If there is a collision, then UNS updates the current candidate to the subsequent encrypted integer in the list. If UNS "consumes" a user's preference list, then this user is removed (since there is a high chance that the user is malicious, see Lemma 2).

This process repeats until no collision occurs or if all users are removed (which, as mentioned, is unlikely to happen if there are honest users). We present here UNS that is applicable both to trusted hardware and to multiple untrusted parties that cooperatively execute Plaintext Equivalence Tests (see Protocol E in Appendix E). We now provide some notations. Let $UNS^{(i)}(\hat{P})$ be the integer assigned to user i. We denote $E(\lambda_i)$ by $\hat{\lambda}_i$. Let $\hat{P}_i[t]$ denote the t-th element in $\hat{P}_i$ and let SI be a list that consists of the indices that points to the current integer candidates of the users. Let S be the list of encrypted current candidates of integers, i.e., $S = [\hat{P}_1[SI[1]], ..., \hat{P}_n[SI[n]]]$ and let $S_A \subseteq S$ be a list of all the "fresh" current candidates of integers that should be checked for collision with *all* elements in S. In module 1, we use S_A as a stack. We emphasize that UNS module, which is executed over an append-only bulletin board, assumes that communication is authenticated. We now show an example that illustrates the execution of UNS with three users. *Let $U = \{A, B, C\}$ be three users who select integers from the set $\{1, 2, 3\}$, and let $\hat{P}_A = [E(1), E(3), E(2)]$, $\hat{P}_B = [E(2), E(3), E(3)]$, and $\hat{P}_C = [E(2), E(1), E(1)]$ be their encrypted preference lists.* Initially the current candidates are $s_A = E(1)$, $s_B = E(2)$, and $s_C = E(2)$. The only collision is between B and C, so their pointers advance and the next candidates become $s_B = E(3)$ and $s_C = E(1)$, while A remains at $E(1)$. Now A and C collide, so A advances to its next candidate $E(3)$ and C advances to its next candidate $E(1)$. Finally, A and B collide on $E(3)$, so A advances once more to $E(2)$ while B stays at $E(3)$. At this point the current candidates are pairwise distinct and the algorithm outputs $S_A = E(2)$, $S_B = E(3)$, and $S_C = E(1)$.

5.1 Performance of UNS

We now analyze the expected running time of UNS that is determined by the number of TC invocations used for collision testing. We also analyze the worst-case running time when an adversary controlling some of the users attempts to maximize the running time.

Protocol 1 Unique Mix Integers $(\hat{P}_1, ..., \hat{P}_n)$

The servers do the following:

1. For each user i, parse the lists $\hat{P}_i$ and push it to a stack S_i
2. Initialize $SI = [1, ..., 1] \in \mathbb{Z}^n$, and $S = \bigcup_{i\in[n]} \hat{P}_i[1]$
3. Initialize $S_A \leftarrow S$
4. While $S_A \neq \emptyset$:
   (a) Pop s_i from S_A
   (b) For all $s_j \in S \backslash s_i$, the servers compute $O = TC(s_i, s_j)$
   - If $O = 1$ (a collision is found) then
     i. set $SI[i] \leftarrow SI[i] + 1$, $S[i] \leftarrow \hat{P}_i[SI[i]]$, and push $S[i]$ to SA
     ii. set $SI[j] \leftarrow SI[j] + 1$, $S[j] \leftarrow \hat{P}_j[SI[j]]$, and push $S[j]$ to SA
     iii. For $t \in \{i, j\}$, if $SI[t] > log(n)$ remove user t (delete t,$S[t]$,$SI[t]$, and from S_A)

   If $SI = \emptyset$ return FAIL

return $(S[SI[1]], \ldots, S[SI[n]])$ together with the surviving index vector SI; equivalently, each surviving user i identifies the unique output ciphertext that matches one of its posted candidates and locally recovers $\lambda_i = \ell_{i,SI[i]}$ as the corresponding plaintext value.

Lemma 1. *The expected total number of TC invocations in protocol 1 is at most* $n\frac{a}{a-2}$.

Proof. To simplify the analysis we represent this process as a stochastic ball into bins problem where we have a bag full of balls and we throw the ball to one of the bins at random (which is the analog of checking if the mix integer is colliding with any other users' mix integer). If the bin is empty then the ball stays in the bin (i.e., representation no collision occurs). However, if the bin is occupied by another ball then we take both balls out of the bin and put them back in the bag (i.e., in case of collision the servers update both users mixed integers). This process proceeds until the bag is empty. As noted above, to prevent an exponential number of balls, we expand the set of mix integers from $[1, n]$ to $[1, an]$. In each draw, the probability of adding one ball is no more than $\frac{1}{a}$, (the probability of collision is $\frac{1}{a}$ when all balls occupy a bin). Therefore the probability of removing one ball (representing the case of no collision) from the bag is at least $\frac{a-1}{a}$. Therefore, we can treat this process as a biased random walk where a random walker starts at a point located n steps from the destination, where the probability of taking one step towards the destination is $\frac{a-1}{a}$ and the probability of taking one step backward from the destination is $\frac{1}{a}$. Since the expected number of balls decreases each time by at least $\frac{a-1}{a} - \frac{1}{a} = \frac{a-2}{a}$,(or the random walker moves towards the destination by a fraction of $n\frac{a-2}{a}$ in expectation). Since there are at most $\frac{n}{a}$ occupied bins and at least $n\frac{a-1}{a}$ are unoccupied when drawing a new ball, the probability of a new collision, which implies returning the ball into the bag, is at most $\frac{1}{a}$ and the probability of having no collisions, which implies removing a ball from the bag, is at least $\frac{a-1}{a}$. Let x_i be the number of balls in the bag in round i, i.e., $P(x_i = x_{i-1} + 1) \geq p$ where $p = \frac{a-1}{a}$ and $P(x_i = x_{i+1} - 1) \leq q$ where $q = \frac{1}{a}$. We want to lower bound the expected number

of balls until the bag becomes empty. Therefore Let $\delta_i = x_i - x_{i-1}$ and let T be the number of rounds. Then $E(\delta_i) \geq p - q = \frac{a-2}{a}$ Note that $x_i = \sum_1^T \delta_i$. Then, $E(x_i) = E(\sum_1^{E(T)} \delta_i) = \sum_1^{E(T)} E(\delta_i) \geq E(T)\frac{a-2}{a}$. Since $x_i = n$ when the bag becomes empty, we have $E(T)\frac{a-2}{a} \leq n$ and therefore $E(T) \leq n\frac{a}{a-2}$.

Note that according to Lemma 1, if we set $a = 4$, the expected total number of TC invocations is at most $2n$. In the worst case, corrupted users can force at most $K \cdot |\mathsf{corrupted}|$ additional tests before their lists are exhausted and they are removed; after that point the remaining honest users complete in the expected time given by Lemma 1.

5.2 Handling Denial of Service

If the lists of the users had not been limited by size, then two colluding users could have submitted two encrypted lists for which their underlying integers are identical, which would incur a very large number of collisions and even prevent the protocol from terminating. Therefore we limit the size of the encrypted list to $log(n)$ elements. Therefore, UNS tests no more than $log(n)$ items for collisions. If a user's list is consumed then the user is tagged as malicious and removed from the process. This limitation raises a concern that honest users will also be misidentified as malicious. The following lemma guarantees that the probability that such an event can occur is small. More precisely, it upper bounds the chances that protocol 1 consumes one of the honest user's lists in case the user's preference list has a size of $log(n)$.

Lemma 2. *Suppose that then expansion factor is at least 4 (i.e., $m \geq 4n$). If all parties submit a list of size $log(n)$, then the probability that protocol 1 consumes an entire list of at least one honest user is at most $\frac{n}{e}\left(\frac{e}{log(n)}\right)^{log(n)}$.*

Proof. Let $K = \lceil \log n \rceil$ be the list size, and fix an honest user i with submitted candidates $\ell_{i,1}, \ldots, \ell_{i,K}$. For $t \in [K]$, let C_t be the event that the t-th candidate of user i collides when it becomes active.

Fix t and condition on any protocol history up to the moment that $\ell_{i,t}$ is checked. At that time, there are at most $n - 1$ active candidates belonging to the other users. Let $A_t \subseteq [m]$ denote the set of those active values. Since the preference lists are encrypted under an IND-CCA scheme before UNS begins, a computationally bounded adversary cannot choose A_t as a function of the honest plaintext candidate $\ell_{i,t}$; thus the required independence is justified. As $\ell_{i,t}$ was sampled independently and uniformly from $[m]$, we have

$$\Pr[C_t \mid \text{history}] = \Pr[\ell_{i,t} \in A_t \mid \text{history}] \leq \frac{|A_t|}{m} \leq \frac{n-1}{m} \leq \frac{1}{4},$$

where the last inequality uses $m \geq 4n$.

For user i to be removed, all K candidates must collide. Hence, by the chain rule,

$$\Pr[\text{user } i \text{ is removed}] = \Pr\left[\bigwedge_{t=1}^{K} C_t\right] = \prod_{t=1}^{K} \Pr[C_t \mid C_1, \ldots, C_{t-1}] \leq \left(\frac{1}{4}\right)^K.$$

Applying the union bound over the at most n honest users gives

$$\Pr[\exists \text{ honest user removed}] \leq n\left(\frac{1}{4}\right)^K.$$

For $K = \lceil \log n \rceil$, this is negligible in n.

Throughout the paper we set the expansion factor to $a = 4$, which satisfies both the $a > 2$ condition of Lemma 1 and the $m \geq 4n$ condition of Lemma 2. With this choice, Lemma 1 gives an expected total of at most $2n$ TC invocations. To capture intuition, the chance for such misclassification of an honest user as dishonest, for $n = 1000$, is at most $\frac{1}{439{,}000}$. In terms of anonymity, Protocol 1 is designed to allocate unique mix integers for the users while maintaining anonymity.

Moreover, by UNS anonymity (Definition 2), the collision feedback reveals no information about honest users' underlying candidates beyond whether a collision occurred; hence a malicious coalition cannot steer collisions toward a specific honest target.

5.3 Testing Collision

The testing collision operation (TC) (algorithm 1), can be instantiated in multiple ways, where each makes different assumptions on the trusted parties for retaining privacy. Since TC is assumed to be operated over an authenticated bulletin board, it ensures that if one of the parties does not follow the protocol then it can be detected. Thus, we assume that the parties are following the TC module.

Testing Collision via Trusted Hardware. In our model, we assume that the host of the trusted hardware (TH) is dishonest. Therefore, there should be a mechanism that enforces TH to perform a computation only according to the program installed in the TH. To this end, an attestation to the program should be published by the TH [37, 36]. We first describe how to perform TC with a single TH. Since the TH is the sole owner of the decryption key, it can decrypt both ciphertexts and test if the underlying plaintexts are equal. However, since the TH host may be malicious, it can feed the TH with any pair of ciphertexts encrypted with the TH public key. Hence, the host can completely break the privacy of the users by encrypting each of the possible integers (from a small set $[m]$) and comparing each with the users' encrypted integer, for which the TH should output accept (equal plaintexts) or reject (not equal plaintexts).

