Non-Interactive Differentially Anonymous Router

Benedikt Bünz Yuncong Hu Shin'ichiro Matsuo Elaine Shi

September 18, 2021

Abstract

A recent work by Shi and Wu (Eurocrypt'21) suggested a new, non-interactive abstraction for anonymous routing, coined Non-Interactive Anonymous Router (NIAR). They show how to construct a NIAR scheme with succinct communication from bilinear groups. Unfortunately, the router needs to perform quadratic computation (in the number of senders/receivers) to perform each routing.

In this paper, we show that if one is willing to relax the security notion to (ϵ, δ)-differential privacy, henceforth also called (ϵ, δ)-differential anonymity, then, a non-interactive construction exists with subquadratic router computation, also assuming standard hardness assumptions in bilinear groups. Morever, even when $1 - 1/\mathsf{poly}\log n$ fraction of the senders are corrupt, we can attain strong privacy parameters where $\epsilon = O(1/\mathsf{poly}\log n)$ and $\delta = \mathsf{negl}(n)$.

Contents

1 Introduction **1**
1.1 Our Results and Contributions . . . 1
1.2 Technical Overview . . . 3
1.3 Additional Related Work . . . 5

2 Definitions and Preliminaries **5**
2.1 Syntax . . . 5
2.2 Computational Differential Anonymity . . . 6
2.3 Background on NIAR . . . 7

3 Two-Layer NIDAR Construction **8**
3.1 A Two-Step Permutation Algorithm . . . 8
3.2 Two-Layer NIDAR . . . 10
3.3 Proofs . . . 13
3.3.1 Sequence of Hybrids . . . 14
3.3.2 Differential Privacy Lemma . . . 18

4 Multi-Layer NIDAR **23**
4.1 Multi-Layer NIDAR Construction . . . 23
4.2 Recursive Subroutines . . . 25
4.3 Proofs . . . 28
4.3.1 Sequence of Hybrids . . . 28
4.3.2 Differential Privacy Lemma . . . 31

1 Introduction

Anonymous communication systems allow users to communicate in an insecure channel without leaking information about their identities or message contents. Interest in anonymous communication systems has increased in recent years because they promise a foundation for building anonymous data-sharing systems [BHKP16, HOWW19, HKP20] and anonymous cryptocurrency systems [CFN90, Cha82, BCG$^+$14, HAB$^+$17, HMPS14, RMSK14, RMSK17, Dia21, LYK$^+$19]. Various abstractions and techniques for anonymous communication systems have been proposed [DD08, EY09, SSA$^+$18], including mix-nets [Cha81, Abe99, BG12], the Dining Cryptographers' nets [Cha88, CGF10, APY20], onion routing [DMS04, GRS99], multi-server PIR-write [CBM15, OS97, GIKM00], as well as other variants [ZZZR05, vdHLZZ15, TGL$^+$17]. Notably, almost all of these anonymous routing schemes are *interactive* in nature, where multiple players or routers engage in some interactive protocol to accomplish the routing. Further, security typically relies on *threshold*-type assumptions, e.g., either majority or at least one of the routers must be honest for security to hold.

Interestingly, the very recent work by Shi and Wu [SW21] suggest a new, non-interactive abstraction called Non-Interactive Anonymous Router (NIAR), where routing is accomplished on a *single, untrusted* router. In their scheme, there are n senders and n receivers, and each sender wants to talk to a unique receiver. A trusted setup takes in the routing permutation π and generates secret sender and receiver keys for everyone, as well as a token tk for the untrusted router that secretly "encrypts" the routing permutation π. From this moment on, the n senders and n receivers can engage in multiple rounds of communication. In each time step, each sender uses its sender key to encrypt its message. When the router collects all n ciphertexts from the senders, it can apply the token tk and transform the n incoming ciphertexts into n *transformed and permuted* ciphertexts. At this moment, the receivers can each use their receiver key to decrypt its corresponding transformed ciphertext. Importantly, the router is applying the permutation π encoded in the token tk in an oblivious manner without learning what π is.

Shi and Wu show how to construct such a NIAR scheme assuming standard bilinear group assumptions. Their scheme achieves succinct communication: the total communication in each time step is only linear in n, i.e., each sender or receiver sends or receives only $O(1)$ amount of data. Unfortunately, although their NIAR scheme guarantees succinctness in communication, it is not so succinct in terms of router computation. Specifically, the router needs to perform $\Theta(n^2)$ computation to perform the routing in each time step. In this paper, we raise the following natural question:

Can we achieve privacy-preserving non-interactive routing with sub-quadratic router computation?

1.1 Our Results and Contributions

We show that if we are willing to relax the security notion to differential privacy [DMNS06], then indeed, we can achieve sub-quadratic computation. We call our construction Non-Interactive Differentially Anonymous Router (NIDAR). A NIDAR scheme has exactly the same syntax as the NIAR scheme of Shi and Wu [SW21], except that we now define a more relaxed security notion called (ϵ, δ)-*computational differential anonymity* (CDA), i.e., the analogy of (ϵ, δ)-differential privacy in the context of anonymous routing.

To define differential anonymity, we first define a notion of neighboring for permutations. Two permutations on the domain $[n]$ are said to be *neighboring*, iff their only difference is that two honest senders' destinations are swapped. Informally speaking, a NIDAR scheme is said to be differentially anonymous, iff no computationally bounded adversary can "effectively" distinguish two neighboring routing permutations. The actual definition is a little more technical since we need to take into account the fact that the adversary can corrupt a subset of the senders and receivers. Like in the work of Shi and Wu [SW21], we assume that each sender knows their own destinations, so if the adversary corrupts some senders, it will learn their destinations. Further, if the adversary corrupts a subset of the receivers, it naturally learns the messages received by the corrupt receivers in each time step, and this is inherent. Therefore, in our actual definition of (ϵ, δ)-CDA which is formalized in Section 2.2, we effectively consider two worlds. The only difference between the two worlds is that an honest pair of senders swap their destinations. Our security definition requires that for a computationally bounded adversary controlling the router and a subset of corrupt senders and receivers, these two worlds are (ϵ, δ)-indistinguishable (by the standard distance notion in the differential privacy literature [DMNS06, DR14, Vad17]), as long as the following conditions are respected: 1) each corrupt sender has the same receiver in both worlds, and 2) each corrupt receiver always receives the same message in both worlds. Note that these conditions are necessary to make sure that the adversary cannot trivially distinguish between the two worlds.

Specifically, we prove the following theorem — we will first give the version with the most general parameters, and then we will help the reader interpret the parameters with some typical choices.

Theorem 1.1. *Let $\rho \in (0,1)$ be the fraction of corrupt senders. Fix any desired $\delta \in (0,1)$ and any constant $L = O(1)$, fix any Z such that $(1-\rho)Z \geq C \cdot (\log \frac{1}{\delta} + \log n)$ where C is a sufficiently large constant. Then, assuming the hardness of the Decisional Linear assumption in suitable bilinear groups, there exists a* NIDAR *scheme that satisfy (ϵ, δ)-CDA for $\epsilon = O(1) \cdot \frac{\sqrt{Z \cdot \min((1-\rho), \rho)(\log \frac{1}{\delta} + \log n)} + (\log \frac{1}{\delta} + \log n)}{(1-\rho)Z}$, that satisfies the following asymptotical performance bounds where κ is a security parameter related to the strength of the hardness assumption:*

- *the router's computation per time step is $O(\kappa^L \cdot n^{1+1/L} \cdot Z^{1-1/L})$,*
- *the per-sender communication and encryption time is $O(k^L \cdot \mathsf{len})$ where len is the length of the plaintext message;*
- *each sender key is of length $O(\kappa \cdot n^{1/L} \cdot Z^{1-1/L})$; and*
- *each receiver key is of length $O(\kappa)$.*

Typical parameters choices. We now give a typical example to help the reader interpret the parameters. Below, we use poly_1 and poly_2 to denote potentially different polynomial functions.

Suppose that up to $1 - \frac{1}{\mathsf{poly}_1 \log n}$ fraction of the senders are corrupt, then, there exists a NIDAR scheme that is (ϵ, δ)-CDA for $\epsilon = \frac{1}{\mathsf{poly}_2 \log n}$ and $\delta = \mathsf{negl}(n)$, with the following asymptotical bounds where L denotes an any arbitrary constant and $\widetilde{O}$ hides polylogarithmic factors:

- the router's computation per time step is $\widetilde{O}(\kappa^L \cdot n^{1+1/L})$,
- the per-sender ciphertext size and encryption time is $O(k^L \cdot \mathsf{len})$ where len is the length of the plaintext message;
- each sender key is of length $\widetilde{O}(\kappa \cdot n^{1/L})$; and
- each receiver key is of length $O(\kappa)$.

Intuitively, this means that even when a very large fraction of senders are corrupt, we can still achieve $(\frac{1}{\mathsf{poly} \log n}, \mathsf{negl}(n))$-CDA. Recall that in the standard differential privacy literature, we typically want $\epsilon = O(1)$ and $\delta = \mathsf{negl}(n)$. Here we achieve something even better because our $\epsilon = o(1)$; and the smaller the ϵ, the more private it is.

1.2 Technical Overview

For simplicity, let us first consider a two-layer scheme that makes use of two onion layers of NIAR. We will assume that up to 99% fraction of the senders are corrupt for the time being, although in our technical sections later, we remove this assumption and give the most general parametrization.

Reducing routing to shuffling. Instead of constructing NIDAR directly, it is sufficient to construct a primitive where the router outputs the transformed ciphertexts destined for the n receivers in a random order, where the permutation is hidden in the (ϵ, δ)-CDA sense. Suppose that π is the actual permutation we want to realize, and π_{mid} is the secret random permutation applied by the shuffler, then the complement permutation π' is defined to be a permutation that satisfies $\pi' \circ \pi_{\text{mid}} = \pi$. We can give the complement permutation π' to the router in the clear, such that after the router applies the shuffler to the incoming ciphertexts, and obtains the randomly permuted and transformed ciphertexts, it can next apply π' to the permuted and transformed ciphertexts. The outcome will be in the order to be received by the n receivers, respectively.

Therefore, below we may focus on how to construct the differentially anonymous shuffler.

A two-step, matrix permutation algorithm. To overcome the quadratic router computation, we would like to break up the task of permuting n elements to roughly $O(\sqrt{n})$ permutations each of size only $\widetilde{O}(\sqrt{n})$. To achieve this, our idea is to arrange the elements in a square matrix, where each entry in the matrix is a bucket of polylogarithmic size. For technical reasons needed later, we need to introduce filler elements. Specifically, assume that initially, each bucket has exactly half real elements and half filler elements. Our goal is to output a random permutation of the real elements.

We perform the permutation in the following two steps:

1. *Row-wise permutations.* We permute each row in the matrix as follows. First, throw each real element into a random bucket (and if the bucket is full, simply abort throwing an overflow exception). Next, for each bucket in the row, pad each empty slot with a random, unconsumed filler element.
2. *Column-wise permutations.* For each column, output a random permutation of the real elements in the column, and throw away all the filler elements in the output.

Now, if we concatenate the outputs of all column-wise permutations, we obtain a permutation of the real elements in the input matrix. This permutation has negligible statistical distance from a uniform random one. In particular, it is not hard to show that if the buckets are of infinite size, then the output permutation would indeed be random. In our case, however, we can prove through standard measure concentration arguments that none of the buckets overflow except with negligible probability, as long as the buckets are polylogarithmic in size.