TC-misuse detection. Algorithm 2) is a program run in the TH (e.g., on an isolated enclave) for checking collisions between pair ciphertexts. The TH first

run Setup for generating a pair of keys. The public key is expected to be used by the users to encrypt their integers before they are published to the bulletin board. Checking collisions is done by the TH host which sends the pair of ciphertexts (c_0, c_1) to the TH. In order to prevent the TC host from misusing the TH at any time during the execution of the protocol without being detected. , each time the TH host invokes $TC(c_1, c_2)$, the TH updates the state by hashing the pair of ciphertexts and the result of the collision test t, along with the previous state using a collision-resistant hash function H. Any output return by the TH is signed using the TH vendor secret key, denoted TH_{sk} (the TH vendor's public key is assumed to be available publically). When the execution of the UNS ends, the TH host is expected to send the "Terminate" command to the TH. When the TH receives this command it returns the last state of the computation and halts, namely, it ignores any further input commands from the TH. The reason for this functionality is to prevent the TH host from sending further unnecessary "TestCollisions" commands to TH. Since all operations (hashing, signing, and decrypting) in Algorithm 2 require $O(1)$ memory, and the elements that should be maintained require $O(1)$ amount of memory (the secret key sk, and the state S)

Protocol 2 Program Test-Collisions run by TH

On receiving input ("Setup")
1. Generate (sk, pk)
2. Set $S \leftarrow 0$
3. Compute $\sigma_{pk} = Sign(TH_{sk}, pk)$
4. Return (σ_{pk}, pk)

On input ("TestCollisions", c_0, c_1**)**
1. Compute $m_0 = D(sk, c_0)$
2. Compute $m_1 = D(sk, c_1)$
3. If $(m_0 = m_1)$, $t \leftarrow 1$, otherwise $t \leftarrow 0$
4. Compute $S = H(S, c_0, c_1, t)$
5. Compute $\sigma = Sign(TH_{sk}, S)$
6. Return (σ, S, c_0, c_1, t)

On input ("Terminate")
1. Compute $S = H(S, "TERMINATE")$
2. Compute $\sigma = Sign(S, TH_{sk})$
3. Return $(\sigma, S, "TERMINATE")$ and HALT

the TH host should send a termination command to the TC $finilizeTC$, for which the TH is programmed to do the following:

Lemma 3. *No TH host can gain any information from misusing TC.*

Proof. Since the order of the pairs of ciphertexts (c_1, c_2) that are tested for collisions invoked by UNS is deterministically determined based on the output t of the TH (i.e., if the TH finds collision or not), the output of the TC as

published on bulletin board can be publically verified. Therefore, if the TH host tests a collision through the TH that should not have been checked (at a current state), then the hash of the new state will not match the expected state. Since the hash function is collision-resistant, with overwhelming probability, the TH host cannot attack by sending requests to the TH to get the correct expected state. In addition, the public servers expect to receive a "Terminate" message in the end, with the expected state. Since the TC aborts any further message, the TH host cannot use the TH afterward to test the collision of any other pair.

TC Executed by Servers. Here, in order to compare two ciphertexts, k servers $S_1, \ldots, S_k$ engage in PET (Protocol E in Appendix E) to cooperatively test whether the two plaintexts are equal [15]. The server cooperatively engages in a distributed ElGamal key generation algorithm, where pk is the private key, and $x_1, ..., x_k$ is their share of their secret keys. Since all server can combine their shares of the secret key and hence decrypt the encryptions of the users' integer (and hence later break the anonymity of the users), we assume that at least one of the servers is honest. The output of PET is published on the bulletin board. Since PET is run in the public domain, any deviation from the protocol can be detected (and the dishonest user can be removed).

6 Self Mapping of Private Permutation

The goal here is to describe a protocol that allows a user to "shuffle a message" to distinct locations of a public table, without disclosing the identity of the user.

Definition 1. *A protocol is Self Mapping of Private Permutation (SMPP) if, given as input from n users a sequence of pairs $((\lambda_1, E(M_1)), \ldots, (\lambda_n, E(M_n)))$ with pairwise distinct slots $\lambda_1, \ldots, \lambda_n \in [m]$, SMPP outputs a table T such that for each $i \in [n]$, $T[\lambda_i] = E(M_i)$. The value λ_i is called the mix location of user i. In the composed VSM protocol, pairwise distinctness is guaranteed by UNS integrity.*

Note that we did not specify the content of all entries that are not mixed location of any user (i.e., $NL := [m] \setminus \cup_{j \in n} \lambda_j$ as it does not affect the anonymity of SMPP. Under our construction for any $j \in NL$, $T[j] = E(0)$. Despite the functionality of SMPP, as defined above, the goal of SMPP is to preserve anonymity, namely, when user i wishes to write $E(M_i)$ in a $T(\lambda_i)$, no honest (computationally bounded) user will be able detected who sent $E(M_i)$.

We now give an overview of the construction of our SMPP. The user creates two m-sized vectors: and V^{λ_i, m_i} such that

1. $L_i^{\lambda_i} := (L_{i1}, ..., L_{im})$ such that $L_{i,\lambda_i} = 1$, and for any $j \neq \lambda_i$, $L_{ij} = 0$. Then compute $\hat{L}_i^{\lambda_i} := (\hat{L}_{1j}, ..., \hat{L}_{mj})$, where $\hat{L}_{ij}$ denotes the encryption of L_{ij}, i.e., $\hat{L}_{ij} := E(L_{ij})$.
2. $V_i^{\lambda_i, m_i} := (V_{i1}, ..., V_{im})$, where $V_{ij} := E(V_{ij})$, such that $V_{i\lambda_i} = M_i$, where M_i is the user's message, and $\forall j \neq \lambda_i$, $V_{ij} = 1$. Then compute $\hat{V}_i^{\lambda_i, m_i} := (\hat{V}_{i1}, ..., \hat{V}_{im})$, where $\hat{V}_{ij} := E(V_{ij})$

Table 2. Matrices $\hat{L}$ and $\hat{V}$ for $n = 5, m = 5$

$\hat{L}$	1	2	3	4	5
$\hat{L}_1$	E(0)	E(0)	E(0)	E(1)	E(0)
$\hat{L}_2$	E(0)	E(0)	E(1)	E(0)	E(0)
$\hat{L}_3$	E(1)	E(0)	E(0)	E(0)	E(0)
$\hat{L}_4$	E(0)	E(1)	E(0)	E(0)	E(0)
$\hat{L}_5$	E(0)	E(0)	E(0)	E(0)	E(1)

$\hat{V}$	1	2	3	4	5
$\hat{V}_1$	E(1)	E(1)	E(1)	E(β_1)	E(1)
$\hat{V}_2$	E(1)	E(1)	E(β_2)	E(1)	E(1)
$\hat{V}_3$	E(β_3)	E(1)	E(1)	E(1)	E(1)
$\hat{V}_4$	E(1)	E(β_4)	E(1)	E(1)	E(1)
$\hat{V}_5$	E(1)	E(1)	E(1)	E(1)	E(β_5)
$\prod$	E(β_3)	E(β_4)	E(β_2)	E(β_1)	E(β_5)

The user publishes $\hat{L}_i^{\lambda_i}$ and $\hat{V}_i^{\lambda_i,m_i}$ on the bulletin board. For ease of notation, in most cases, we will omit the superscript from $\hat{L}_i^{\lambda_i}$ and $V_i^{\lambda_i,m_i}$. Then, each user i needs to provide some mechanism (NIZK Π^{EG} in case of ElGamal encryption) for proving that:

1. L_i is a unit vector, i.e., that each element is a bit (i.e., equals 0 or 1), and the sum of all elements is 1
2. $\forall j \neq \lambda_i$, $\hat{V}_{ij} = E(1)$

Consider $V = (V_1, ..., V_m)$ as a matrix where each V_i acts as a row. Given that there are no collisions as seen in table 2 for $\hat{L}$ and $\hat{V}$ (note that $\hat{L}_{ij} = E(0)$ if and only if $\hat{V}_{ij} = E(1)$), in order to compute VSM output, the servers can publicly compute the product of each column in V in case of ElGamal-based SMPP (see Section 6.1) as $\prod_i \hat{V}_{ij} = E(\sum_i V_{ij}) = E(M_i)$ or compute the product $L \cdot V$ in case of BGN-based SMPP (see Section 6.2) which results in the encrypted message where each is located in the corresponding mix location. The following matrices are examples of $\hat{V}$ and $\hat{L}$ in the ElGamal-based VSM. The line under $\hat{V}$ (colored blue) depicts the VSM output which is computed as the product of the ciphertext on each column.

We define SMPP-Dec as SMPP such that all the messages writted table T (outputs by SMPP) are subsequently decrypted by the servers.

Linking UNS to SMPP. After user i receives its encrypted mix location $\hat{\lambda}_i$ as input from UNS, it needs to prove that $\hat{L}_i^{\lambda_i}$ is consistent with $\hat{\lambda}_i$. That is, given that the user retreived $\hat{\lambda}_i$ from UNS, it publishes a NIZK argument π^T which proves that $\hat{L}_i = (\hat{b}_{i1}, ..., \hat{b}_{i\lambda}, ..., \hat{b}_{im})$ satisfies $b_{i\lambda} = 1$ and $\forall\ j \neq \lambda,\ b_{ij} = 0$. In order to do this, first note that as $(b_1, ..., b_m)$ forms a unit vector, it holds that $\lambda = \sum_{j=0}^{m} j b_j$. Therefore, $\hat{\lambda} = (g^{r'}, h^{r'} g^{\lambda}) = (g^{r'}, h^{r'} g^{\sum_{j=0}^{m} j b_j}))$ for some $r' \in \mathbb{F}$. Since $g^{\sum_{j=1}^{m} j b_j} = (g^0)^{b_0} \cdots (g^m)^{b_m}$, anyone can compute g^j, $\forall j \in [m]$. Formally, let B_λ be the vector that satisfies $b_\lambda = 1$ and $\forall\ j \neq \lambda,\ b_j = 0$, let $R = (r_j)_{j=1}^{m} \in \mathbb{F}^m$, given user i's pubic input $(\hat{L}, \hat{\lambda})$, the user provides a NIZK proof for the relation:

$$\mathbb{R}_\lambda = \{((g, h, \hat{L}, \hat{\lambda}), (R, B_\lambda, r')) : ((g, h, \hat{L}), R, B_\lambda) \in \mathbb{R}_{unit}$$
$$, \hat{\lambda} = (g^{r'}, h^{r'} \prod_{j \in m} (g^j)^{b_j})\}$$

This relation can be proved using the Sigma protocol for linear relations [42] by presenting each g^j as some publically known group element g_j since the verifier can compute $g^j, \forall j \in [m]$. Note that the prover cannot provide encryption of a unit vector that is consistent with wrong $\lambda' \neq \lambda$ by the soundness of the Sigma protocol.