Two onion layers of NIAR. Our differentially anonymous shuffler scheme employs two onion layers of NIAR to realize the permutation in the above two-step manner. We invoke a NIAR instance for each row-wise and each column-wise permutation. Each sender will be encrypting two elements, a real element that encodes the message it wants to send, and a filler element that encodes no information but is necessary for security. For each of the real and filler element of the sender, the sender obtains two encryption keys, one corresponding to the column-wise NIAR instance, and one corresponding to the row-wise NIAR instance. When a sender encrypts its real or filler element, the encryption is performed in the reverse order: the inner encryption uses the sender key corresponding to the column-wise NIAR instance, and the inner ciphertext will be encrypted again using the sender key corresponding to the row-wise NIAR instance. The router's routing operation, on the other hand, is done in the forward order: it applies the NIAR tokens corresponding to all row-wise permutations first, and then applies the NIAR tokens corresponding to all column-wise permutations.

Performance analysis. With the above scheme, the router only needs to perform the routing operation for $O(\sqrt{n})$ many NIAR instances, each of size $\widetilde{O}(\sqrt{n})$. Recall that the routing cost of each NIAR instance is quadratic in the size of the instance. Thus, the total routing cost is $\widetilde{O}(n^{1.5})$. The two onion layers of encryption, however, incurs an additional blowup: in each layer, the plaintext message is encrypted bit by bit, and each bit encrypts to $O(1)$ bilinear group elements. Therefore, the ciphertext to plaintext ratio in each layer is some security parameter κ that is related to the length of a bilinear group element. With two layers of encryption, we will incur κ^2 blowup in the ciphertext size as well as the router computation.

Understanding the leakage. The key technical challenge in our proof is how to bound the leakage of the above two-layer onion construction. The issue is that the adversary can learn which buckets the corrupt sender's elements land in during the row-wise permutations, and as we explain below, this leaks a little information about how many honest real elements choose each bucket during the row-wise permutations.

Recall that in our random process, all the real elements choose their destinations first in the row-wise permutations, and then random filler elements are used to pad each bucket to its full capacity. If a bucket has fewer real elements, it will demand more fillers. Since fillers are randomly drawn among the set of real and corrupt fillers belonging to this row, chances are more corrupt fillers will choose that bucket. Since the adversary knows how many corrupt fillers land in each bucket during the row-wise permutations, it has a little leakage on the total number of real elements in that bucket. Since the adversary also knows how many corrupt real elements land in the bucket, it gets a little information about how many honest real elements land in the bucket.

Fortunately, despite this bit of leakage, we can prove that the scheme still satisfies a very strong notion of (ϵ, δ)-differential anonymity. As we explained earlier, our parameters give strong privacy guarantees particularly because we can achieve $\epsilon = 1/\mathsf{poly}\log n = o(1)$ even when $1 - 1/\mathsf{poly}\log n$ fraction of the senders may be corrupt, assuming a negligibly small δ.

Proof of differential anonymity. To show our scheme differentially anonymous, imprecisely speaking, we need to prove the following statements:

1. the adversary's view is simulatable, solely based on the leakage how many corrupt fillers go into each bucket — henceforth, we call this leakage the "corrupt filler load vector"; and
2. the corrupt filler load vector is (ϵ, δ)-differentially private for appropriate choices of ϵ and δ.

For the former statement, we go through a sequence of hybrids and rely on the security of the underlying NIAR scheme. Moreover, we need to rewrite the randomized experiment and change the order in which the events are sampled, without changing the nature of the randomized process. The actual proof is somewhat technical, so we defer the detailed explanation to subsequent technical sections.

For proving the second statement, we draw an interesting connection to a *database sampling mechanism* proposed by Chaudhuri and Mishra [CM06]. In their work, they consider a database where each element has an attribute. A random fraction of the database elements are sampled, and the frequency of each attribute is tallied and released. Chaudhuri and Mishra [CM06] show that in this sampling mechanism, the released histogram (i.e., frequency of all attributes) satisfies (ϵ, δ)-differential privacy for reasonable choices of ϵ and δ, as long as no individual attribute is too rare by some technical definition of "rare".

In our case, we can view the empty slots in each row (after throwing the real elements) as the database elements. The attribute of each element is the bucket it belongs to. Imagine we are sampling k elements where k corresponds to the number of corrupt fillers in this row. The adversary is then able to see the attributes of these sampled k elements. In other words, the adversary can see the number of corrupt fillers that go into each bucket in each row.

Unfortunately, we cannot directly use the analysis of Chaudhuri and Mishra [CM06]. They aim to make their proof work for the most general parameters, and as a result, their parameters are too loose for the special case we are interested in. Furthermore, their analysis makes some undesirable assumptions, e.g., the sampling probability must be at most $1/2$, and in our case, this roughly translates to the requirement that the majority of senders must be honest. Finally, jumping ahead a little, their analysis also does not exactly match the random process for the multi-layer scheme to be described later.

Instead of using their analysis, we present a new differential privacy analysis that is particularly optimized for the parameters we are interested in. In this way, we avoid the restrictions on the corruption threshold, and our analysis gives tighter bounds on the final ϵ and δ parameters. Again, we defer the detailed proof to the subsequent technical sections.

Extension: $O(1)$ layers of onions. We can further extend our construction to multiple layers. However, since each onion layer will incur a κ blowup in the ciphertext size, we can only support a constant number of layers.

To formally describe the L-layer scheme where $L = O(1)$, we use a recursive formulation. When the recursion is fully expanded out, it corresponds to routing on an R-way butterfly network $R = O(n^{1/L})$, and each atomic unit in this butterfly network is again a bucket of polylogarithmic size — see Figure 1 of Section 4 for a graphical illustration. Again, the senders perform encryption in the reverse order of the network, whereas the router's evaluation is performed in the forward order. We defer the technical details to Section 4.

1.3 Additional Related Work

Besides our work, differentially anonymous routing was also considered in private messaging systems such as Vuvuzela [vdHLZZ15], Stadium [TGL$^+$17], and Karaoke [LGZ18]. All of these prior works, however, are in the interactive setting, whereas we propose a non-interactive abstraction. In this sense, our security definitions are new. A few works on differential private information retrieval [TDG16, AIVG20] are also remotely related to our work, but again their abstractions are incomparable.

2 Definitions and Preliminaries

2.1 Syntax

A Non-Interactive Differentially Anonymous Router (NIDAR) has the same syntax as a Non-Interactive Anonymous Router (NIAR) which was first proposed by Shi and Wu [SW21] — the main difference from NIAR is in the security definition which we shall present in Section 2.2. Concretely, NIDAR is a cryptographic scheme consisting of the following, possibly randomized algorithms:

- $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}_\pi) \leftarrow \mathbf{Setup}(1^\lambda, n, \pi, \mathsf{len})$: the trusted **Setup** algorithm takes the security parameter 1^λ, the number of senders/receivers n, a permuation $\pi \in \mathsf{Perm}([n])$ that represents the mapping between the senders and the receivers, and the length of a plaintext message len. The **Setup** algorithm outputs a sender key for each sender denoted $\{\mathsf{ek}_u\}_{u\in[n]}$, a receiver key for each receiver denoted $\{\mathsf{rk}_u\}_{u\in[n]}$, and a token for the router denoted tk_π.
- $\mathsf{CT}_{u,t} \leftarrow \mathbf{Enc}(\mathsf{ek}_u, x_{u,t}, t)$: sender u uses its sender key ek_u to encrypt the message $x_{u,t}$ where $t \in \mathbb{N}$ denotes the current time step. The **Enc** algorithm produces a ciphertext $\mathsf{CT}_{u,t}$.
- $(\mathsf{CT}'_{1,t}, \mathsf{CT}'_{2,t}, \ldots, \mathsf{CT}'_{n,t}) \leftarrow \mathbf{Rte}(\mathsf{tk}_\pi, \mathsf{CT}_{1,t}, \mathsf{CT}_{2,t}, \ldots, \mathsf{CT}_{n,t})$: the routing algorithm **Rte** takes its token tk_π, and n ciphertexts received from the n senders denoted $\mathsf{CT}_{1,t}, \mathsf{CT}_{2,t}, \ldots, \mathsf{CT}_{n,t}$, and produces *transformed ciphertexts* $\mathsf{CT}'_{1,t}, \mathsf{CT}'_{2,t}, \ldots, \mathsf{CT}'_{n,t}$ where $\mathsf{CT}'_{u,t}$ is destined for the receiver $u \in [n]$.
- $x \leftarrow \mathbf{Dec}(\mathsf{rk}_u, \mathsf{CT}'_{u,t})$: the decryption algorithm **Dec** takes a receiver key rk_u, a transformed ciphertext $\mathsf{CT}'_{u,t}$, and outputs a decrypted message x.

In the above formulation, the permutation π is known a-priori at **Setup** time. Once **Setup** has been run, the senders can communicate with the receivers over an unbounded number of time steps t.

Correctness. Correctness requires that with probability $1 - \mathsf{negl}(\lambda)$, the following holds for any $\lambda \in \mathbb{N}$, any $\mathsf{len} \in \mathbb{N}$, any $(x_1, x_2, \ldots, x_n) \in (\{0,1\}^{\mathsf{len}})^n$, any $t \in \mathbb{N}$, and any permutation $\pi \in \mathsf{Perm}([n])$: let $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}_\pi) \leftarrow \mathbf{Setup}(1^\lambda, n, \pi, \mathsf{len})$, let $\mathsf{CT}_{u,t} \leftarrow \mathbf{Enc}(\mathsf{ek}_u, x_u, t)$ for $u \in [n]$, let $(\mathsf{CT}'_{1,t}, \mathsf{CT}'_{2,t}, \ldots, \mathsf{CT}'_{n,t}) \leftarrow \mathbf{Rte}(\mathsf{tk}_\pi, \mathsf{CT}_{1,t}, \mathsf{CT}_{2,t}, \ldots, \mathsf{CT}_{n,t})$, and let $x'_u \leftarrow \mathbf{Dec}(\mathsf{rk}_u, \mathsf{CT}'_{u,t})$ for $u \in [n]$; it must be that

$$x'_{\pi(u)} = x_u \text{ for every } u \in [n].$$

Communication compactness. We say that a NIDAR scheme satisfies communication compactness, iff the total communication cost per time step is upper bounded by $\mathsf{poly}(\lambda) \cdot O(n) \cdot \mathsf{len}$.

2.2 Computational Differential Anonymity

We now define computational differential anonymity (CDA). Consider the following experiment $\mathsf{NIDAR\text{-}Expt}^{b,\mathcal{A}}$ parametrized by a bit $b \in \{0,1\}$:

- $n, \mathcal{K}_S, \mathcal{K}_R, \pi^{(0)}, \pi^{(1)}, \mathsf{len} \leftarrow \mathcal{A}(1^\lambda)$
- $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}(1^\lambda, n, \pi^{(b)}, \mathsf{len})$
- For $t = 1, 2, \ldots$:
  - if $t = 1$ then $\{x^{(0)}_{u,t}\}_{u\in\mathcal{H}_S}, \{x^{(1)}_{u,t}\}_{u\in\mathcal{H}_S} \leftarrow \mathcal{A}(\mathsf{tk}, \{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{rk}_u\}_{u\in\mathcal{K}_R})$;
    else $\{x^{(0)}_{u,t}\}_{u\in\mathcal{H}_S}, \{x^{(1)}_{u,t}\}_{u\in\mathcal{H}_S} \leftarrow \mathcal{A}(\{\mathsf{CT}_{u,t-1}\}_{u\in\mathcal{H}_S})$;
  - for $u \in \mathcal{H}_S$, $\mathsf{CT}_{u,t} \leftarrow \mathbf{Enc}(\mathsf{ek}_u, x^{(b)}_{u,t}, t)$
- The adversary $\mathcal{A}$ outputs $b' \in \{0,1\}$, and the experiment also outputs b'.

We say that $\mathcal{A}$ is admissible iff with probability 1, it guarantees that

1. there exist $u \in \mathcal{H}_S$ and $v \in \mathcal{H}_S$ such that

- $u \neq v$;
- $\pi^{(0)}(u) = \pi^{(1)}(v)$ and $\pi^{(0)}(v) = \pi^{(1)}(u)$;
- for any $k \in [n]$, where $k \neq u$ and $k \neq v$, $\pi^{(0)}(k) = \pi^{(1)}(k)$;

2. for any $u \in \mathcal{K}_R \cap \pi^{(0)}(\mathcal{H}_S) = \mathcal{K}_R \cap \pi^{(1)}(\mathcal{H}_S)$, $x^{(0)}_{v_0,t} = x^{(1)}_{v_1,t}$ where for $b \in \{0,1\}$, $v_b{:=}(\pi^{(b)})^{-1}(u)$. In other words, here we require that in the two alternate worlds $b = 0$ or 1, every corrupt receiver receiving from an honest sender must receive the same message.

Definition 1 (Computational differential anonymity)**.** Let $\epsilon > 0$ and $\delta \in (0,1)$ be functions of the security parameter λ. We say that a NIDAR scheme satisfies (ϵ, δ)-*computational differential anonymity (or* (ϵ,δ)*-CDA for short)*, iff for every nonuniform p.p.t. adversary $\mathcal{A}$, for every $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\mathsf{NIDAR\text{-}Expt}^{0,\mathcal{A}}(1^\lambda) = 1\right] \leq e^{\epsilon(\lambda)} \times \Pr\left[\mathsf{NIDAR\text{-}Expt}^{1,\mathcal{A}}(1^\lambda) = 1\right] + \delta(\lambda) \ .$$

2.3 Background on NIAR

In our NIDAR construction, we will use two or more onion layers of NIAR. A NIAR scheme has the same syntax as NIDAR (see Section 2.1), but satisfies a stronger, simulation-based security notion as defined below.

We consider static corruption where the set of corrupt players are chosen prior to the **Setup** algorithm.

Real-world experiment $\mathsf{Real}^{\mathcal{A}}(1^\lambda)$**.** The real-world experiment is described below where $\mathcal{K}_S \subseteq [n]$ denotes the set of corrupt senders, and $\mathcal{K}_R \subseteq [n]$ denotes the set of corrupt receivers. Let $\mathcal{H}_S = [n] \setminus \mathcal{K}_S$ be the set of honest senders and $\mathcal{H}_R = [n] \setminus \mathcal{K}_R$ be the set of honest receivers. Let $\mathcal{A}$ be a *stateful* adversary:

- $n, \pi, \mathcal{K}_S, \mathcal{K}_R, \mathsf{len} \leftarrow \mathcal{A}(1^\lambda)$
- $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}(1^\lambda, n, \pi, \mathsf{len})$
- For $t = 1, 2, \ldots$:
  - if $t = 1$ then $\{x_{u,t}\}_{u\in\mathcal{H}_S} \leftarrow \mathcal{A}(\mathsf{tk}, \{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{rk}_u\}_{u\in\mathcal{K}_R})$; else $\{x_{u,t}\}_{u\in\mathcal{H}_S} \leftarrow \mathcal{A}(\{\mathsf{CT}_{u,t-1}\}_{u\in\mathcal{H}_S})$;
  - for $u \in \mathcal{H}_S$, $\mathsf{CT}_{u,t} \leftarrow \mathbf{Enc}(\mathsf{ek}_u, x_{u,t}, t)$

Ideal-world experiment $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}(1^\lambda)$**.** The ideal-world experiment involves not just $\mathcal{A}$, but also a p.p.t. (stateful) simulator denoted Sim, who is in charge of simulating $\mathcal{A}$'s view knowing essentially only what corrupt senders and receivers know. Further, the $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}(1^\lambda)$ experiment is parametrized by a leakage function denoted Leak to be defined later. Henceforth for $\mathcal{C} \subseteq [n]$, we use $\pi(\mathcal{C})$ to denote the set $\{\pi(u) : u \in \mathcal{C}\}$.

- $n, \pi, \mathcal{K}_S, \mathcal{K}_R, \mathsf{len} \leftarrow \mathcal{A}(1^\lambda)$
- $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}) \leftarrow \mathsf{Sim}(1^\lambda, n, \mathsf{len}, \mathcal{K}_S, \mathcal{K}_R, \mathsf{Leak}(\pi, \mathcal{K}_S, \mathcal{K}_R))$
- For $t = 1, 2, \ldots$:
  - if $t = 1$ then $\{x_{u,t}\}_{u\in\mathcal{H}_S} \leftarrow \mathcal{A}(\mathsf{tk}, \{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{rk}_u\}_{u\in\mathcal{K}_R})$; else $\{x_{u,t}\}_{u\in\mathcal{H}_S} \leftarrow \mathcal{A}(\{\mathsf{CT}_{u,t-1}\}_{u\in\mathcal{H}_S})$;
  - $\{\mathsf{CT}_{u,t}\}_{u\in\mathcal{H}_S} \leftarrow \mathsf{Sim}\left(\{\forall u \in \mathcal{K}_R \cap \pi(\mathcal{H}_S) : (u, x_{v,t}) \text{ for } v = \pi^{-1}(u)\}\right)$.

In the above the function $\mathsf{Leak}(\pi, \mathcal{K}_S, \mathcal{K}_R)$ constains the destination of each corrupt sender, as defined below:

$$\mathsf{Leak}^{\mathrm{S}}(\pi, \mathcal{K}_S, \mathcal{K}_R) := \{\forall u \in \mathcal{K}_S : (u, \pi(u))\}$$

Definition 2 (NIAR simulation security)**.** We say that a NIAR scheme satisfies simulation security (with receiver insider protection), iff there exists a p.p.t. simulator Sim such that for any non-uniform p.p.t. adversary $\mathcal{A}$, $\mathcal{A}$'s view in $\mathsf{Real}^{\mathcal{A}}(1^\lambda)$ and $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}(1^\lambda)$ are computationally indistinguishable.

Note that the above simulation-secure definition is equivalent to $(0, \mathsf{negl}(\lambda))$-CDA. In particular, Shi and Wu [SW21] proved that the simulation-based security notion is equivalent to a natural indistinguishability-based definition; and their indistinguishability-based notion is equivalent to $(0, \mathsf{negl}(\lambda))$-CDA due to a simple hybrid argument, since given any two permutations $\pi^{(0)}$ and $\pi^{(1)}$, we can transform $\pi^{(0)}$ to $\pi^{(1)}$ in polynomially many steps, each time swapping the destinations of two honest senders.

3 Two-Layer NIDAR Construction

3.1 A Two-Step Permutation Algorithm

We want to anonymously permute n elements. However, recall that a NIAR scheme on n elements would incur at least n^2 computational cost to route n messages in each time step. To reduce the computational cost, our idea is to break up the big permutation on n elements into roughly $\sqrt{n}$ permutations each of size $\sqrt{n}$.

Strawman attempt. A strawman attempt is to write the n elements as a matrix — we first permute each row, and then permute each column. Unfortunately, it turns out that this does not result in a random permutation. More specifically, elements originally in the same row must all go to distinct columns. Based on this, a polynomial time adversary could easily distinguish the resulting permutation from a completely random one.

Our approach. Our approach is inspired by this strawman attempt. However, we make two critical modifications. First, we introduce as many filler elements as there are real elements. Second, we will work on a matrix of buckets, i.e., each entry in the matrix is now a bucket of size Z. We will prove that if Z is at least polylogarithmic in size, then our algorithm produces a permutation that has negligible statistical distance from a uniform random one. We will describe our MatrixPerm algorithm for an input of $2n$ elements rather than n, since later, in our NIDAR scheme, we always invoke MatrixPerm on $2n$ elements, where each of the n senders contributes one *real* element and one *filler* element. For simplicity, the reader may first imagine that the matrix is a square one, i.e., $R = C$ and $R \cdot C \cdot Z = 2n$, although in the case when $2n/Z$ is not a perfect square, R and C do not need to be strictly equal.

Algorithm MatrixPerm

Input. The input $\mathbf{I}$ is an array of $2n$ elements among which at least half are fillers, and the remaining are real elements. View the input as an $(R \times C)$-sized matrix (also denoted $\mathbf{I}$) where each entry in the matrix $\mathbf{I}[i, j]$ is a bucket consisting of Z elements, such that *at least half of the elements in each bucket are fillers.* For simplicity, we assume that $R \cdot C \cdot Z = 2n$ (we will explain how to deal with the case when $2n$ is not divisible by $R \cdot Z$ later).

Algorithm.

1. **Permute rows.** For each row $i \in [R]$, let $\mathbf{I}[i, :] = \mathsf{RowPerm}(\mathbf{I}[i, :])$.
2. **Permute columns.** For each column $j \in [C]$, let $\mathbf{I}[:, j] = \mathsf{ColPerm}(\mathbf{I}[:, j])$.
3. **Output.** For each column $j \in [C]$, output all the real elements in column j in lexicographical ordering of their offset β within the column (and ignore all the filler elements).

Subroutine RowPerm

Input. A list of C buckets each of size Z, and each bucket contains at least half filler elements.

Algorithm.

1. Initialize C output buckets each of capacity Z. Initially, all output buckets are empty.
2. For each real element in the input list, place it into a random output bucket.
3. If any bucket's load exceeds Z, return overflow; else, for each empty slot in each bucket, choose a random unconsumed filler element from the input array to fill the slot.
4. Randomly permute the elements within each bucket.
5. Return the list of output buckets.

Subroutine ColPerm

Input. A list of R buckets each of size Z.

Algorithm.

1. Let X be the list of all real elements contained in the input, and let Y be the list of all filler elements in the input.
2. Return $\mathsf{RandPerm}(X) || \mathsf{RandPerm}(Y)$ where $\mathsf{RandPerm}(\cdot)$ outputs a random permutation of the input array.

Lemma 3.1 (MatrixPerm $\approx$ uniform random permutation)**.** *The output of the* $\mathsf{MatrixPerm}(\mathbf{I})$ *algorithm outputs a permutation of the real elements contained in the input* $\mathbf{I}$*, and moreover, the resulting permutation has statistical distance at most* $O(n) \cdot \exp(-\Omega(Z))$ *from a uniform random permutation.*

Proof. First, pretend that in our algorithm, even if some bucket receives more than Z real elements during the row-wise permutations, we do not abort throwing overflow, but instead continue with the algorithm allowing the buckets to contain arbitrarily many elements. In this case, it is not hard to see that the algorithm must output a uniform random permutation. Therefore, it suffices to prove that the probability of overflow is upper bounded by $O(n) \cdot \exp(-\Omega(Z))$. This follows from a simple application of the Chernoff bound for each fixed bucket, and then taking a union bound over all buckets. $\square$

More general parameters. So far, we have assumed that $R \cdot C \cdot Z = 2n$. However, in some cases, $2n$ may not be divisible by $R \cdot Z$. In this case, we may distribute the input elements as evenly as possible across all rows, and as evenly as possible across the buckets in the same row. Later in our application of MatrixPerm, each of the n senders contributes a real and a filler element, and we want that for the same sender, its real and filler elements be assigned to the same bucket in the input. Therefore, when $2n$ is not divisible by $R \cdot Z$, we may assume each row has either Q or $Q+2$ elements for some Q, and each bucket has either Z elements or $Z+2$ elements. Like before, we still require that each bucket has at least as many filler elements as there are real elements. Lemma 3.1 would still hold for this indivisible case as well.