Note that the proof of encryption of unit vector can be combined with algorithm 3) to save computation. The proof is signed by the user. If verification does not pass, the SMPP vector is ignored, and the user is omitted from the system. The anonymity of the user is maintained, with regard to the output of UNS, as the proof is zero-knowledge with respect to the position λ. The formal argument that π_i^T is sufficient for VSM integrity under malicious inputs appears in Appendix F.4 (Theorem 6).

Therefore, each user i that participate in SMPP, select a message m_i and mix location λ_i and submit $I_i(m_i, \lambda_i) := (\hat{L}_i^{\lambda_i}, \hat{V}_i^{\lambda_i, m_i}, \Pi_i^{EG}, E(\lambda_i), \Pi_i^T)$ in case of ElGamal-based SMPP, and $I_i(m_i, \lambda_i) := (\hat{L}_i^{\lambda_i}, \hat{V}_i^{\lambda_i, m_i}, E(\lambda_i), \Pi_i^T)$ in case of BGN-based SMPP.

6.1 ElGamal-based VSM

We now present an El-Gamal-VSM. Each user encrypts the $\hat{L}_i$ and $\hat{V}_i$ elements using signed multiplicative ElGamal and the $\hat{V}_i$ (see Appendix section B.2), that is proven to be a non-malleable encryption scheme [53]. We note that homomorphic operations on these ciphertexts apply only to the first two elements in the tuple (as described in Appendix B.2), exactly as in standard ElGamal. Non-malleability is required only for user-submitted ciphertexts; the aggregated column products are server-computed public outputs and do not require authentication tags. Non-malleability prevents a malicious user from manipulating another user's submitted ciphertexts to gain information on their mix location. The user also publishes a Non-Interactive Zero-Knowledge proof (NIZK) that $\hat{L}_i$ and $\hat{V}_i$ has the right form (line 5, Protocol 3).

This proves that each element in $\hat{L}_i$ is an encryption of a bit, the sum of elements in L_i equals 1 (and hence L_i is a unit vector), and each element in $\hat{V}_i$ is an encryption of 1 except for the λ_ith element, which is the encryption of the message (see also Section D for a related relation). The proof reveals only that a unique special position exists and that the message vector is consistent with it; by zero knowledge it does not reveal which coordinate is the hidden slot. In order to ensure that all users encrypted their plaintexts with the shared public key, the trusted party computes the product ciphertexts in each row, i.e., $\prod_j \hat{L}_{ij} = (g^{\sum_i r_i}, h^{\sum_i r_i} g^{\tilde{M}_i})$ and test if the decrypted plaintext equals g^1 (in case where there are no trusted parties, i.e., when the shared public key was generated by the users, a proof of knowledge on the randomness $\sum_i r_i$ used for the encryption should be published). In order to compute the permutation, the servers compute the product of the elements in each column in $\hat{V}$ (compute the product of the first two elements of the standard). This is followed as for the user who selected $\lambda_i = j$ we have, $T(j) := \prod_i \hat{V'}_{ij} = E(\sum_i V'_{ij}) = E(M_i)$.

Protocol 3 Additively-Homomorphic VSM

Setup.

1. Multiple servers cooperatively generate an ElGamal public key pk and post it to the bulletin board. The private key sk is shared between the servers and can be a threshold (see appendix, section E.1).

Each user i:

1. Selects at random mix location $\lambda_i \in [m]$.
2. Prepares a vector $L_i = (L_{i1}, ..., L_{im})$ such that $L_{i,\lambda_i} = 1$ and $L_{ij} = 0$ for $j \neq \lambda_i$
3. Computes a vector of ciphertexts $\hat{L}_i = (E(L_{i1}), ..., E(L_{im}))$
4. Computes $\hat{V}_i$ so that $V_{i,\lambda_i} = E(M_i)$ and for any $j \neq \lambda_i$ compute $\hat{V}_{ij} = E(1)$. Let $\hat{V}_i = (V_{i1}, ..., V_{im})$.
5. Submits a NIZK proof $\Pi_i^{EG} = \Pi_{i1} \wedge \Pi_{i2}$ (zero knowledge with respect to the hidden position λ_i) such that
   (a) Π_{i1} is proof that $\forall j \in [m], (L_{ij} = 0 \cap V_{ij} = 1) \cup L_{ij} = 1$.
   (b) Π_{i2} is a proof that $\prod \hat{L}_i = E(1)$.
6. Publish $(\hat{L}_i, \hat{V}_i, \Pi_i^{EG})$ on the bulletin board.

Servers.

1. Wait until all users publish their output their tuples $(\hat{L}_i, \hat{V}_i, \Pi_i)$
2. For each user i, the servers verify the proof Π_i^{EG} with respect to $(\hat{L}_i, \hat{V}_i)$. If the verification fails then the servers ignore user i.
3. Construct matrices from all verified tuples $\hat{L}' = (\hat{L}_{i_1}, ..., \hat{L}_{i_k})$ and $\hat{V}' = (\hat{V}_{i_1}, ..., \hat{V}_{i_1})$, for $k \leq n$.
4. For every column j the servers compute $T(j) := \prod_i \hat{V'}_{ij}$ and publish the permutation $(T_1, ..., T_m)$.

6.2 BGN-based VSM

Trust assumption. The BGN setup requires an honest-majority MPC among the servers to generate the public parameters; this is the additional trust assumption relative to the ElGamal threshold-key variant.

We now describe BGN-based VSM where the vectors $\hat{L}_i$ and $\hat{V}_i$ are encrypted with the BGN cryptosystem (see Appendix C). The main difference from Protocol 6.1 is that users do not provide per-user NIZKs proving well-formedness of the submitted vectors $\hat{L}_i$ and $\hat{V}_i$. Instead, the servers exploit the somewhat homomorphic property of BGN, which allows arbitrarily many homomorphic additions and one homomorphic multiplication [9].

To prevent malicious users from mauling ciphertexts, we use IND-CCA secure encryption via the Naor–Yung transformation [44]. For ease of exposition we omit the duplicate-encryption and accompanying well-formedness proof from Algorithm 4. In order to generate the public and private parameters, the servers engage in secure multi-party computation (SMPC) with an honest majority to compute $n = q_1 q_2$ and the group generator h^{q_2} (this SMPC can be avoided by applying Freeman's technique to construct a prime-order group that supports pairings [29]).

This setup allows the users to send only encryptions of the elements of L_i and V_i to the servers, without per-user ZK proofs. The protocol is partially based on the construction in [9] and is presented in Algorithm 4. To check that the

encryptions in $\hat{L}_i$ are bits, the servers test whether $\sum_j r_j L_{ij}(1-L_{ij}) = 0$ via the pairing check in line 2(a), using the somewhat homomorphic property of BGN. If this verification passes, the servers then test whether there is a single mix location by checking line 2(b), i.e., $\prod_j \hat{L}_{ij} = E(\sum_j L_{ij}) = E(1)$. Together these imply that $\hat{L}_i$ encrypts a unit vector with high probability.

The servers do not need an explicit structural check on $\hat{V}_i$. It suffices that $\hat{V}_{i\lambda_i} = E(M_i)$, because the message at location j is recovered by computing the encrypted scalar product between the j-th columns of $\hat{L}$ and $\hat{V}$ across users. Using the somewhat homomorphic property of BGN,

$$\prod_{i\in[n]} e(\hat{L}_{ij}, \hat{V}_{ij}) = E'\left(\sum_{i\in[n]} L_{ij}V_{ij}\right) = E'\left((\sum_{t\neq\lambda_i} 0\cdot 1) + 1\cdot M_i\right) = E'(M_i),$$

where E' denotes the transformed encryption obtained after applying the pairing function (see Appendix C).

Protocol 4 Somewhat Homomorphic VSM

Setup.

1. The servers run a distributed key-generation protocol to compute pk and shares of the secret key $sk_1, \ldots, sk_m$.
2. The servers cooperatively select random values $r_1, \ldots, r_n$.
3. The servers fix a deterministic encryption of 1, denoted by [1].

Users.

1. Select a random mix location $\lambda_i \in [n]$.
2. Prepare a vector $L_i = (L_{i1}, \ldots, L_{im})$ such that $L_{i,\lambda_i} = 1$ and $L_{ij} = 0$ for $j \neq \lambda_i$.
3. Compute a vector of ciphertexts $\hat{L}_i = (E(L_{i1}), \ldots, E(L_{im}))$.
4. Compute $\hat{V}_i$ so that $\hat{V}_{i,\lambda_i} = E(M_i)$ and $\hat{V}_{ij} = E(1)$ for all $j \neq \lambda_i$.

Servers.

1. Construct matrices $\hat{V}$ and $\hat{L}$, and then perform the following tests:
   (a) $\prod_{j\in[n]} e(\hat{L}_{ij}, \hat{L}_{ij}\cdot[1]^{-1})^{r_j} \stackrel{?}{=} E'(0)$.
   (b) $\prod_j \hat{L}_{ij} \stackrel{?}{=} E(1)$.
2. If both tests pass, then for each column $j \in [m]$ compute

$$E'(M_j) = \prod_{i\in[n]} e(\hat{L}_{ij}, \hat{V}_{ij}). \tag{1}$$

3. Publish the permutation $(E'(M_1), \ldots, E'(M_m))$.

6.3 SMPP via Fully Homomorphic Encryption

Another possible instantiation of SMPP uses fully homomorphic encryption (FHE), such as the Brakerski–Gentry–Vaikuntanathan (BGV) scheme. The purpose of this variant is to avoid sending explicit m-dimensional encrypted vectors.

Instead of publishing the full vectors $\hat{L}_i$ and $\hat{V}_i$, user i publishes encryptions of a binary description of its mix location λ_i, together with encrypted payload bits that are routed homomorphically to the selected leaf. This reduces the client upload from $O(m)$ to $\tilde{O}(\log m)$, while requiring FHE evaluation and consistency proofs for the compressed input. Correctness of the compressed encoding can be proved using polylogarithmic-size lattice-based Σ-protocols [6].

The construction is based on *homomorphic traversal* [16]. Consider a complete binary tree with m leaves. A value initially placed at the root is routed to one leaf using encrypted controller bits. At each internal node, the homomorphic selection subroutines

$$\text{HS-L}(b, c) := (1 - c) \cdot b \quad \text{and} \quad \text{HS-R}(b, c) := c \cdot b$$

copy the current value b either to the left child or to the right child, according to the controller bit c. All controller bits and intermediate node values remain IND-CPA encrypted throughout the traversal. Since a complete binary tree with m leaves has $m - 1$ internal nodes, the traversal uses $m - 1$ homomorphic selection steps and has depth $\log m$.