3.2 Two-Layer NIDAR

In our construction, we will implement a permutation using the two-step MatrixPerm algorithm described in Section 3.1. Moreover, each sender encrypts one *real* element and one *filler* element, in a two-layer onion fashion as we explain in more detail below. The real element corresponds to the message the sender actually wants to send, whereas the filler element does not encode any useful content, and is only introduced for security. Recall that in MatrixPerm, we divide the input elements, symbolically denoted (1R, 1F, 2R, 2F, $\ldots$, nR, nF) into a matrix of $R \times C$ buckets, each of size Z — here, for $u \in [n]$, uR denotes sender u's real element, and uF denotes sender u's real element. We first randomly permute each row; then, we randomly permute each column while moving all the real elements within each column to the front. In our NIDAR construction, we will invoke an NIAR instance for each of the R row-wise permutations, and similarly, invoke an NIAR instane for each of the C column-wise permutations. The column instances of NIAR will be used to encrypt the actual messages, whereas the row instances of NIAR will be used to encrypt the ciphertexts produced by the column instances, thus creating a two-layer onion.

Notational conventions. We always use the variable $i \in [R]$ to index into rows, and the variable $j \in [C]$ to index into columns. When we write (i, α) where $i \in [R]$ and $\alpha \in [C \cdot Z]$, we refer to the α-th *element* (as opposed to bucket) of the i-th row, when the C buckets in row i is flattened out as a one-dimensional array. Similarly, when we write (j, β) where $j \in [C]$ and $\beta \in [R \cdot Z]$, we refer to the β-th *element* (as opposed to bucket) of the j-th column. We often use the superscripts "$-$" and "$|$" to differentiate between variables of the row instances and variables of the column instances.

Our two-layer NIDAR construction. We now describe our two-layer NIDAR construction below.

Two-Layer NIDAR Construction

Parameters: let Z be the bucket size such that $(1-\rho)Z \geq \Theta(\log \frac{1}{\delta})$ where ρ is the fraction of corrupt senders, and $\Theta(\cdot)$ hides an appropriately large constant. For simplicity, assume that $2n = R \cdot C \cdot Z$ and $R = C$. We will deal with the case when $2n/Z$ is not a perfect square or $2n$ is not divisible by Z later.

Assume: after the adversary chooses which users to corrupt and before the **Setup** algorithm is first invoked, all senders are randomly permuted, and we renumber the senders from 1 to n after this initial permutation. Throughout the following algorithms, we refer to senders by these randomly renumbered identities.

- **Setup**$(1^\lambda, n, \pi, \mathsf{len})$:

1. *Simulate* $\mathsf{MatrixPerm}$. Simulate a random run of the $\mathsf{MatrixPerm}$ algorithm on the symbolic array (1R, 1F, 2R, 2F, ..., nR, nF), where uR and uF denote the real and filler elements corresponding to sender $u \in [n]$, respectively. Let π_i^{-} denote the permutation applied to row i in $\mathsf{RowPerm}$, and let $\pi_j^{|}$ denote the permutation applied to column j in $\mathsf{ColPerm}$. Let π_{mid} be the permutation output by $\mathsf{MatrixPerm}$ on the real elements in the input. Let $m_1, m_2, \ldots, m_C$ denote the number of real elements in each column after the row-wise permutations. Let len' denote the ciphertext length of a NIAR scheme (with $R \cdot Z$ senders) when the message length is len.
2. *Set up row instances of* NIAR. For each row $i \in [R]$, let
$$\left(\{\mathsf{ek}_{i,\alpha}^{-}\}_{\alpha\in[C\cdot Z]}, \{\mathsf{rk}_{i,\alpha}^{-}\}_{\alpha\in[C\cdot Z]}, \mathsf{tk}_i^{-}\right) \leftarrow \mathsf{NIAR}.\mathbf{Setup}(1^\lambda, C\cdot Z, \pi_i^{-}, \mathsf{len}')$$
3. *Set up column instances of* NIAR. For each column $j \in [C]$, let
$$\left(\{\mathsf{ek}_{j,\beta}^{|}\}_{\beta\in[R\cdot Z]}, \{\mathsf{rk}_{j,\beta}^{|}\}_{\beta\in[R\cdot Z]}, \mathsf{tk}_j^{|}\right) \leftarrow \mathsf{NIAR}.\mathbf{Setup}(1^\lambda, R\cdot Z, \pi_j^{|}, \mathsf{len})$$
4. *Output sender keys.*
   - suppose that in the $\mathsf{MatrixPerm}$ algorithm earlier, the symbolic elements uR and uF are initially in positions (i, α) and $(i, \alpha+1)$, respectively[a]; moreover, after the row-wise permutations, they are in positions (j, β) and $(\widetilde{j}, \widetilde{\beta})$, respectively.
   - give user u the encryption key $\mathsf{ek}_u := (\mathsf{ek}_{i,\alpha}^{-}, \mathsf{ek}_{j,\beta}^{|}, \mathsf{ek}_{i,\alpha+1}^{-}, \mathsf{ek}_{\widetilde{j},\widetilde{\beta}}^{|})$.
5. *Output receiver keys.* Let $(\mathsf{rk}_1, \ldots, \mathsf{rk}_n) := \pi'\left(\{\mathsf{rk}_{j,\beta}^{|}\}_{j\in[C],\beta\in[m_j]}\right)$ where $\{\mathsf{rk}_{j,\beta}^{|}\}_{j\in[C],\beta\in[m_j]}$ is flattened to a 1-dimensional array based on lexicographical order of (j, β).
6. *Output token.* Let π' be the "complement permutation" such that $\pi' \circ \pi_{\text{mid}} = \pi$; output
$$\mathsf{tk} := (\pi', \{m_j\}_{j\in[C]}, \{\mathsf{tk}_i^{-}\}_{i\in[R]}, \{\mathsf{tk}_j^{|}\}_{j\in[C]}, \{\mathsf{rk}_{i,\alpha}^{-}\}_{i\in[R],\alpha\in[C\cdot Z]})$$

- $\mathbf{Enc}(\mathsf{ek}_u, x_{u,t}, t)$:
  1. parse $\mathsf{ek}_u := (\mathsf{ek}^{-}, \mathsf{ek}^{|}, \mathsf{fek}^{-}, \mathsf{fek}^{|})$;
  2. let $\mathsf{ict} \leftarrow \mathsf{NIAR}.\mathbf{Enc}(\mathsf{ek}^{|}, x_{u,t}, t)$; and let $\mathsf{ct} \leftarrow \mathsf{NIAR}.\mathbf{Enc}(\mathsf{ek}^{-}, \mathsf{ict}, t)$;
  3. let $\widetilde{\mathsf{ict}} \leftarrow \mathsf{NIAR}.\mathbf{Enc}(\mathsf{fek}^{|}, \mathbf{0}, t)$; and let $\widetilde{\mathsf{ct}} \leftarrow \mathsf{NIAR}.\mathbf{Enc}(\mathsf{fek}^{-}, \widetilde{\mathsf{ict}}, t)$; and
  4. output $\mathsf{CT} := (\mathsf{ct}, \widetilde{\mathsf{ct}})$.

- $\mathbf{Rte}(\mathsf{tk}, \mathsf{CT}_{1,t}, \ldots, \mathsf{CT}_{n,t})$:
  1. for each $u \in [n]$, parse $\mathsf{CT}_{u,t} := (\mathsf{ct}_{u,t}, \widetilde{\mathsf{ct}}_{u,t})$; view $\{\mathsf{CT}_{u,t}\}_{u\in[n]}$ as an $(R \times C)$-matrix where each entry is a bucket of size Z. Henceforth, we use $\mathsf{CT}[i :]$ to denote the i-th row of this ciphertext matrix.
  2. parse $\mathsf{tk} := (\pi', \{m_j\}_{j\in[C]}, \{\mathsf{tk}_i^{-}\}_{i\in[R]}, \{\mathsf{tk}_j^{|}\}_{j\in[C]}, \{\mathsf{rk}_{i,\alpha}^{-}\}_{i\in[R],\alpha\in[C\cdot Z]})$;
  3. for each row $i \in [R]$, let $\overline{\mathsf{ict}}_{i,1}, \ldots, \overline{\mathsf{ict}}_{i,C\cdot Z} \leftarrow \mathsf{NIAR}.\mathbf{Rte}(\mathsf{tk}_i^{-}, \mathsf{CT}[i :])$, and for each $\alpha \in [C \cdot Z]$, let $\mathsf{ict}_{i,\alpha} \leftarrow \mathsf{NIAR}.\mathbf{Dec}(\mathsf{rk}_{i,\alpha}^{-}, \overline{\mathsf{ict}}_{i,\alpha})$;

4. view $\{\mathsf{ict}_{i,\alpha}\}_{i\in[R],\alpha\in[C\cdot Z]}$ also as a $(R\times C)$-matrix where each entry is a bucket of size Z — we shall use $\mathsf{ict}[:j]$ to denote the j-th column of this matrix;
5. for each column $j\in[C]$, let $\mathsf{CT}'_{j,1},\dots,\mathsf{CT}'_{j,R\cdot Z}\leftarrow \mathsf{NIAR}.\mathbf{Rte}\left(\mathsf{tk}^{|}_{j},\mathsf{ict}[:j]\right)$;
6. view $\{\mathsf{CT}'_{j,\beta}\}_{j\in[C],\beta\in[m_j]}$ as an array, apply π' to the array, and output the result.

- $\mathbf{Dec}(\mathsf{rk}_u,\mathsf{CT}'_u)$: output $\mathsf{NIAR}.\mathbf{Dec}(\mathsf{rk}_u,\mathsf{CT}'_u)$.

[a]We may assume that uR and uF must be in adjacent positions in the same row initially.

More general parameters. So far, we assumed that $R\cdot C\cdot Z=2n$ and $R=C$. In the more general case, $2n$ may not be divisible by Z, or $2n/Z$ may not be a perfect square. In this case, we can choose $R=\left\lceil\sqrt{2n/Z}\right\rceil$, and the above algorithm would still work, as long as when we assign the elements 1R, 1F, …, nR, nF to the initial matrix of buckets, the following constraints are respected:

1. For any $u\in[n]$, uR and uF are always assigned to the same bucket;
2. All rows' total capacities (of real and filler elements) differ by at most 2;
3. All buckets' total capacities (of real and filler elements) differ by at most 2. In other words, not all buckets are of equal capacity Z; the capacity could be either Z or $Z+2$.

The latter two constraints basically says that the loads across all rows and the loads across all buckets within the same row should be as even as possible, subject to the first constraint.

When we have fixed the capacities of all buckets in this $R\times C$ matrix as mentioned above, we can adjust the size parameter of each row-wise and column-wise NIAR instance accordingly. Our proof can easily be modified to make this case work as well.

Correctness. Fix any security parameter $\lambda\in\mathbb{N}$, any message length $\mathsf{len}\in\mathbb{N}$, any plaintext messages $(x_1,x_2,\dots,x_n)\in(\{0,1\}^{\mathsf{len}})^n$, any timestamp $t\in\mathbb{N}$, and any permutation $\pi\in\mathsf{Perm}([n])$. Let $(\{\mathsf{ek}_u\}_{u\in[n]},\{\mathsf{rk}_u\}_{u\in[n]},\mathsf{tk}_\pi)$ be any key tuples output by the algorithm $\mathbf{Setup}(1^\lambda,n,\pi,\mathsf{len})$, let CT_u be the ciphertext of x_u output by the algorithm $\mathbf{Enc}(\mathsf{ek}_u,x_u)$ for $u\in[n]$, let $(\mathsf{CT}'_1,\mathsf{CT}'_2,\dots,\mathsf{CT}'_n)$ be the shuffled ciphertext output by the algorithm $\mathbf{Rte}(\mathsf{tk}_\pi,\mathsf{CT}_1,\mathsf{CT}_2,\dots,\mathsf{CT}_n)$, and let x'_u be the decryption result output by the algorithm $\mathbf{Dec}(\mathsf{rk}_u,\mathsf{CT}'_u)$ for $u\in[n]$.

We need to show that $x_u=x'_{\pi}(u)$ for $u\in[n]$. According to the proof of Lemma 3.1, with probability $1-\mathsf{negl}(\lambda)$, the algorithm **Setup** succeeds in simulating MatrixPerm algorithm and produces corresponding key tuples. We first consider the correctness of the row-wise permutation. For a user $u\in[n]$, suppose the symbolic elements uR and uF in the MatrixPerm are initially in positions (i,α) and $(i,\alpha+1)$ ($i\in[R]$ and $\alpha\in[C\cdot Z]$), and are in positions (j,β) and $(\widetilde{j},\widetilde{\beta})$ ($j,\widetilde{j}\in[C]$ and $\beta,\widetilde{\beta}\in[R\cdot Z]$), after the row-wise permutation. The user u receives the encryption key $\mathsf{ek}^{-}_{i,\alpha}$ and $\mathsf{ek}^{-}_{i,\alpha+1}$ for the outer encryption in the two-layer onion. Due to the correctness of the NIAR instance for the row i, after running the algorithms $\mathsf{NIAR}.\mathbf{Rte}$ and $\mathsf{NIAR}.\mathbf{Dec}$, with probability 1, the inner ciphertext of the user u's real and filler messages will be in positions (j,β) and $(\widetilde{j},\widetilde{\beta})$, respectively.

Now, we consider the correctness of the column-wise permutation. Similarly, the user u receives the encryption key $\mathsf{ek}^{|}_{j,\beta}$ and $\mathsf{ek}^{|}_{\widetilde{j},\widetilde{\beta}}$ for the inner encryption in the two-layer onion. Suppose the symbolic element uR is in the position (j',β') ($j'\in[C]$ and $\beta'\in[m_j]$) after the column-wise permutation. Due to the correctness of the NIAR instance for the column j, after running the algorithm $\mathsf{NIAR}.\mathbf{Rte}$, with probability 1, the intermediate result of the inner ciphertext will be in the position (j', β').

Let $\{\mathsf{CT}'_{j,\beta}\}_{j\in[C],\beta\in[m_j]}$ be the array of ciphertexts after the column permutation. Then the $\pi_{\text{mid}}(u)$-th element is the ciphertext of the user u's real message and will be send to the receiver $\pi' \circ \pi_{\text{mid}}(u)$. Due to the correctness of the NIAR instance for the column j, after running the algorithm $\mathsf{NIAR}.\mathbf{Dec}$, with probability 1, the decryption result $x'_{\pi'\circ\pi_{\text{mid}}(u)}$ equals to the message x_u. Because $\pi' \circ \pi_{\text{mid}} = \pi$, we have $x'_{\pi(u)} = x_u$.

Efficiency. We now analyze the efficiency of our NIDAR scheme assuming that the underlying NIAR is instantiated with the construction of Shi and Wu [SW21]. We first review the efficiency of the NIAR scheme by Shi and Wu. We will use the notation $O_\lambda(\cdot)$ to hide $\mathsf{poly}(\lambda)$ factors. In the underlying NIAR scheme by Shi and Wu [SW21], the per-sender ciphertext length and per-sender computation in each time step are upper bounded by $O_\lambda(\mathsf{len})$; each sender's key is at most $O_\lambda(n)$ in size; each receiver's key is $O_\lambda(1)$ in size; the router's token has length $O_\lambda(n^2)$; and finally, the computational overhead for performing the $\mathbf{Rte}$ operation is $O_\lambda(n^2)$.

In our NIDAR scheme, each sender needs to compute $O(1)$ many NIAR ciphertexts in every time step, which takes $O_\lambda(\mathsf{len})$ amount of time, and moreover, the ciphertext size (per sender) is upper bounded by the same expression. Note that the row instances of NIAR have message lengths that are polynomially larger than the column instances, and this polynomial blowup is accounted for in the $O_\lambda(\cdot)$ notation. It is also not hard to verify that each sender's key is $O_\lambda(R\cdot Z + C\cdot Z) = O_\lambda(\sqrt{nZ})$ in size, and each receiver's key is $O_\lambda(1)$ in size.

We now analyze the computational overhead of the $\mathbf{Rte}$ operation as well as the router's token size. To perform the $\mathbf{Rte}$ operation, the router needs to evaluate the underlying NIAR's $\mathbf{Rte}$ function for R row instances each with $C\cdot Z$ senders, and for C column instances each with $R\cdot Z$ senders. Therefore, the router's total work is upper bounded by $O_\lambda(R\cdot(C\cdot Z)^2 + C\cdot(R\cdot Z)^2) = O_\lambda(\sqrt{n/Z}\cdot(\sqrt{n/Z}\cdot Z)^2) = O_\lambda(n^{\frac{3}{2}}\cdot Z^{\frac{1}{2}})$. Again, the $O_\lambda(\cdot)$ notation accounts for the polynomial blowup in the plaintext size for the row instances. The router's token size is also upper bounded by the same expression, that is, $O_\lambda(n^{\frac{3}{2}}\cdot Z^{\frac{1}{2}})$.

3.3 Proofs

Recall that in our NIDAR construction, we first randomly permute all the users at the beginning of **Setup**, and reassign their identities after this random permutation. This random permutation step is there only to make sure that corruption choices are random. Therefore, henceforth in the proof, we may equivalently pretend that the corruption choices are randomly made, and we skip this random permutation step in the algorithm.

Suppose we start out with the symbolic vector (1R, 1F, 2R, 2F, ..., nR, nF) where each index $u\in[n]$ denotes a user, the letter "R" denotes a real message, and "F" denotes a filler message. A random subset of these users are corrupt. We now view this vector as a matrix of $R\times C$ buckets each of size Z, and we permute this vector using the $\mathsf{MatrixPerm}$ algorithm, where we first apply a row-wise permutation to each row of buckets, and then we apply a column-wise permutation to each column of buckets. Each position in this matrix can be denoted either as (i,α) where $i\in[R]$ and $\alpha\in[C\cdot Z]$ meaning it is the α-th position of the i-th row; or as (j,β) where $j\in[C]$ and $\beta\in[R\cdot Z]$ meaning it is the β-th position of the j-th column.

We will use the following notations to denote the "senders" and "receivers" from the perspective of each NIAR instance:

- **Sources and destinations of the row-wise permutations.** Let $\mathcal{K}^-_{i,S}$ (or $\mathcal{H}^-_{i,S}$, resp.) denote all coordinates (i, α) that correspond to a corrupt (or honest, resp.) element before applying the row-wise permutation.

Note that all destinations in every row-wise permutation are considered corrupt, since the adversary receives all of $\{\mathsf{rk}^{-}_{i,\alpha}\}_{i\in[R],\alpha\in[C\cdot Z]}$ as part of the token. Therefore, we let $\mathcal{K}^{-}_{i,R} = \{(i,\alpha)\}_{\alpha\in[C\cdot Z]}$.

- **Sources and destinations of the column-wise permutations.** Let $\mathcal{K}^{|}_{j,S}$ (or $\mathcal{H}^{|}_{j,S}$, resp.) denote the all coordinates (j, β) that correspond to corrupt (or honest, resp.) elements sources in j-th column-wise permutation.

  Let $\mathcal{K}^{|}_{j,R}$ (or $\mathcal{H}^{|}_{j,R}$, resp.) denote the all coordinates (j, β) such that $\beta \leq m_j$, and moreover (j, β) corresponds to a corrupt (or honest, resp.) destination in the j-th column-wise permutation — recall that after the column-wise permutation, only the first m_j coordinates of column j contain real elements, and the adversary receives only the receiver keys (of the column instances) corresponding to the corrupt real destinations.

3.3.1 Sequence of Hybrids

Experiment $\mathsf{NIDAR\text{-}Expt}^0$. Same as the original $\mathsf{NIDAR\text{-}Expt}^0$ experiment as defined in Section 2.

Experiment Hyb^0_1. Almost the same as $\mathsf{NIDAR\text{-}Expt}^0$ except that we replace each the column instance of NIAR with a NIAR simulator. Recall that the NIAR's simulator only needs to know the destinations of all corrupt sources, as well as what message each corrupt destination receives in each time step. We describe the modifications from $\mathsf{NIDAR\text{-}Expt}^0$ below, where we use $\mathsf{Sim}^{|}_j$ to denote the (stateful) NIAR simulator corresponding to the j-th column instance:

1. During $\mathbf{Setup}(1^\lambda, n, \pi^{(0)}, \mathsf{len})$, instead of calling $\left(\{\mathsf{ek}^{|}_{j,\beta}\}_{\beta\in[R\cdot Z]}, \{\mathsf{rk}^{|}_{j,\beta}\}_{\beta\in[R\cdot Z]}, \mathsf{tk}^{|}_j\right) \leftarrow \mathsf{NIAR}.\mathbf{Setup}(1^\lambda, R\cdot Z,\ \pi^{|}_j, \mathsf{len})$, we now call
$$\left(\{\mathsf{ek}^{|}_{j,\beta}\}_{\beta\in[R\cdot Z]}, \{\mathsf{rk}^{|}_{j,\beta}\}_{\beta\in[R\cdot Z]}, \mathsf{tk}^{|}_j\right) \leftarrow \mathsf{Sim}^{|}_j(1^\lambda, R\cdot Z, \mathsf{len}, \mathcal{K}^{|}_{j,S}, \mathcal{K}^{|}_{j,R}, \mathsf{Leak}(\pi^{|}_j, \mathcal{K}^{|}_{j,S}, \mathcal{K}^{|}_{j,R}))$$

2. During $\mathbf{Enc}(\mathsf{ek}_u, x^{(0)}_{u,t}, t)$ for $u \in \mathcal{H}_S$, instead of calling $\mathsf{ict}_{u,t} \leftarrow \mathsf{NIAR}.\mathbf{Enc}(\mathsf{ek}^{|}_u, x_{u,t}, t)$ and $\widetilde{\mathsf{ict}}_{u,t} \leftarrow \mathsf{NIAR}.\mathbf{Enc}(\mathsf{fek}^{|}_u,\ \mathbf{0},\ t)$ for $u \in \mathcal{H}_S$, we now call
$$\{\mathsf{ict}_{u,t}, \widetilde{\mathsf{ict}}_{u,t}\}_{u\in\mathcal{H}_S} \leftarrow \mathsf{Sim}^{|}_j\left(\left\{\forall (j,\beta) \in \mathcal{K}^{|}_{j,R} \cap \pi^{|}_j(\mathcal{H}^{|}_{j,S}) : (\beta, y_{j,\beta,t})\right\}\right)$$
   where $y_{j,\beta,t}$ is the correct plaintext finally decrypted at the position β in the j-column in time step t, assuming that $\{x^{(0)}_{u,t}\}_{u\in\mathcal{H}_S}$ is the challenge vector being encrypted.