To compress the location vector, user i decomposes its mix location $\lambda_i \in [m]$ into bits

$$b_{i,0}, \ldots, b_{i,d-1}, \qquad d = \lceil \log m \rceil, \qquad \lambda_i = \sum_{j=0}^{d-1} 2^j b_{i,j}.$$

The encrypted bits $\hat{b}_{i,j} = E(b_{i,j})$ serve as the demultiplexer controllers for the traversal circuit. To compress the message vector, the user also decomposes a fixed-size ℓ-bit payload representation into bits $v_{i,0}, \ldots, v_{i,\ell-1}$ and publishes encryptions $\hat{v}_{i,j} = E(v_{i,j})$. The same encrypted controller bits route these encrypted payload bits to the leaf indexed by λ_i, thereby producing under encryption the same sparse placement that the explicit vectors $\hat{L}_i$ and $\hat{V}_i$ would have represented.

The user publishes the compressed location encoding

$$(\hat{\lambda}_i, \hat{b}_{i,0}, \ldots, \hat{b}_{i,d-1}, \pi_i^{\lambda})$$

and the compressed payload encoding

$$(\hat{V}_i, \hat{v}_{i,0}, \ldots, \hat{v}_{i,\ell-1}, \pi_i^{V}).$$

The accompanying lattice-based zero-knowledge proofs certify that the submitted ciphertexts encrypt bits and that the encrypted location is consistent with its binary decomposition. Equivalently, the proof enforces the relation

$$\hat{\lambda}_i = h^r g^{\sum_{j=0}^{d-1} 2^j b_{i,j}},$$

in the corresponding encrypted-domain notation, together with the bitness of the $b_{i,j}$'s and the well-formedness of the encrypted payload bits. Such relations can be proved using compressed lattice-based proof systems [6]. The FHE traversal

Table 3. Online cost summary for SMPP with n users and m slots (one batch).

Instantiation	Client upload	Public aggregation	Output decryption
ElGamal+NIZK	$O(m)$	$O(nm)$ group mults	m threshold decryptions
BGN+H-Eval	$O(m)$	$O(nm)$ group mults	m threshold decryptions
FHE traversal	$\tilde{O}(\log m + \ell)$	$O(\ell m)$ mults; depth $\log m$	decrypt m slots

requires encrypting the relevant bits under a ring learning with errors (RLWE)-based encryption scheme and using the RLWE-to-RGSW (Ring Gentry–Sahai–Waters) conversion procedure of [16] for the encrypted controller values used by the selection circuit.

Routing one encrypted value through a complete binary tree with m leaves uses $m-1$ selection nodes. If both children are materialized using HS-L and HS-R, this requires $2(m-1)$ ciphertext multiplications and has multiplicative depth $\log m$. Routing one location marker and ℓ payload bits therefore requires $2(\ell+1)(m-1)$ ciphertext multiplications.

To ensure malicious security, the user additionally proves that the encrypted controller values are bits and that they form a valid binary decomposition of the encrypted location. These are circuit-level estimates and do not include the concrete costs of RLWE-to-RGSW conversion or of generating and verifying these consistency proofs. Thus, the FHE construction reduces client upload asymptotically, while its concrete computation cost depends on the selected FHE and proof systems. An online cost summary for SMPP with n users and m slots in one batch is shown in Table 3.

The FHE row treats ℓ as the payload bit length and reports the routing work per user; across all users the total routing work is $O(n\ell m)$. For fixed-size payloads, the client-upload term is $\tilde{O}(\log m)$.

Server-side aggregation cost. In the ElGamal-based SMPP, after verifying users, the servers compute for each slot $j \in [m]$ the ciphertext product $T(j) = \prod_{i=1}^{k} \hat{V}'_{ij}$ and publish $(T(1), \ldots, T(m))$ (where $k \leq n$ is the number of accepted users). This requires $(k-1)$ ciphertext multiplications per slot, hence $m(k-1)$ ciphertext multiplications total. Since an ElGamal ciphertext consists of two group elements and multiplication is component-wise, each ciphertext multiplication costs two group multiplications, for a total of $2m(k-1)$ group multiplications.

In the BGN-based SMPP, for each slot j the servers compute $E'(M_j) = \prod_{i=1}^{k} e(\hat{L}_{ij}, \hat{V}_{ij})$. This uses k pairings and $(k-1)$ multiplications in the target group per slot, hence mk pairings and $m(k-1)$ target-group multiplications overall.

FHE traversal gate count. In the FHE variant, homomorphic traversal evaluates selection gates of the form HS-L$(b, c) = (1-c)\cdot b$ and HS-R$(b, c) = c\cdot b$. Computing both children uses two ciphertext multiplications per node. For a complete binary tree with m leaves, traversal uses $(m-1)$ selections, so the traversal circuit contains $2(m-1)$ ciphertext multiplications and has multiplicative depth $\log m$.

Practical considerations. Our FHE-based compression is primarily of asymptotic interest. Gate-bootstrapping FHE schemes typically require substantial evaluation key material (bootstrapping keys) that is generated once and reused across evaluations. For example, TFHE reports parameterizations with bootstrapping keys on the order of tens of megabytes (e.g., 16 MB reported in the journal version) [14]. In contrast, early FHEW parameter sets reported bootstrapping keys in the hundreds of megabytes (e.g., approximately 846 MB, plus additional key-switching material in one reference implementation report) [23]. Accordingly, for realistic deployments our ElGamal- and BGN-based SMPP instantiations remain more practical, while the FHE variant demonstrates that $O(\log n)$-size location encodings are possible under standard lattice assumptions.

7 VSM

We are finally ready to define VSM. VSM is the composition of UNS and SMPP. UNS is executed once per *epoch*: each user selects at random $O(\log m)$ candidate mix locations (as justified by Lemma 2), and the servers run UNS so that each surviving user obtains an encrypted mixed location together with its plaintext interpretation from its own submitted list. The users then run SMPP for the current batch using their plaintext slots λ_i. If the same slots are reused across multiple epochs, the resulting transcript yields a pseudonymous slot for each user; therefore a fresh UNS execution is required whenever unlinkability across epochs is desired. The SMPP outputs a table T that realizes the permutation determined by the mixed locations $\lambda_{i i\in[n]}$, i.e., $T(\lambda_i) = E(m_i)$.

Protocol 5 VSM

Phase I: Preprocessing by UNS

1. Set a list-length parameter $K = O(\log m)$.
2. Each user $i \in [n]$ randomly selects K candidate locations and computes $\hat{P}_i := (E_{pk}(l_{i1}), E_{pk}(l_{i2}), \ldots, E_{pk}(l_{iK}))$, which it sends to the servers.
3. The servers compute $UNS(\hat{P}_1, \ldots, \hat{P}_n)$ and publish the resulting encrypted mix locations. Each surviving user i then recovers λ_i in plaintext by identifying which of its posted ciphertexts appears in the public UNS output.

Phase II: Message submission by SMPP in the current epoch

1. Each participating user computes the SMPP input $I_i(m_i, \lambda_i)$ and publishes it on the bulletin board.
2. The servers execute the verification and aggregation steps of SMPP over $I_1(m_1, \lambda_1), \ldots, I_k(m_k, \lambda_k)$ and output a public aggregated table T.
3. The servers jointly decrypt the messages in T.

8 Security Analysis

The following analysis applies to a single epoch. If UNS is not rerun across epochs, outputs remain anonymous with respect to user identity but become link-

able to the same pseudonymous slot; formal unlinkability across epochs therefore requires a fresh UNS execution per epoch and is not claimed here.

We analyze VSM in the malicious model, where parties may deviate arbitrarily from the protocol. The adversary $\mathcal{A}$ may corrupt any number of users except two, and any number of servers except one. This is the minimal condition under which anonymity is definable; in practice, the effective anonymity set grows with the number of honest participants. The bulletin board is public, authenticated, and append-only, and all messages posted to it are signed. If the board provides signed submission acknowledgments, omitting an acknowledged post makes the board publicly accountable for that removal. Otherwise, such omissions are classified as board-level censorship.

We establish three properties for UNS, SMPP, and their composition VSM. *Anonymity* captures that an adversary cannot link honest users to their mixed locations or messages beyond what is implied by the final multiset of outputs. *Integrity* captures that, except with negligible probability, honest users' messages appear at their assigned locations (or the protocol aborts). *Self-verifiability* holds for SMPP-Dec: once the output table is jointly decrypted, each honest user can verify correctness by checking the entry at their own secret location. A full security analysis (definitions and proofs) appears in Appendix F for the malicious model; here we state the main results.

Theorem 1. *Protocol 1 satisfies UNS anonymity and integrity.*

Theorem 2. *Protocols 3 and 4 satisfy SMPP anonymity and integrity.*

Theorem 3. *The VSM protocol (UNS followed by SMPP) satisfies anonymity and integrity, and SMPP-Dec provides unconditional individual verifiability for honest users.*

9 Conclusions and Future Work

In this work, we introduce a new ACN scheme called verifiable self-mix (VSM) that executes private permutation of encrypted messages, distributed in an easily verifiable manner. We formally defined VSM and proved the anonymity, integrity, and perfect verifiability properties of UNS and SMPP. We presented different designs for "sensitive" protocols concerning different trust assumptions and, in particular, the effort to circumvent the mixnet's trust assumption on the mix server. We built our architecture modularly so that modifying one part of the system may not affect the other parts of the system. VSM inherits robustness properties from the underlying bulletin board: malicious users cannot undetectably modify or remove other users' ciphertexts, because all messages are signed and posted to an append-only log. Unlike classic DC-nets, where a single jammer can prevent delivery of honest messages until they are identified, in VSM a malicious user can at most withhold or corrupt their *own* contribution; attempts to tamper with other users' inputs are either prevented or detected by

the UNS and SMPP checks. Denial-of-service attacks on the bulletin board itself (e.g., network-level flooding or censorship) remain out of scope.

We now discuss other possible directions for instantiating VSM. UNS can be modified by using error correction codes as used in [17]. A different approach from UNS is followed by the scheme of [50], which uses FHE for UNS and reduces the server-side collision-identification work to $O(n)$ operations via homomorphic hashing. Instead of performing PET, the servers invoke a homomorphic hash h_k, for a joint encryption key k, over the encryption of the mixed integer $E(\lambda_i)$, which results in $E(h(\lambda_i))$. After joint decryption, the values $h(\lambda_i)$ are published and used in a hash table for identifying collisions. The amortized lookup cost is then $O(1)$, analogously to testing whether a hash-table bucket is occupied.

Open directions include making UNS non-interactive (or reducing the number of collision-testing rounds), obtaining $o(n)$ per-user SMPP communication without relying on FHE, and analyzing VSM under adaptive corruptions and long-lived epochs.