Claim 3.2. *Suppose that the* NIAR *scheme is SIM-secure. The adversary's views in* Hyb^0_1 *and* $\mathsf{NIDAR\text{-}Expt}^0$ *are computationally indistinguishable.*

Proof. Follows in a straightforward fashion due to the SIM-security of the underlying NIAR scheme and the hybrid argument. Specifically, we can swap the column instances of NIAR one by one with an NIAR simulator. □

Experiment Hyb_2^0. Almost the same as Hyb_1^0, except that that we now replace each row instance of NIAR with a NIAR simulator as well, as described below — here we use Sim_i^- to denote the (stateful) NIAR simulator corresponding to the i-th column instance:

1. During $\mathbf{Setup}(1^\lambda, n, \pi^{(0)}, \mathsf{len})$, instead of calling $\left(\{\mathsf{ek}_{i,\alpha}^-\}_{\alpha\in[C\cdot Z]}, \{\mathsf{rk}_{i,\alpha}^-\}_{\alpha\in[C\cdot Z]}, \mathsf{tk}_i^-\right) \leftarrow \mathsf{NIAR}.\mathbf{Setup}(1^\lambda, R\cdot Z, \pi_i^-, \mathsf{len}')$, we now call

$$\left(\{\mathsf{ek}_{i,\alpha}^-\}_{\alpha\in[C\cdot Z]}, \{\mathsf{rk}_{i,\alpha}^-\}_{\alpha\in[C\cdot Z]}, \mathsf{tk}_i^-\right) \leftarrow \mathsf{Sim}_i^-(1^\lambda, C\cdot Z, \mathsf{len}', \mathcal{K}_{i,S}^-, \mathcal{K}_{i,R}^-, \mathsf{Leak}(\pi_i^-, \mathcal{K}_{i,S}^-, \mathcal{K}_{i,R}^-))$$

   where $\mathcal{K}_{i,R}^- := \{(i,\alpha)\}_{\alpha\in[C\cdot Z]}$.

2. During $\mathbf{Enc}(\mathsf{ek}_u, x_{u,t}^{(0)}, t)$ for $u \in \mathcal{H}_S$, instead of calling $\mathsf{ct}_{u,t} \leftarrow \mathsf{NIAR}.\mathbf{Enc}(\mathsf{ek}_u^-, \mathsf{ict}_{u,t}, t)$ and $\widetilde{\mathsf{ct}}_{u,t} \leftarrow \mathsf{NIAR}.\mathbf{Enc}(\mathsf{fek}_u^-, \widetilde{\mathsf{ict}}_{u,t}, t)$, we now call

$$\{\mathsf{ct}_{u,t}, \widetilde{\mathsf{ct}}_{u,t}\}_{u\in\mathcal{H}_S} \leftarrow \mathsf{Sim}_i^-\left(\left\{\forall(i,\alpha)\in\pi_i^-(\mathcal{H}_{i,S}^-) : (\alpha, y_{i,\alpha,t})\right\}\right)$$

   where $y_{i,\alpha,t}$ is the simulated inner ciphertext to be routed to position α of row i during the row-wise permutation in time step t.

Claim 3.3. *Suppose that the* NIAR *scheme is SIM-secure. Then, the adversary's views in* Hyb_1^0 *and* Hyb_2^0 *are computationally indistinguishable.*

Proof. Follows in a straightforward fashion due to the SIM-security of the underlying NIAR scheme and the hybrid argument. Specifically, we can swap the row instances of NIAR one by one with an NIAR simulator. □

Experiment Hyb_3^0. Hyb_3^0 is a rewrite of Hyb_2^0 where we change how we sample the random coins. In Hyb_3^0, we introduce an *initial sampling phase* where a subset of the random coins and events are sampled. Then, based on the outcomes of these partial random coins and events, we invoke a *simulator* that completes the rest of the experiment including interactions with the adversary.

Below we first describe the initial sampling phase.

1. Sample $m_1, m_2, \ldots, m_C$, by throwing n balls into C bins, and counting the bin loads.

2. Sample the complement permutation π' at random, and compute $\pi_{\text{mid}} := (\pi')^{-1} \circ \pi^{(0)}$, which is the permutation to be realized by $\mathsf{MatrixPerm}$. At this moment, the following random coins are fully determined:

   - which bucket each real element uR (either real or filler) should land in during the row-wise permutations, and if any bucket's load exceeds Z, return $\mathsf{overflow}$ just like before. More specifically, suppose that sender u is mapped to $\pi^{(0)}(u)$ as its final destination. Now, let k be the unique integer such that $\sum_{j=1}^{k} m_j < \pi_{\text{mid}}(u) \leq \sum_{j=1}^{k+1} m_j$, then the real element uR should go into the k-th bucket in its corresponding row; and
   - the destination of each real element uR during each of the column-wise permutations.

3. Sample the number of corrupt filler elements for all the buckets in all rows.

At this point, imagine we run the following simulator which continues to interact with the adversary:

Input:

1. set of corrupt senders $\mathcal{K}_S \subseteq [n]$ and set of corrupt receivers $\mathcal{K}_R \subseteq [n]$;
2. for every $u \in [n]$, if $(\pi^{(0)})^{-1}(u) \in \mathcal{H}_S$, what message the corrupt receiver u receives from some honest sender in each time step, based on the $\{x^{(0)}_{u,t}\}_{u,t}$ values.
3. the destination of every corrupt sender $u \in \mathcal{K}_S \subseteq [n]$ based on $\pi^{(0)}$;
4. $m_1, m_2, \ldots, m_C$;
5. π';
6. how many corrupt filler elements land in each bucket during the row-wise permutations;

Simulator algorithm. The simulator now performs the following:

- For each row $i \in [R]$, based on how many corrupt filler elements are to be received in each bucket during the i-th row-wise permutation, randomly assign the corrupt filler elements belonging to this row to the buckets;

  Recall that which bucket each corrupt real element should go during the row-wise permutations was already determined during the initial sampling phase. Therefore, at this time, the simulator knows which bucket each corrupt element (including real and filler) lands in during the row-wise permutations.
- For each bucket, the simulator picks a random unconsumed position for each corrupt element that is supposed to go into this bucket during the row-wise permutation. At this moment, it is fully determined where all corrupt elements go during the row-wise permutation.
- For each $j \in [C]$, for all the corrupt filler elements in column j after the row-wise permutation, pick a random (non-overlapping) position among the last $R \cdot Z - m_j$ positions to be its destination during the j-th column-wise permutation. At this moment, the routes of all corrupt elements during the row-wise and column-wise permutations are fully determined.
- At this moment, it is not hard to see that the simulator can accomplish the interactions with the adversary, since it knows all the inputs needed for calling the NIAR's simulators $\{\mathsf{Sim}^{|}_j\}_{j\in[C]}$ and $\{\mathsf{Sim}^{-}_i\}_{i\in[R]}$.

Claim 3.4. Hyb^0_3 *and* Hyb^0_2 *are identically distributed.*

Proof. It is not difficult to check that Hyb^0_3 is simply a rewrite of Hyb^0_2, where the random coins are sampled in a different manner, by sampling a subset of the random coins and events first in an initial sampling stage, and then having a simulator accomplish the remaining. ☐

Experiment Hyb^1_3. Hyb^1_3 is almost the same as Hyb^0_3, except the following modifications. Recall that the adversary submits $\pi^{(0)}$ and $\pi^{(1)}$ that are almost identical except for swapping the destinations of two honest senders. Moreover, recall that in Hyb^0_3, we first have an initial sampling stage where part of the random coins are sampled; then we invoke a simulator with some input, and the rest of the simulation is completed by this simulator.

1. During the initial sampling stage: let $\pi_{\mathrm{mid}} := (\pi')^{-1} \circ \pi^{(1)}$.
2. Part of the inputs to the simulator is changed to the following:

- for every $u \in [n]$, if $(\pi^{(1)})^{-1}(u) \in \mathcal{H}_S$, what message the corrupt receiver u receives from some honest sender in each time step, based on the $\{x^{(1)}_{u,t}\}_{u,t}$ values;
- the destination of every corrupt sender $u \in \mathcal{K}_S \subseteq [n]$ based on $\pi^{(1)}$.

Lemma 3.5. *Suppose that $(1-\rho)Z \geq \Theta(\log \frac{1}{\delta})$ where $\Theta(\cdot)$ hides some appropriately large constant, where ρ is the fraction of corrupt senders. For any S,*

$$\Pr[\mathsf{view}_{\mathcal{A}}(\mathsf{Hyb}^0_3) \in S] \leq e^{\epsilon} \cdot \Pr[\mathsf{view}_{\mathcal{A}}(\mathsf{Hyb}^1_3) \in S] + \delta'$$

where $\mathsf{view}_{\mathcal{A}}(\mathsf{Hyb}^b_3)$ denotes the adversary's view in experiment Hyb^b_3 for $b \in \{0,1\}$, and

$$\epsilon = O(1) \cdot \frac{\sqrt{Z \cdot \min((1-\rho), \rho) \log \frac{1}{\delta}} + \log \frac{1}{\delta}}{(1-\rho)Z}, \quad \delta' = O(n\delta)$$

Proof. Observe that due to the admissibility rule on the adversary, for the above inputs to the simulator, it does not matter whether we use $\{x^{(0)}_{u,t}\}_{u,t}, \pi^{(0)}$ or $\{x^{(1)}_{u,t}\}_{u,t}, \pi^{(1)}$ — the outcomes are the same. In this sense, the only real difference in Hyb^0_3 and Hyb^1_3 is that π_{mid} is now computed as $(\pi')^{-1} \circ \pi^{(1)}$.

In both Hyb^0_3 and Hyb^1_3, the adversary's view depends only on the simulator's input (and the internal random coins tossed by the simulator). Let the simulator's input be Inp^b in Hyb^b_3 for $b \in \{0,1\}$. By the post-processing lemma of differential privacy [Vad17, DR14], it suffices to prove that for any S,

$$\Pr[\mathsf{Inp}^0 \in S] \leq e^{\epsilon} \cdot \Pr[\mathsf{Inp}^1 \in S] + \delta' \tag{1}$$

Recall that the input to the simulator consists of the following:

1. set of corrupt senders $\mathcal{K}_S \subseteq [n]$ and set of corrupt receivers $\mathcal{K}_R \subseteq [n]$;
2. for every $u \in [n]$, if $(\pi^{(b)})^{-1}(u) \in \mathcal{H}_S$, what message the corrupt receiver u receives from some honest sender in each time step, based on the $\{x^{(b)}_{u,t}\}_{u,t}$ values.
3. the destination of every corrupt sender $u \in \mathcal{K}_S \subseteq [n]$ based on $\pi^{(b)}$;
4. $m_1, m_2, \ldots, m_C$;
5. π';
6. how many corrupt filler elements land in each bucket during the row-wise permutations;

For 1-5, they are the same no matter whether we are in Hyb^0_3 or Hyb^1_3. Therefore, it suffices to prove that the numbers of corrupt filler elements that land in all buckets satisfy the above Equation (1) in the two experiments. This is the most technical step in our proof, we therefore present the full proof of this statement in Lemma 3.10 of Section 3.3.2. □

Experiment Hyb^1_2. Same as Hyb^0_2 except that $\pi^{(1)}$ and $\{x^{(1)}_{u,t}\}_{u,t}$ are used in place of $\pi^{(0)}$ and $\{x^{(0)}_{u,t}\}_{u,t}$.

Claim 3.6. *Hyb^1_2 and Hyb^1_3 are identically distributed.*

Proof. The proof is the same as Claim 3.4, i.e., Hyb^1_3 is a rewrite of Hyb^1_2 where the sampling is performed in a different way by sampling a subset of the random coins and events first, and then invoking a simulator which samples the remaining randomness and completes the interactions with the adversary. □

Experiment Hyb_1^1**.** Same as Hyb_1^0 except that $\pi^{(1)}$ and $\{x_{u,t}^{(1)}\}_{u,t}$ are used in place of $\pi^{(0)}$ and $\{x_{u,t}^{(0)}\}_{u,t}$.

Claim 3.7. *Suppose that the* NIAR *scheme is SIM-secure. Then, the adversary's views in* Hyb_1^1 *and* Hyb_2^1 *are computationally indistinguishable.*

Proof. The proof follows in the same way as that of Claim 3.3. □

Experiment $\mathsf{NIDAR\text{-}Expt}^1$**.** Same as the original $\mathsf{NIDAR\text{-}Expt}^1$ experiment as defined in Section 2.

Claim 3.8. *Suppose that the* NIAR *scheme is SIM-secure. Then, the adversary's views in* Hyb_1^1 *and* $\mathsf{NIDAR\text{-}Expt}^1$ *are computationally indistinguishable.*

Proof. The proof follows in the same way as that of Claim 3.2. □

Theorem 3.9 (2-layer NIDAR)**.** *Let $\mathcal{A}$ be an arbitrary non-uniform p.p.t. adversary that controls ρ fraction of the senders, and recall that for $b \in \{0, 1\}$, the experiment* $\mathsf{NIDAR\text{-}Expt}^b$ *outputs the adversary $\mathcal{A}$'s output. Suppose that $(1-\rho)Z \geq \Theta(\log \frac{1}{\delta})$ where $\Theta(\cdot)$ hides a suitably large constant; further, suppose that the underlying* NIAR *scheme is SIM-secure. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$*, for any S,*

$$\Pr[\mathsf{NIDAR\text{-}Expt}^0 \in S] \leq e^{\epsilon} \cdot \Pr[\mathsf{NIDAR\text{-}Expt}^1 \in S] + \delta'$$

where

$$\epsilon = O(1) \cdot \frac{\sqrt{Z \cdot \min((1-\rho), \rho) \log \frac{1}{\delta}} + \log \frac{1}{\delta}}{(1-\rho)Z}, \qquad \delta' = O(n\delta) + \mathsf{negl}(\lambda)$$

Proof. Due to the hybrid argument, the theorem follows from Claims 3.2, 3.3, 3.4, Lemma 3.5, as well as Claims 3.6, 3.7, and 3.8.

□

3.3.2 Differential Privacy Lemma

Consider the following experiment DPExpt^b where $b \in \{0, 1\}$, in which the adversary's view capatures the simulator's input in the earlier hybrid experiment Hyb_3^b.

DPExpt^b

1. The adversary submits two neighboring permutations $\pi^{(0)}$ and $\pi^{(1)}$, i.e., the two permutations are otherwise identical except for swapping the destinations of two honest senders.
2. The challenger randomly chooses a set of corrupt senders $\mathcal{K}_S$ and tells the adversary the set $\mathcal{K}_S$.
3. The challenger samples $m_1, \ldots, m_C$ at random as mentioned before and tells the adversary these values.
4. The challenger samples a random π'. The challenger computes $\pi_{\text{mid}} = (\pi')^{-1} \circ \pi^{(b)}$. Tell the adversary π'.
5. Each row has C buckets, and each real element uR belongs to some row as mentioned earlier where $u \in [n]$. For each row $i \in [R]$, the challenger throws all real elements belonging to row i into C buckets. Suppose that $\pi_{\text{mid}}(u) = v$ where $\sum_{j=1}^{k} m_j < v \leq \sum_{j=1}^{k+1} m_j$, then the

element uR should go into the k-th bucket in its respective row. If any bucket exceeds Z real elements, abort throwing overflow.

6. For each row i, let $\mu_{i,1}, \ldots, \mu_{i,C}$ denote the remaining empty slots inside the buckets belonging to row i. For each empty slot, fill it with a random filler element belonging to this row. Tell the adversary for each $i \in [R]$ and $j \in [C]$, exactly how many corrupt filler elements go into the j-th bucket of row i.
7. Return the adversary's view.

We want to prove the following lemma, which is the core technical lemma needed the proof of Lemma 3.5.

Lemma 3.10. *Assume that $(1-\rho)Z \geq \Theta(\log \frac{1}{\delta})$ where ρ denotes the fraction of corrupt senders and $\Theta(\cdot)$ hides a sufficiently large constant. For any S,*

$$\Pr[\mathsf{DPExpt}^0 \in S] \leq e^{\epsilon} \cdot \Pr[\mathsf{DPExpt}^1 \in S] + \delta'$$

where

$$\epsilon = O(1) \cdot \frac{\sqrt{Z \cdot \min((1-\rho), \rho) \log \frac{1}{\delta}} + \log \frac{1}{\delta}}{(1-\rho)Z}, \quad \delta' = O(n\delta)$$

Proof. The experiment DPExpt^b needs to flip various coins. We will use the following names to refer to these coins:

- *Corruption coins:* the coins used to determine the corrupt set of senders $\mathcal{K}_S$;
- *Routing coins:* includes the random choice of $m_1, \ldots, m_C$, and π' — these coins determine which bucket each real element will be thrown into;
- *Filler coins:* the coins used to decide which filler elements are used to fill the remaining empty slots in each bucket, after the real elements are thrown into buckets.

Let $\widetilde{M}^b_{i,j}$ denote the total number of filler elements that the j-th bucket of the i-th row wants to receive, assuming we are in DPExpt^b where $b \in \{0, 1\}$. Let u^* and v^* be the two honest senders whose destinations got swapped in $\pi^{(0)}$ and $\pi^{(1)}$. If u^* and v^* belong to the same row initially, then, fixing the same routing coins in DPExpt^0 or DPExpt^1 respectively, it must be that $\widetilde{M}^0_{i,j} = \widetilde{M}^1_{i,j}$ for all i, j. In this case, the adversary's views are identical in DPExpt^0 and DPExpt^1.

Below we focus on the case when u^* and v^* do not belong to the same row initially — specifically, suppose u^* belongs to row i_0 and v^* belongs to row i_1, respectively. Once the routing coins are fixed in both DPExpt^0 and DPExpt^1, the following must hold:

$\widetilde{M}^0_{i,j} = \widetilde{M}^1_{i,j}$, for almost all $i \in [R]$ and $j \in [C]$, except for some i_0, j_0, and i_1, j_1, where

$$\widetilde{M}^0_{i_1,j_0} = \widetilde{M}^1_{i_1,j_0} - 1, \qquad \widetilde{M}^0_{i_1,j_1} = \widetilde{M}^1_{i_1,j_1} + 1$$
$$\widetilde{M}^0_{i_0,j_0} = \widetilde{M}^1_{i_0,j_0} + 1, \qquad \widetilde{M}^0_{i_0,j_1} = \widetilde{M}^1_{i_0,j_1} - 1$$

Focus on a single row r_0. Henceforth we shall first focus on the row i_0, since analyzing the other row i_1 is similar. Suppose we have fixed the routing coins. We use the vector $\{M_1, M_2, \ldots, M_{j_b} + 1, \ldots, M_C\}_{j\in[C]}$ to denote the total number of filler elements in each bucket of the i_0-th row, when we are in DPExpt^b.

$$\begin{aligned}
&\# \text{ filler elements each bucket in row } i_0 : \\
&\mathsf{DPExpt}^0 : \Big(M_1, M_2, \ldots, M_{j_0-1}, \underline{M_{j_0}+1}, M_{j_0+1}, \ldots, \qquad\qquad\qquad\qquad \ldots, M_C\Big) \\
&\mathsf{DPExpt}^1 : \Big(M_1, M_2, \ldots, \qquad\qquad\qquad\qquad \ldots, M_{j_1-1}, \underline{M_{j_1}+1}, M_{j_1+1}, \ldots, M_C\Big)
\end{aligned}$$

For $j \in [C]$, we use $\mu_j \le M_j$ to denote the number of corrupt filler elements in the j-th bucket of the i_0-th row. In the random process of DPExpt^b, we can imagine that first, the total number of filler elements of each bucket $\{M_1, M_2, \ldots, M_{j_b} + 1, \ldots, M_C\}_{j\in[C]}$ is determined, and then, the random variables $\{\mu_j\}_{j\in[C]}$ can be determined in the following way. Suppose that the i_0-th row has ρ' fraction of corrupt senders. Suppose we have a database where exactly $M_1, M_2, \ldots, M_{j_b}+1, \ldots, M_C$ elements have the attributes $1, 2, \ldots, C$, respectively. We now sample $\rho' \cdot \frac{n}{R}$ elements at random without replacement from this database, and μ_j is the number of elements with attribute j. Note that $\frac{n}{R}$ denotes the total number of filler elements (including honest and corrupt) belonging to any specific row, and this is fixed regardless of whether we are in DPExpt^0 or DPExpt^1.

Claim 3.11. *Suppose that $(1-\rho)Z \ge \Theta'(\log\frac{1}{\delta})$ where $\Theta'(\cdot)$ hides a sufficiently large constant. No matter whether we are in DPExpt^0 or DPExpt^1, the following statements hold. For any fixed bucket, with probability at least $1-\delta$ over the choice of the routing coins, its load of real elements is between $[Z - O(\sqrt{Z\cdot\log\frac{1}{\delta}}), Z + O(\sqrt{Z\cdot\log\frac{1}{\delta}})]$. Further, for any fixed bucket, with probability at least $1-\delta$ over the choice of routing coins, its load of filler elements is between $[Z - O(\sqrt{Z\cdot\log\frac{1}{\delta}}), Z + O(\sqrt{Z\cdot\log\frac{1}{\delta}})]$.*

As a direct corollary, for any fixed $j \in [C]$, with probability at least $1-\delta$, $M_j = \Theta(Z)$ where $\Theta(\cdot)$ hides an appropriately large constant.

Proof. With the random process of DPExpt, essentially, every real element is assigned to a random bucket within its row. The expected number of real elements each bucket receives is exactly Z. Consider one fixed bucket. By the Chernoff bound, there are some appropriate constants c and c' such that

$$\begin{aligned}
&\Pr\left[\text{a fixed bucket's real load} \in \left[Z - c\cdot\sqrt{Z\cdot\log\frac{1}{\delta}}, Z + c\cdot\sqrt{Z\cdot\log\frac{1}{\delta}}\right]\right] \\
\ge\,& 1 - \exp\left(-c'\cdot\log\frac{1}{\delta}\right) = 1-\delta
\end{aligned}$$

□

Claim 3.12. *Suppose that $(1-\rho)Z \ge \Theta'(\log\frac{1}{\delta})$ where $\Theta'(\cdot)$ hides an appropriately large constant. Then, with probability $1-\delta$ over the choice of the corruption coins, it must be that $1-\rho' = \Theta(1-\rho)$ and $\rho' Z = O(\rho Z + \log\frac{1}{\delta})$.*

Proof. Follows in a straightforward fashion from the Chernoff bound. □

Lemma 3.13. *Let ρ' be the fraction of corrupt senders in row i_0. For any fixed $j \in [C]$, the following holds regardless of the choice of b, with probability at least $1-\delta$ over the choice of filler coins (of row i_0):*

$$\mu_j \in \left[\rho' M_j - O\left(\sqrt{\min(\rho', 1-\rho')M_j \log \frac{1}{\delta}} + \log \frac{1}{\delta}\right), \rho' M_j + O\left(\sqrt{\min(\rho', 1-\rho')M_j \log \frac{1}{\delta}} + \log \frac{1}{\delta}\right)\right]$$

As a corollary, suppose that $(1-\rho)Z \geq \Theta(\log \frac{1}{\delta})$ where $\Theta(\cdot)$ hides a sufficiently large constant. For any fixed j, it must be that conditioned on any specific choice of good routing coins and corruption coins that satisfy the good events of Claims 3.11 and 3.12, with probability at least $1-\delta$ over the choice of the filler coins (of row i_0),

- $M_j - \mu_j \geq \Theta((1-\rho)Z)$;
- $\mu_j < M_j - 1$.