References

1. M. Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 437–447. Springer, 1998.
2. I. Abraham, B. Pinkas, and A. Yanai. Blinder: MPC based scalable and robust anonymous committed broadcast. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS 2020)*. ACM, 2020.
3. B. Adida and D. Wikström. How to shuffle in public. In *Theory of Cryptography Conference*, pages 555–574. Springer, 2007.
4. B. Adida and D. Wikström. Offline/online mixing. In *International Colloquium on Automata, Languages, and Programming*, pages 484–495. Springer, 2007.
5. N. Alexopoulos, A. Kiayias, R. Talviste, and T. Zacharias. Mcmix: Anonymous messaging via secure multiparty computation. In *USENIX security symposium*, pages 1217–1234, 2017.
6. T. Attema, R. Cramer, and L. Kohl. A compressed σ-protocol theory for lattices. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II*, pages 549–579. Springer, 2021.
7. L. Barman, I. Dacosta, M. Zamani, E. Zhai, A. Pyrgelis, B. Ford, J. Feigenbaum, and J. Hubaux. Prifi: Low-latency anonymity for organizational networks. *Proceedings on Privacy Enhancing Technologies*, 2020(4):24–47, 2020.
8. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 394–403. IEEE, 1997.
9. D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of cryptography conference*, pages 325–341. Springer, 2005.
10. E. Boyle, S. Goldwasser, and S. Tessaro. Communication locality in secure multi-party computation. In *Theory of Cryptography Conference*, pages 356–376. Springer, 2013.
11. D. Chaum. The dining cryptographers problem. *J. cryptology*, 1:65–75, 1988.

12. D. Chaum, D. Das, F. Javani, A. Kate, A. Krasnova, J. De Ruiter, and A. T. Sherman. cmix: Mixing with minimal real-time asymmetric cryptographic operations. In *Applied Cryptography and Network Security, ACNS*. Springer, 2017.
13. D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.
14. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.
15. M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a secure voting system. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 354–368. IEEE, 2008.
16. K. Cong, D. Das, J. Park, and H. V. Pereira. Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. *Cryptology ePrint Archive*, 2022.
17. H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE, 2015.
18. H. Corrigan-Gibbs and B. Ford. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 340–350, 2010.
19. H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford. Proactively accountable anonymous messaging in verdict. In *22nd USENIX Security Symposium (USENIX Security 2013)*, pages 147–162. USENIX Association, 2013.
20. R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European transactions on Telecommunications*, 8(5):481–490, 1997.
21. G. Danezis and C. Diaz. A survey of anonymous communication channels. Technical Report MSR-TR-2008-35, Microsoft Research, Feb. 2008.
22. D. Das, E. V. Mangipudi, and A. Kate. Organ: Organizational anonymity with low latency. *Proceedings on Privacy Enhancing Technologies*, 2022(3):582–605, 2022.
23. L. Ducas and D. Micciancio. Fhew: bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 617–640. Springer, 2015.
24. L. A. Dunning and R. Kresman. Privacy preserving data sharing with anonymous id assignment. *IEEE transactions on information forensics and security*, 8(2):402–413, 2012.
25. S. Eskandarian and D. Boneh. Clarion: Anonymous communication from multiparty shuffling protocols. In *Network and Distributed System Security Symposium (NDSS 2022)*. Internet Society, 2022.
26. S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh. Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. In *30th USENIX Security Symposium (USENIX Security 2021)*, pages 1775–1792. USENIX Association, 2021.
27. S. Faust, M. Kohlweiss, G. A. Marson, and D. Venturi. On the non-malleability of the fiat-shamir transform. In *Progress in Cryptology-INDOCRYPT 2012: 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings 13*, pages 60–79. Springer, 2012.
28. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

29. D. M. Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 44–61. Springer, 2010.
30. A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *International Workshop on the Theory and Application of Cryptographic Techniques*, pages 244–251. Springer, 1992.
31. K. Gjøsteen and A. S. Lund. An experiment on the security of the norwegian electronic voting protocol. *Annals of Telecommunications*, 71(7):299–307, 2016.
32. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on computing*, 17(2):281–308, 1988.
33. P. Golle and A. Juels. Dining cryptographers revisited. In *Advances in Cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques*, pages 456–473. Springer, 2004.
34. J. Groth. A verifiable secret shuffle of homomorphic encryptions. *Journal of Cryptology*, 23(4):546–579, 2010.
35. K. Hamada, R. Kikuchi, D. Ikarashi, K. Chida, and K. Takahashi. Practically efficient multi-party sorting protocols from comparison sort algorithms. In *Information Security and Cryptology–ICISC 2012: 15th International Conferenc*, pages 202–216. Springer, 2013.
36. X. Huang and R. Chen. A survey of key management service in cloud. In *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, pages 916–919. IEEE, 2018.
37. S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen. Intel software guard extensions: Epid provisioning and attestation services. *White Paper*, 1(1-10):119, 2016.
38. A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford. Atom: Horizontally scaling strong anonymity. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP 2017)*, pages 406–422. ACM, 2017.
39. S. Langowski, S. Servan-Schreiber, and S. Devadas. Trellis: Robust and scalable metadata-private anonymous broadcast. In *Network and Distributed System Security Symposium (NDSS 2023)*. Internet Society, 2023.
40. V. Lyubashevsky, N. K. Nguyen, and M. Plançon. Lattice-based zero-knowledge proofs and applications: shorter, simpler, and more general. In *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA*, pages 71–101. Springer, 2022.
41. D. Mardi, S. Tanwar, and J. Howlader. Multiparty protocol that usually shuffles. *Security and Privacy*, 2021.
42. U. Maurer. Zero-knowledge proofs of knowledge for group homomorphisms. *Designs, Codes and Cryptography*, 77(2):663–676, 2015.
43. M. Movahedi, J. Saia, and M. Zamani. Secure multi-party shuffling. In *International Colloquium on Structural Information and Communication Complexity*, pages 459–473. Springer, 2015.
44. M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 427–437, 1990.
45. C. A. Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125, 2001.

46. V. Niemi and A. Renvall. How to prevent buying of votes in computer elections. In *International Conference on the Theory and Application of Cryptology*, pages 164–170. Springer, 1994.
47. C. Park, K. Itoh, and K. Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 248–259. Springer, 1993.
48. T. P. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology—EUROCRYPT'91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10*, pages 522–526. Springer, 1991.
49. A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. The loopix anonymity system. In *26th USENIX Security Symposium (USENIX Security 2017)*, pages 1199–1216. USENIX Association, 2017.
50. P. B. Rønne, A. Atashpendar, K. Gjøsteen, and P. Y. Ryan. Coercion-resistant voting in linear time via fully homomorphic encryption. *Financial Cryptography and Data Security*, 2019.
51. T. Ruffing, P. Moreno-Sanchez, and A. Kate. P2p mixing and unlinkable bitcoin transactions. *Network and Distributed System Security Symposium*, 2017.
52. K. Sako and J. Kilian. Receipt-free mix-type voting scheme. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 393–403. Springer, 1995.
53. C. P. Schnorr and M. Jakobsson. Security of signed elgamal encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 73–89. Springer, 2000.
54. M. Stadler. Publicly verifiable secret sharing. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 190–199. Springer, 1996.
55. M. Taassori, A. Shafiee, and R. Balasubramonian. Vault: Reducing paging overheads in sgx with efficient integrity verification structures. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 665–678, 2018.
56. D. Wikström. A sender verifiable mix-net and a new proof of a shuffle. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 273–292. Springer, 2005.
57. A. Damodaran, S. Rastikian, P. B. Rønne, and P. Y. A. Ryan. Hyperion: Transparent end-to-end verifiable voting with coercion mitigation. Cryptology ePrint Archive, Paper 2024/1182, 2024.
58. R. Küsters, J. Müller, E. Scapin, and T. Truderung. sElect: A lightweight verifiable remote voting system. In *IEEE 29th Computer Security Foundations Symposium (CSF 2016)*, pages 341–354, 2016.
59. P. Y. A. Ryan, P. B. Rønne, and V. Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. Cryptology ePrint Archive, Paper 2015/1105, 2015.

A Detailed Cryptographic Background

This appendix mainly fixes notation and collects standard cryptographic background used throughout the paper; the protocol-specific details for PET, threshold ElGamal, and the full security proofs appear in the later appendices. Let q be a prime and $\mathbb{F}_q$ be a prime field for which it is hard to compute the discrete log. Let $\mathbb{G}$ and $\mathbb{G}_1$ be two cyclic groups of order n and g and g_1 be the generators of these groups respectively. Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ denote a pairing function. Encryption using public key pk is denoted by $E_{pk}(m)$. For ease of notation, we will mostly omit pk from notation (and simply write $E(m)$), when m is encrypted using the server's public key. We will make use of Signed ElGamal, Threshold ElGamal, and BGN cryptosystems (described in the Appendix). The notation $[n]$ refers to the set of integers $\{1, ..., n\}$. We also use the notation of a ciphertext that is an encryption of x as $\hat{x} := E(x)$. We extend this notation entry-wise to vectors, i.e., for $v = (v_1, \ldots, v_m)$ we write $\hat{v} := (E(v_1), \ldots, E(v_m))$.

B The ElGamal cryptosystem

B.1 The ElGamal Cryptosystem

Let p and q be primes such that $q|p-1$. Let $\mathbb{G}$ be a subgroup of $\mathbb{F}_q^*$ of order q with generator g. Let E be an ElGamal cryptosystem, with secret key sk and corresponding public key $h := g^{sk}$. To encrypt a message, the user first encodes the message in $\mathbb{F}_q^*$. Then, it picks a random $\mathbb{F}_q^*$ and computes $(g^r \ mod \ p, h^r g^m \ mod \ p)$. We will mostly make use of other types of ElGamal, such as Signed-ElGamal, and Threshold-ElGamal (in Appendix E.1) that is described in the appendix.

B.2 Signed ElGamal

The description of this scheme is according to [53]. Let p and q be primes such that $q|p-1$. Let $\mathbb{G}$ be a subgroup of $\mathbb{F}_p^*$ of order q with generator g. The private and public key pair for ElGamal encryption is $x, h = g^x$ where x is random in $\mathbb{F}_q$.

Encryption:

1. Map the message M to an element $m \in \mathbb{G}$ using an invertible function.
2. Pick random $r, s \in \mathbb{F}_q$
3. compute $(g^r, h^r m)$
4. compute $c := H(g^s, g^r, h^r m)$ or $H(g^s, g^r, h^r g^m)$ for multiplicative ElGamal
5. compute $z := s + cr$
6. output $(g^r, h^r m, c, z)$

Decryption: Let (a, b, c, z) be the received message. In order to decrypt the ciphertext, the decryptor:

1. Tests if $c = H(g^z a^{-c} (mod \ p), a, b) \ (mod \ q)$
2. If test does not pass then abort

3. Compute $\frac{b}{a^x}$

Homomorphic property: The additive homomorphism of signed ElGamal applies only to the first two ElGamal components, exactly as in standard ElGamal. In particular, multiplying two ciphertexts combines only their (a, b) components. The Schnorr authentication tag is verified on each user-submitted ciphertext before aggregation; it is not recomputed for the public aggregated outputs produced by the servers. Thus non-malleability is required only for user-submitted ciphertexts, while the aggregated column products are treated as public server outputs.

C The Boneh-Goh-Nissim Cryptosystem

Let q_1 and q_2 be two primes of τ bits (which represent the security parameter) and let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups both of order $N = q_1 q_2$. We denote by g_1, g_2 a generators in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. Let e : $\mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be a bilinear maping such that $g_2 = e(g, g)$ generates $\mathbb{G}_2$ which satisfies the property that $\forall a, b \in \mathbb{G}_1$ and $\forall x, y \in \mathbb{F}_q$, $e(a^x, b^y) = e(a, b)^{xy}$ [9].

Key generation. Select two random generators g, u in $\mathbb{G}_1$ and set ouboic key $h = u^{q_2}$. Then h is a random generator of the subgroup of $\mathbb{G}_1$ of order q_1. The public key is $PK = (n, \mathbb{G}_1, \mathbb{G}_2, e, g, h)$. The private key is $SK = q_1$.
To encrypt a message $M \in [T]$, where $T < q_2$, select a random $r \in \mathbb{Z}_N$ and computes $c = g^M h^r$.
To decrypt ciphertext c, given secret key $sk = q_1$, compute the discrete logarithm of c^{q_1} to the base g^{q_1} (note that $c^{q_1} = (g^M h^r)^{q_1} = g^{Mq_1}$) by brute force over the declared message space $[T]$, which is polynomial in the security parameter in our applications (e.g., when M is a mix location or another bounded-domain value).

Homomorohic Properties The system is additively homomorphic, since given encryptions $C_1, C_2 \in \mathbb{G}_1$ of the messages $m_1, m_2 \in \{0, 1, ..., T\}$ respectively, anyone can create an encryption of $m_1 + m_2 \mod N$ by computing the product $C = C_1 C_2 h$ for a random $r \in \{0, ..., N-1\}$.

In addition, BGN is also one-time multiplicatively homomorphic as anyone can multiply two encrypted messages (once) using the bilinear map. Let $g_1 = e(g, g)$ and $h_1 = e(g, h)$. Then g_1 is of order N and h_1 is of order q_1. Let $h = g^{kq_2}$ for some (unknown) $k \in \mathbb{F}$. Suppose we are given ciphertexts $C_1 = g^{m_1} h^{r_1} \in \mathbb{G}_1$ and $C_2 = g^{m_2} h^{r_2} \in \mathbb{G}_1$. In order to compute the encryption of the product $m1 \cdot m2 \mod N$ given C_1 and C_2, select a random $r \in Z_N$ and compute $C = e(C_1, C_2) h_1^r \in \mathbb{G}_2$ Then $e(C_1, C_2) h_1^r = e(g^{m_1} h^{r_1}, g^{m_2} h^{r_2}) h_1^r = g_1^{m_1 m_2} h_1^{\tilde{r}}$ where $\tilde{r} = m_1 r_2 + r_2 m_1 + k q_2 r_1 r_2 + r$.

D Special Honest Verifier Zero-Knowledge Argument

Let $\mathbb{R}$ be a polynomial time decidable relation. A non-interactive zero-knowledge argument (we refer to it as NIZK) for NP relation $\mathbb{R}$ consists of PPT algorithms

(G, P, V) where: G generates the crs (group specification, public key). Given statement x, and witness w, such that $(x, w) \in \mathbb{R}$, the algorithm $P(crs, x, w)$, outputs a proof π, and $V(crs, x, \pi) \to \{0, 1\}$ (1 for accept and 0 for reject). A non-interactive zero-knowledge argument satisfies the following conditions:

1. Completeness. A NIZK system is said to be perfectly complete if an honest prover with a valid witness can always convince an honest verifier. More formally, for any $crs, x, w) \in R$, we have that

$$\Pr\left[\begin{array}{c}\text{crs} \leftarrow \mathcal{G}\left(1^{\Lambda}, \mathbb{R}\right), \pi \leftarrow \mathcal{P}(\text{crs}, x, w) : \\ \mathcal{V}(\text{crs}, x, \pi) = 1\end{array}\right] = 1$$

2. Computational soundness. A NIZK scheme for the language $\mathbb{R}$ is said to be computationally sound if for all polynomially bounded adversaries $\mathcal{A}$,

$$\Pr\left[\begin{array}{l}\text{crs} \leftarrow \mathcal{G}\left(1^{\Lambda}, \mathbb{R}\right), (x, \pi) \leftarrow \mathcal{A}(\text{crs}) : \\ (\mathcal{V}(\text{crs}, x, \pi) = 1) \wedge (x \notin \mathbb{R})\end{array}\right] \approx 0$$

3. Honest Verifier Zero Knoledge. An interactive proof system between a prover and a verifier is statistical if for every PPT verifier V^* and polynomially bounded prover, there exists a PPT simulator such that $\forall x \in \mathcal{R}$ and $Z \in \{0, 1\}^{|x|}$ the distribution of the transcript $< P(x), V^*(x, z)$ is indistinguishable from the distribution of $S(x, z)$

In addition, we require NIZK to be non-malleable so that the adversary cannot produce the same proof for the same statement (this can be done by adding implicitly a session ID to the statement). We use the fact that it is constructed as a sigma protocol on which a Fiat-Shamir heuristic is applied. [28] (assuming random oracle model).

NIZK for Encryption of a Unit Vector A unit vector is a vector consisting of 0 in all entries except a single entry of 1. To prove that a vector of ciphertexts $\hat{b} = (\hat{b}_j)_{j=1}^m \in \mathbb{G}^m$ is an ElGamal encryption of a unit vector (given that the prover does not know the trapdoor x, such that $h = g^x$), we first formalize a relation that $\hat{b}_j$ for a simple ElGamal encryption of a bit :

$$\mathbb{R}_{\hat{b}(j)} = \{((g, h, \hat{b}_j), (r_j, b_j)) : b_j \in \{0, 1\}, \hat{b_j} = (g^{r_j}, h^{r_j} g^{b_j})\}$$

and for signed ElGamal:

$$\mathbb{R}_{\hat{b}(j)} = \{(g, h, \hat{b}_j, c_j, z_j), (r_j, s_j, b_j)) : b_j \in \{0, 1\}, \hat{b_j} = (g^{r_j}, h^{r_j} g^{b_j})$$
$$, c = H(g^{s_j}, g^{r_j}, h^{r_j} g^{b_j}), z_j = s_j + c_j r_j\}$$

To prove that $\hat{b}$ is a unit vector, we first provide NIZK $\mathbb{R}_{\hat{b}(j)}$ for all $j \in [m]$ and then show that the multiplication of all ciphertexts gives an encryption of

1. Let $b = (b_j)_{j=1}^m \in \{0,1\}^m$ and $R = (r_j)_{j=1}^m \in \mathbb{F}_q^m$ then the prover provide a NIZK proof for the relation:

$$\mathbb{R}_{unit} = \{((g,h,\hat{b}),(R,b)) : \forall j \in [m], ((g,h,\hat{b}_j), r_j, b_j) \in \mathbb{R}_{\hat{b}(j)}$$
$$, \prod_{j\in[m]} \hat{b}_j = (g^{\sum_{j\in[m]} r_j}, h^{\sum_{j\in[m]} r_j} g^1)\}$$

where the second term ensures that $g^{\sum_{j\in m} b_j} = g^1$. In order to do this, the prover can provide proof of knowledge of R and b, via generic linear Sigma protocol [42].

E Plaintext Equivalence Test

PET is a protocol between multiple servers for testing if two ElGamal cipher texts are encryptions of the same plaintext. Both ciphertexts are encrypted with a common public key where the corresponding secret key is shared between the servers (the is shared using decentralized key generation protocol [48, 54]). This protocol is secure in the malicious setting, such that any server that does not follow the steps protocol can be identified.

Protocol 6 Plaintext Equivalence Test [15]

Input: Public ciphertexts c_1, c_2, where $c_i := (a_i, b_i)$ for $i \in \{1,2\}$, and private secret key share x_i of server i
Each server i computes:
1. Compute $(d,e) = (a_0/a_1, b_0/b_1)$
2. Select $z_i \leftarrow \mathbb{F}_q$, and compute $(d_i, e_i) = (d^{z_i}, e^{z_i})$
3. Publish commitment to (d_i, e_i)
4. Wait until all commitments are published
5. Compute a NIZK of $\Pi_i^{dlog} = EqDlog(d, e, d_i, e_i)$ [15]
6. Publish (d_i, e_i, Π_i^{dlog})
7. V all commitments and proofs
8. Compute $c' = (\prod_i d_i, \prod_i e_i)$

All server compute m' =distributedDecryptionElGamal(c') [15] using private key shares $(x_1, ..., x_k)$
if $m' = 1$ **then** return 1
else return 0
end if

E.1 Threshold ElGamal

According to [20]: the object of a threshold scheme for public-key encryption is to share a private key among a set of receivers such that messages can only be decrypted when a a substantial set of receivers cooperate. The main protocols of a threshold system are:

1. a key generation protocol to generate the private key jointly by the receivers

2. a decryption protocol to jointly decrypt a ciphertext without explicitly reconstructing the private key

Key generation. As part of the setup procedure, the authorities execute a key generation protocol such that each authority A_j possesses a share $s_j \in \mathbb{F}_q$ of a secret s. The authorities are committed to these shares as the values $h_j = g^{s_j}$ are made public. Furthermore, the shares s_j are such that the secrets can be reconstructed from any set $\wedge$ of t shares using appropriate Lagrange coefficients say:

$$s = \sum_{j \in \wedge} s_j \lambda_{j,\wedge}, \quad \lambda_{j,\wedge} = \prod_{l \in \wedge \setminus \{j\}} \frac{l}{l-j}$$

The public key $h = g^s$ is announced to all participants in the system.

Decryption. To decrypt a ciphertext $(x, y) = (g^\alpha, h^\alpha m)$ without reconstructing the secret s, the authorities execute the following protocol:

1. Each authority Aj broadcasts $w_j = x^{s_j}$ and proves in zero-knowledge that $\log_g h_j = \log_x w_j$ according to the following protocol:

Prover		Verifier
$[(x, y) = (g^\alpha, h^\alpha)]$		
$w \in_R \mathbb{Z}_q$		
$(a, b) \leftarrow (g^w, h^w)$	$\xrightarrow{a,b}$	
		$c \in_R \mathbb{F}_q$
	$\xleftarrow{c}$	
$r \leftarrow w + \alpha c$	$\xrightarrow{r}$	$g^r \stackrel{?}{=} ax^c$
		$h^r \stackrel{?}{=} by^c$

2. Let $\wedge$ denote any subset of t authorities who passed the zero-knowledge proof. The plaintext can be recovered as

$$m = y / \prod_j w_j^{\lambda_{j,\wedge}}$$

F Full Security Proofs

The following analysis applies to a single epoch; if UNS is reused across epochs, the resulting outputs remain anonymous with respect to user identity but become linkable to the same pseudonymous slot.

We establish the validity of our protocol within the context of the malicious model. In this model, protocol participants have the freedom to deviate from the specified protocol in arbitrary ways. $\mathcal{A}$ can compromise any number of users except two users, and any number of servers except one. We suppose that the communication of all entities to the bulletin board is done through secure channels (authenticated) and that the bulletin board is public and authenticated (all parties are assumed to have associated private and public keys that allow them

to sign their messages). We assume that the bulletin board is append-only so that information cannot be deleted or modified.

Whenever there is a use of trusted hardware (TH), we do not assume any trust regarding the host of the TH. In particular, the host may try to modify the message that should be assigned to the TH. In addition, the TH host has a private channel to the TH which may be used to send messages to the TH without being detected e.g., to break the anonymity of honest users). Within this section, we present three properties of UNS, SMPP and VSM: anonymity, integrity, and verifiability. ewline ewline

Let $\epsilon > 0$ be a security parameter. A function $negl : \mathbb{N} \to \mathbb{R}$ is called negligible if for all $c > 0$. there exists k such that $negl(k) < \frac{1}{k^c}$ for all $k > k_0$.

To prove the security properties, we will use a modified version, though equivalent, of the CCA game. [2] The CCA game *Anon-SMPP* is defined as follows:

1. Setup phase. The challenger $\mathcal{C}$ chooses a keypair (x, y), and gives the public key y to the $\mathcal{A}$.
2. First query phase. $\mathcal{A}$ encrypt polynomially many messages C using y and send them to $\mathcal{C}$, who sends $Dec_x(C)$ back $\mathcal{A}$.
3. Challange phase. $\mathcal{A}$ chooses two message m_0, m_1 and send to $\mathcal{C}$. $\mathcal{C}$ selects uniformly at random a bit $b \in \{0, 1\}$ and compute ciphertext $(E(m_b), E(m_{1-b}))$
4. Second query phase. Repeat the process as in the first query phase for polynomially more messages.
5. Output phase: $\mathcal{A}$ outputs its guess b' for b.

F.1 Anonymity

We define the notion of anonymity as a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, where the adversary $\mathcal{A}$ is a probabilistic polynomial time (PPT) Turing machine. Upon request, $\mathcal{C}$ runs the protocol and returns the outputs to $\mathcal{A}$. Let n be the number of users and assume each user sends a message during every round. Adversary $\mathcal{A}$ plays the role of dishonest users, while challenger $\mathcal{C}$ executes the protocol on behalf of honest nodes and users. $\mathcal{A}$ is assumed not to know the private keys of the honest senders.

$\mathcal{A}$ can read all messages from the bulletin board, but cannot modify them.

UNS Anonymity The anonymity game *A-UNS* is defined as follows:

1. Setup phase. $\mathcal{A}$ sets a limit for the size of integer list T.
2. First query phase. For $Poly(n)$ many times, the challenger $\mathcal{C}$ takes input messages in the form of $(user_i, \hat{P}_i)$-pairs for all the n users from $\mathcal{A}$, where $\hat{P}_i$ is the encrypted integer preference list of size T, and runs the UNS protocol with those inputs, and returns the output received from UNS. If the UNS

[2] the modification is that in the challenge phase, the challenger sends a tuple of two ciphertexts $(E(m_0), E(m_1))$ or $(E(m_0), E(m_1))$ depending on the tossed coin b, instead of sending encryption of one message that is chosen according to b

fails, i.e., there were more than $k = O(\log\ n)$ (so that one of the users' preference lists was completely consumed), then $\mathcal{C}$ receives FAIL and abort. Otherwise, $\mathcal{C}$ send to $\mathcal{A}$ the encrypted mixed location lists $\hat{P}_i$), of each user $i \in [n]$ (the list includes the "potential" mixed locations in each preference list that were not discarded).

3. Challenge phase. $\mathcal{A}$ selects two honest users p_0 and p_1 and two distinct integers λ_0 and λ_1 and sends them to the $\mathcal{C}$. Then, $\mathcal{C}$ creates two lists $\hat{P}_{0,1}, ..., \hat{P}_{0,n}$ and $\hat{P}_{1.1}, ..., \hat{P}_{1,n}$, corresponding to users p_0 and p_1, such that for each P_i^j for $j \in \{0,1\}, i \in [T]$, $\mathcal{C}$ tosses a coin $b \in \{0,1\}$ uniformally at random and sets $P_i^j = \lambda_b$.
4. Second query phase. Repeat the process as in the first query phase.
5. Output phase: $\mathcal{A}$ outputs a guess for b.

Let $\mathcal{A}^{(b)}$ be the output of $\mathcal{A}$, given that the challenger selected b. The adversary's advantage in the anonymity game is:

$$Adv_{\mathcal{A},\Pi}^{A-UNS}(\epsilon) := |Pr[\mathcal{A}^{(1)} = 1] - Pr[\mathcal{A}^{(0)} = 1]|$$

Definition 2. *UNS satisfies anonymity if it aborts or for all adversary $\mathcal{A}$, $Adv_{\mathcal{A},\Pi}^{A-UNS}(\epsilon) \leq negl(\epsilon)$.*

Theorem 4. *Protocol 1 satisfies UNS anonymity.*

Proof. By reduction from an CCA-secure encryption scheme, we show if an adversary wins the anonymity game as defined in the anonymity game, then there is a distinguisher D who can break the CCA-security of the encryption scheme that is used to encrypt the integers (as defined by the security game in [8]). The oracle plays the role of the servers and checks whether two ciphertexts are equal (via PET or secure hardware). Suppose that the adversary $\mathcal{A}$ can construct a PPT distinguisher D that has a non-negligible advantage as defined in the security game. $\mathcal{C}$ runs UNS with $\mathcal{A}$ who controls the corrupted users. $\mathcal{C}$ encrypts a preference list of size T as requested by $\mathcal{A}$. In the challenge phase, $\mathcal{A}$ request D to select two honest users p_0 and p_1, select two mixed integers l_a and l_b, and send them to the encryption challenger CE. CE tosses a random coin β and computes $(c_0, c_1) = (E_{pk}(l_a), E_{pk}(l_b))$ if $\beta = 0$ and $(c_0, c_1) = (E_{pk}(l_b), E_{pk}(l_a))$ if $\beta = 1$. $\mathcal{A}$ then computed two preference lists of size T according to the following process: for each list $j \in a, b$ and $i \in [T]$ the challenger selects a random r, tosses a random coin β', and rerandomizes accordingly, i.e., it computes $c_i^j = (A^r, B^r)$ if $\beta' = 0$ or $c_i^j = (A'^r, B'^r)$ if $\beta = 1$. Since we assume the run of UNS successfully ends, UNS finds two distinct integers for p_0 and p_1, which guarantees $\lambda(\beta) = l_a$ and $\lambda(1-\beta) = l_b$ for $\beta \in 0, 1$. We will henceforth assume that UNS does not fail. Since the candidates of integers in the list are selected independently from each other, $\mathcal{A}$ does not gain any information from the current candidates in $C(0)$ and $C(1)$ when it receives information on past collisions. Let $\lambda_i = UNS(i)$ be the chosen mixed integer for i. Since $\mathcal{A}$ knows whether λ_i is encryption of l_a or l_b by rerandomization from $E_{pk}(l_a)$ or $E_{pk}(l_b)$ (and the fact that rerandomization does not change the underlying plaintext), if $\mathcal{A}$ has the non-negligible probability to

win the anonymization game, then it has non-negligible probability to decrypt the current candidate in $C(0)$ and $C(1)$. Moreover, since $\mathcal{A}$ can simulate the view to D as defined in the CCA game, D can guess whether CE initially encrypted λ_a for p_0 and λ_b for p_1, or encrypted λ_b for p_0 and λ_a for p_1, breaking the CCA-security of encryption scheme E.

SMPP anonymity Recall that given message m_i mix location and λ_i of the i'th, it submits $\tilde{I} := I_i(m_i, \lambda_i) := (\hat{L}_i, \hat{V}_i, \tilde{E}(\lambda_i), ZKP_i)$ as input for the SMPP, where $\tilde{E}$ is an CCA encryption scheme, ZKP_i denote the ZK-proofs of the well-formedness of $\hat{L}_i, \hat{V}_i$ and $E(\lambda_i)$. Let $T[i]$ denote the output of SMPP for i. The anonymity game *Anon-SMPP* is defined as follows:

1. Setup phase. $\mathcal{A}$ sets a limit T for the size of the submitted vectors.
2. First query phase. $\mathcal{A}$ select two mixed locations λ_0, λ_1 As many times as $\mathcal{A}$ requests, $\mathcal{A}$ select a set of users S_1, and for each $i \in S_1$, it chooses a pair of messages and mixed location in the form of (λ_i, m_i), such that $\lambda_i otin \{\lambda_0, \lambda_1\}$ and sends it to $\mathcal{C}$. Then, $\mathcal{C}$ runs the SMPP protocol with S_1 those inputs and sends to $\mathcal{A}$ the SMPP input (the $\tilde{I}$'s) and output T.
3. Challange phase. $\mathcal{A}$ selects two honest users p_0 and p_1 and two distinct messages m_0 and m_1 and sends them to the $\mathcal{C}$. Then $\mathcal{C}$ tosses independently a uniform coin $b \in \{0, 1\}$ to set two mixed locations $\lambda_0 = b$ and $\lambda_1 = (1 - b)$, computes $\tilde{I}_0 := I_0(\lambda_0, m_0), \tilde{I}_1 := I_1\lambda, m_1)$
4. Second query phase. Repeat the process as in the first query phase for the remaining users, i.e., for $S \backslash S_1 \cap \{0, 1\}$. Then $\mathcal{C}$ computes $T = SMPP(\tilde{I}_0, ..., \tilde{I}_n)$ (so that $T[b] = E(m_0)$ and $T[1 - b] = E(m_1)$), and sends $I_0(\lambda_0, m_0), \tilde{I}_1, T$ to $\mathcal{A}$.
5. Output phase: The adversary outputs its guess b' for b.

Let $b' = \mathcal{A}^{\ b}$ be the output of $\mathcal{A}$, given that the challenger selected b.
Denote $Anon^{cca}_{\mathcal{A},\Pi}(\epsilon)$ be the outcome that the adversary correctly guesses b'
The adversary's advantage $Adv^{A-SMPP}_{\mathcal{A},\Pi}(\epsilon)$ in the anonymity game is: $|Pr[\mathcal{A}^1 = 1] - Pr[\mathcal{A}^0 = 1]|$.

Definition 3. *SMPP satisfies anonymity if for all adversary $\mathcal{A}$, ewline $Adv^{A-SMPP}_{\mathcal{A},\Pi}(\epsilon) \leq egl(\epsilon)$.*

Theorem 5. *Protocols 3 and 4 satisfy SMPP anonymity.*

Proof. By reduction from a CCA-secure encryption scheme and zero-knowledge proof (section D), we show if adversary $\mathcal{A}$ wins the anonymity game with non-negligible probability δ (as defined in *Anon-SMPP*), then there is a distinguisher D who has a non-negligible advantage to win the CCA game.

In the query phase of *Anon-SMPP*, $\mathcal{A}$ simulates all the users in S_1 (that do not include honest users p_0 and p_1). Since it knows the messages and their corresponding mixed locations, it has no problem simulating them. In the challenge phase of the CCA game, adversary D selects two messages m_0, m_1 and two mixed locations l_0, l_1 and sends it to a challenger of the CCA game $\mathcal{C}^{'}$. Then,

$\mathcal{C}'$ selects a bit $b \in \{0,1\}$ at random and sends $(c_0, c_1) = Enc(l_b), Enc(l_{1-b})$ to D. D sends (c_0, c_1) to $\mathcal{A}$, which is "sent" to the simulated $\mathcal{C}$. $\mathcal{C}$ tosses a uniform coin $\hat{b}$ and sets the corresponding inputs for $i \in \{0,1\}$, $\hat{L}_i := \hat{L}^{\lambda_{\hat{b}}}$ (based on (c_0, c_1)), a NIZK ZKP_i of well formation that corresponds to $\hat{L}_i, \hat{V}_i$ (note that the $\mathcal{C}$ can simulate a NIZK for the well formedness of $\hat{L}_i, \hat{V}_i$, but not NIZK that links $\hat{L}_i$ and $E(\lambda_i)$) and set $Inp_i := (\hat{L}_i, \hat{V}_i, ZKP_i)$. Note that $\hat{b}$ may not match $\hat{L}_i, \hat{V}_i$, where in this case the advantage of $\mathcal{A}$ may be neglible. In the second query phase, $\mathcal{A}$ sends to $\mathcal{C}$ pairs of (m_i, λ_i) for all $i \in S \backslash S_1 \cap \{0,1\}$. $\mathcal{A}$ outputs a guess bit b' for $\hat{b}$.

We now define a hybrid game (call it Hyb-A-SMPP) to be the same game as $ANON - SMPP$, except that now $\mathcal{C}$ does not provide a NIZK for linking $E(\lambda_i)$ and $\hat{L}_i$. Let ϵ_1 be the probabilty that $\mathcal{A}$ distinguish between games $Hyb - A - SMPP$ and $ANON - SMPP$. By the zero knowledge property of NIZK, we have

$$Pr(Anon_{\mathcal{A},\Pi}^{SMPP}(\epsilon) = 1) = Pr(Anon_{\mathcal{A},\Pi}^{Hyb-A-SMPP}(\epsilon) = 1) \tag{2}$$

We now argue that the additional information of the I_i's does not help $\mathcal{A}$ to win the game (guess b). In the first query phase, since $\mathcal{A}$ has to choose mix location λ_i that is different from the mixed locations selected in the challenge phase, it must send L_i such that $L_i[\lambda_0] = 0$ and $L_i[\lambda_1] = 0$, and $V_i[\lambda_0] = 0$ and $V_i[\lambda_1] = 0$. Therefore, sending appropriate queries does not add $\mathcal{A}$ with any extra information. The second query phase doesn't help $\mathcal{A}$ either. The reason is that the encryption of the mix location λ is CCA secure, hence if the $\mathcal{A}$ encrypts a queries a mixed location q (different from c_0 and c_1) then it cannot compute a new cipertext c_q as a function of c_0 and c_1. In addition, since the non-interactivne version of the NIZK (Sigma protocol) is based on the Fiat-Shamir tranformation, the zk-proof is non-malleable (in the ROM model) [27]. Since $\hat{L}_i, \hat{V}_i$ needs to be provided with a proof of well formation that is non-malleable, $(\hat{L}_i, \hat{V}_i, ZKP_i)$ is non-malleable (as mauling e.g., $\hat{L}_i$ requires also mauling ZKP_i). Since $\mathcal{A}$ succeeds in simulating a correct input for Hyb-SMPP, with probability $\frac{1}{2}$ (guessing the correct bit $\hat{b}$), we have

$$Pr(Anon_{\mathcal{A},\Pi}^{cca}(\epsilon) = 1) = Pr(Anon_{\mathcal{A},\Pi}^{HybSMM}(\epsilon) = \tag{3}$$

$$\frac{1}{2}(\frac{1}{2} + \delta) + \frac{1}{4} = \frac{1}{2} + \frac{\delta}{2} \geq \frac{1}{2} + negl(\epsilon) \tag{4}$$

Combining Eq.2 and Eq.3 gives $Anon_{\mathcal{A},\Pi}^{cca}(\epsilon) \geq \frac{1}{2} + negl(\epsilon)$

Anonymity of VSM For the anonymity of VSM, we use the same security game as Anon-SMPP. We argue that VSM satisfies anonymity. The reason is that the information that is exposed to the adversary in UNS, i.e., $E(\lambda_i)$, is a part of the input to SMPP. Therefore, given that UNS satisfies anonymity, the probability that the adversary will win the security game is the same as in SMPP, which is negl(ϵ).

Theorem 6. *Assume UNS satisfies anonymity and SMPP satisfies anonymity. Then the composed VSM protocol satisfies anonymity for a single epoch.*

Proof. The output of UNS is an encrypted list of pairwise distinct slots that is then used as part of the input to SMPP. By UNS anonymity, swapping the two challenge honest users changes only the hidden assignment of these encrypted slots. By SMPP anonymity, given such a swap, the adversary cannot distinguish which honest user wrote to which hidden slot. Therefore the sequential composition UNS→SMPP preserves anonymity in a single epoch.

F.2 Integrity

Integrity of UNS Since VSM is composed of two protocols UNS and SMPP, we will define integrity for each.

Let $Col(i, j) = 1$ be the event that UNS outputs two mixed location λ_i for user i and λ_i for user j, such that $\lambda_i = \lambda_j$.

Definition 4. *UNS maintains integrity if any two users $i, j \in [n]$, we have $Prob(Col(i, j) = 1) < negl(\epsilon)$.*

We now argue that the instantiations of UNS as described in section 5 maintain integrity.

Recall that after a successful run of UNS, it outputs a list of encrypted mixed locations $\hat{\lambda}_1, ..., \hat{\lambda}_n$ where each is signed by the servers. Suppose that UNS does not maintain integrity. Then there are at least are two encryptions of the same mixed location. Every pair of mixed locations output by UNS is checked for collision, which means that $TC(E(\lambda_1), E(\lambda_2) = 0$ although $\lambda_1 = \lambda_2$.

We now inspect the security of corrupted parties.

Corrupted users. Users ecnrypt their mixed location. The collision is checked either via PET or via trusted hardware. We claim that the event $TC(E(\lambda_1), E(\lambda_2)) = 0$ if $\lambda_1 = \lambda_2$ cannot happen. In the case of PET, this is proven to be secure [15], and in the case of TH, the ciphertexts are fully decrypted.

Corrupter Servers. In the case of corrupter servers that check collision with PET, the servers are required to publish a zero-knowledge proof (of knowledge) of the correctness of their execution. The soundness property of such proofs ensures that they cannot prove that two equal mixed locations are different except with negligible probability.

Corrupted TH-host. In the case that TC is checked via trusted hardware, the output of the TC is signed with a secure digital signature (existentially unforgeable under a chosen-message attack [32]) by the TH, which cannot be modified by the TH-host.

Integrity of SMPP

Definition 5. *Let $T = SMPP(I_1(m_1, \lambda_1), \ldots, I_n(m_n, \lambda_n))$ such that $\lambda_1 \neq \lambda_2 \neq \cdots \neq \lambda_n$. SMPP satisfies integrity if for every $j \in [n]$ we have $T(\lambda_j) = E_{pk}(m_j)$.*

We now argue that our instantiations of SMPP maintain integrity. In SMPP, each user i submits

$$I_i(m_i, \lambda_i) := (\hat{L}_i^{\lambda_i}, \hat{V}_i^{\lambda_i, m_i}, \Pi_i^V, E(\lambda_i), \Pi_i^T).$$

If the verification of all NIZKs passes for user i, then an extractor for the proof Π_i^T can extract the plaintexts λ_i and L_i and check that $L_{ij} = 0$ for all $j \neq \lambda_i$ and $L_{ij} = 1$ for $j = \lambda_i$. We now analyze the integrity of SMPP within the two instantiations.

1. In the case of ElGamal-based SMPP, the user submits a NIZK proof Π_i^{EG} (see Protocol 3) for $(\hat{L}_i^{\lambda_i}, \hat{V}_i^{\lambda_i, m_i})$. An extractor can extract the plaintexts of the vectors V_i and check that $V_{ij} = 1$ for all $j \neq \lambda_i$.
2. In the case of BGN-based SMPP, since BGN is somewhat homomorphic, the servers can test whether the submitted location vector $\hat{L}_i$ is a unit vector without a user-supplied NIZK. To do so, the servers implicitly check that $\sum_j r_j L_{ij}(1 - L_{ij}) = 0$. Since r_j is sampled from a large domain, the probability that this equation holds when $L_{ij} \notin \{0, 1\}$ is negligible. This is done by verifying that $\prod_j e(\hat{L}_{ij}, \hat{L}_{ij} \cdot E_1^{-1})^{r_j} = E(0)$. Given that all elements of L_i are bits, the server only needs to check that $\sum_j L_{ij} = 1$ to guarantee that L_i is a unit vector (equivalently, $\prod_j \hat{L}_{ij} \stackrel{?}{=} E(1)$). Then the scalar product gives $\sum_{i \in [n]} L_{ij} V_{ij} = m_i$, which is computed as $\prod_{i \in [n]} e(\hat{L}_{ij}, \hat{V}_{ij})$.

In both cases, given that each user has a unique mixed location, for all $j \in [n]$ we have $T(j) = \prod_i \hat{V}_{ij} = E(\sum_i V_{ij}) = E(M_i)$.

Integrity of VSM

Definition 6. *VSM satisfies integrity if for any input $\hat{P}$ such that $\lambda_1, ..., \lambda_n = UNS(\hat{P})$ and correpondig $T = SMPP(I_1(m_1, \lambda_1), ..., I_n(m_n, \lambda_n))$, we have $T(\lambda_j) = E_{pk}(m_j)$ for $j \in [n]$, except with neglible probability.*

We argue that VSM satisfies integrity. By the integrity of UNS, except with negligible probability, at the end of the execution of UNS, each user is assigned an (encrypted) unique number $\hat{\lambda}_i$. Then each user j sends $I(m_j, \lambda_j)$ that contains a zero-knowledge proof (proof of knowledge) π^T , which ensures that except with negligible probability, no attacker cann compute a valid NIZK for $\hat{\lambda}_i$ that does not correspond to $(\hat{L}_i, \hat{V}_i)$. Hence, given a successful run of VSM, except with negligible probability, each user provides input $I(m_i, \lambda_i)$ for a unique mixed location. Then, the integrity of VSM is followed by the integrity of SMPP.

F.3 Self Verifiability

Easy verifiablity captures the intuitive notion that any honest user i can verify that a protocol outputs a correct message corresponding to user i unconditionally, and without relying on any cryptographic assumption.

Theorem 7. *SMPP-Dec provides self-verifiability for every honest user.*

Proof. In SMPP-Dec the jointly decrypted table reveals the plaintext at every published slot. Since honest user i knows its own secret slot λ_i and original message m_i, it can verify correctness by checking whether the entry at position λ_i equals m_i. This check is purely local and does not rely on any additional cryptographic assumption. If the check fails, the published signed inputs allow the parties to identify which submitted tuple was malformed.