Proof. By negative association and the Chernoff bound, with at least $1-\delta$ probability, the total number of honest filler elements in bucket j of row i_0 is within the range $[(1-\rho')M_j - O\left(\sqrt{(1-\rho')M_j \log \frac{1}{\delta}} + \log \frac{1}{\delta}\right), (1-\rho')M_j + O\left(\sqrt{(1-\rho')M_j \log \frac{1}{\delta}} + \log \frac{1}{\delta}\right)]$. Observe that the total number of honest and corrupt filler elements of the j-th bucket of row i_0 is exactly M_j. It follows that with at least $1-\delta$ probability, $\mu_j \in [\rho' M_j - O\left(\sqrt{(1-\rho')M_j \log \frac{1}{\delta}} + \log \frac{1}{\delta}\right), \rho' M_j + O\left(\sqrt{(1-\rho')M_j \log \frac{1}{\delta}} + \log \frac{1}{\delta}\right)]$.

Similarly, by negative association and Chernoff bound, we get that with at least $1-\delta$ probability, $\mu_j \in [\rho' M_j - O\left(\sqrt{\rho' M_j \log \frac{1}{\delta}} + \log \frac{1}{\delta}\right), \rho' M_j + O\left(\sqrt{\rho' M_j \log \frac{1}{\delta}} + \log \frac{1}{\delta}\right)]$. The lemma follows by combining the above.

□

For convenience, we shall use the notation rc to denote the union of the router coins and the corruption coins. We use rc to denote the random variable and use rc to denote any specific choice of these coins. For any good choice rc that satisfies the good events of Claims 3.11 and 3.12, — note that these coins fix the $M_1, \ldots M_C$ values. Let $\mu_1, \ldots, \mu_C$ be a set of good values that satisfy the good events of Lemma 3.13, w.r.t. these $M_1, \ldots M_C$ values. We have the following where $\Pr_b$ is taken over the choice of filler coins of row i_0 in DPExpt^b.

$$
\begin{aligned}
\frac{\Pr_0[\mu_1,\ldots,\mu_C|\mathsf{rc}=rc]}{\Pr_1[\mu_1,\ldots,\mu_C|\mathsf{rc}=rc]} \leq & \frac{1-\frac{\mu_{j_1}}{M_{j_1}+1}}{1-\frac{\mu_{j_0}}{M_{j_0}+1}} = 1+\frac{\frac{\mu_{j_0}}{M_{j_0}+1}-\frac{\mu_{j_1}}{M_{j_1}+1}}{1-\frac{\mu_{j_0}}{M_{j_0}+1}} \\
\leq & 1+\frac{O(1)\cdot\frac{\sqrt{Z\min(1-\rho',\rho')\log\frac{1}{\delta}}+\log\frac{1}{\delta}}{Z}}{1-\rho} \\
\leq & 1+O(1)\cdot\frac{\sqrt{Z\min(1-\rho',\rho')\log\frac{1}{\delta}}+\log\frac{1}{\delta}}{(1-\rho)Z} \\
\leq & 1+O'(1)\cdot\frac{\sqrt{\min((1-\rho)Z,\rho Z+\log\frac{1}{\delta})\log\frac{1}{\delta}}+\log\frac{1}{\delta}}{(1-\rho)Z} \\
\leq & 1+O''(1)\cdot\frac{\sqrt{Z\cdot\min((1-\rho),\rho)\log\frac{1}{\delta}}+\log\frac{1}{\delta}}{(1-\rho)Z} \\
\leq & \exp\left(O''\left(\frac{\sqrt{Z\cdot\min((1-\rho),\rho)\log\frac{1}{\delta}}+\log\frac{1}{\delta}}{(1-\rho)Z}\right)\right) \qquad (\spadesuit)
\end{aligned}
$$

In the above, the first step of the derivation is proven in Lemma 3.14 and the proof is deferred to later.

Henceforth let $\epsilon := O''\left(\frac{\sqrt{Z\cdot\min((1-\rho),\rho)\log\frac{1}{\delta}}+\log\frac{1}{\delta}}{(1-\rho)Z}\right)$.

Back to considering both rows i_0 and i_1. So far, we have focused only on the row i_0. Below, we shall complete the proof of Lemma 3.10, and we shall now consider both rows i_0 and i_1. Let $\boldsymbol{\mu}[i_0]$ and $\boldsymbol{\mu}[i_1]$ denote the corrupt filler load vector for buckets in rows i_0 and i_1, respectively.

Now, consider an arbitrary set $S := \{(\boldsymbol{\mu}[i_0],\boldsymbol{\mu}[i_1])\}$ containing choices of $(\boldsymbol{\mu}[i_0],\boldsymbol{\mu}[i_1])$ values. We also use S to denote the event that the adversary sees corrupt filler load vectors of rows i_0 and i_1 that lie within S.

$$
\begin{aligned}
\Pr_0[S] &= \sum_{\boldsymbol{\mu}[i_0],\boldsymbol{\mu}[i_1]\in S}\Pr_0[\boldsymbol{\mu}[i_0],\boldsymbol{\mu}[i_1]] = \sum_{\boldsymbol{\mu}[i_0],\boldsymbol{\mu}[i_1]\in S}\sum_{rc}\Pr_0[\boldsymbol{\mu}[i_0],\boldsymbol{\mu}[i_1],rc] \\
&= \sum_{\boldsymbol{\mu}[i_0],\boldsymbol{\mu}[i_1]\in S}\sum_{rc:\text{good}}\Pr_0[\boldsymbol{\mu}[i_0],\boldsymbol{\mu}[i_1],rc] + \sum_{\boldsymbol{\mu}[i_0],\boldsymbol{\mu}[i_1]\in S}\sum_{rc:\neg\text{good}}\Pr_0[\boldsymbol{\mu}[i_0],\boldsymbol{\mu}[i_1],rc] \\
&= \sum_{\boldsymbol{\mu}[i_0],\boldsymbol{\mu}[i_1]\in S}\sum_{rc:\text{good}}\Pr_0[\boldsymbol{\mu}[i_0]\,|rc]\cdot\Pr_0[\boldsymbol{\mu}[i_1]\,|rc]\cdot\Pr_0[rc]+O(C)\cdot\log\frac{1}{\delta} \\
&\leq \sum_{\boldsymbol{\mu}[i_0],\boldsymbol{\mu}[i_1]\in S}\sum_{rc:\text{good}}e^{\epsilon}\cdot\Pr_1[\boldsymbol{\mu}[i_0]|rc]\cdot e^{\epsilon}\cdot\Pr_1[\boldsymbol{\mu}[i_1]|rc]\cdot\Pr_1[rc]+O(C)\cdot\log\frac{1}{\delta} \\
&\leq \sum_{\boldsymbol{\mu}[i_0],\boldsymbol{\mu}[i_1]\in S}\sum_{rc}e^{2\epsilon}\cdot\Pr_1[\boldsymbol{\mu}[i_0]|rc]\cdot\Pr_1[\boldsymbol{\mu}[i_1]|rc]\cdot\Pr_1[rc]+O(C)\cdot\log\frac{1}{\delta} \\
&= e^{2\epsilon}\cdot\Pr_1[S]+O(C)\cdot\log\frac{1}{\delta}
\end{aligned}
$$

In the above, the subscript "rc: good" means that the $M_1, M_2, \ldots, M_C$ values resulting from the choice of rc satisfy the good events of Claims 3.11, 3.12, and Lemma 3.13, w.r.t. the choice of $\boldsymbol{\mu}[i_0]$ and $\boldsymbol{\mu}[i_1]$ which are already fixed in the outer summation. The notation $\Pr_b[\mathsf{evt}]$ denotes the probability of seeing the event evt in DPExpt^b, and if we write $\Pr[\mathsf{evt}]$ without a subscript, it means that the probability of the relevant event evt is the same in both DPExpt^0 and DPExpt^1. The $O(C)$ factor before the $\log \frac{1}{\delta}$ is due to taking a union bound over all choice of $j \in [C]$.

□

Analysis of the sampling mechanism. Imagine that we have a database containing M items, where each item is assigned some attribute from the domain $[C]$, i.e., there are C total attributes. Consider the following sampling mechanism Samp.

The sampling mechanism Samp

Sample s items at random (without replacement) from this database, and output a vector $(\mu_1, \mu_2, \ldots, \mu_C)$ reporting the total number of occurrences for each of the C attributes.

We now prove a useful lemma that will be needed in our differential anonymity proof. Consider two neighboring databases DB and DB'. The only difference in DB and DB' is that the i-th item's attribute is changed from k to k'. Suppose that in database DB, the total number of occurrences for each of the C attributes is denoted $(M_1, M_2, \ldots, M_k+1, \ldots, M_C)$, and in DB', the total number occurrences for each of the C attributes is denoted $(M_1, M_2, \ldots, M_{k'}+1, \ldots, M_C)$. We use the notations $\Pr_{\mathsf{DB}}[\mu_1, \ldots, \mu_C]$ and $\Pr_{\mathsf{DB}'}[\mu_1, \ldots, \mu_C]$ to denote the probabilities of encountering the sample $(\mu_1, \ldots, \mu_C)$, when the database is DB and DB', respectively.

Lemma 3.14. *Given any $(\mu_1, \ldots, \mu_C)$ vector where $\mu_j \le M_j$ for $j \in [C]$,*

$$\frac{\Pr_{\mathsf{DB}}[\mu_1, \ldots, \mu_C]}{\Pr_{\mathsf{DB}'}[\mu_1, \ldots, \mu_C]} = \frac{1 - \frac{\mu_{k'}}{M_{k'}+1}}{1 - \frac{\mu_k}{M_k+1}}$$

Proof. We have

$$\Pr_{\mathsf{DB}}[\mu_1, \ldots, \mu_C] = \frac{\binom{M_k+1}{\mu_k} \cdot \Pi_{j\in[C], j\neq k}\binom{M_j}{\mu_j}}{\binom{M}{s}}$$

$$\Pr_{\mathsf{DB}'}[\mu_1, \ldots, \mu_C] = \frac{\binom{M_{k'}+1}{\mu_{k'}} \cdot \Pi_{j\in[C], j\neq k'}\binom{M_j}{\mu_j}}{\binom{M}{s}}$$

where $M := \sum_{j\in[C]} M_j$ and $s = \sum_{j\in[C]} \mu_j$. Therefore,

$$\frac{\Pr_{\mathsf{DB}}[\mu_1, \ldots, \mu_C]}{\Pr_{\mathsf{DB}'}[\mu_1, \ldots, \mu_C]} = \frac{\binom{M_k+1}{\mu_k} \cdot \binom{M_{k'}}{\mu_{k'}}}{\binom{M_k}{\mu_k} \cdot \binom{M_{k'}+1}{\mu_{k'}}} = \frac{M_k+1}{M_k+1-\mu_k} \cdot \frac{M_{k'}+1-\mu_{k'}}{M_{k'}+1} = \frac{1 - \frac{\mu_{k'}}{M_{k'}+1}}{1 - \frac{\mu_k}{M_k+1}}$$

□

4 Multi-Layer NIDAR

4.1 Multi-Layer NIDAR Construction

Inspired by the two-layer construction, we now suggest a multi-layer variant. To formally describe the scheme, we will use a recursive construction.

L**-layer** NIDAR where $L \geq 2$

Assume: (same as before) after the adversary chooses which users to corrupt and before the **Setup** algorithm is first invoked, all senders are randomly permuted, and we renumber the senders from 1 to n after this initial permutation. Throughout the following algorithms, we refer to senders by these randomly renumbered identities.

Parameters: let $L \geq 2$ be the total number of layers. Let Z be an even number that denotes the bucket size. For simplicity, we will first assume that $R := (2n/Z)^{1/L}$ is an integer, and R is also the fanout in the butterfly network when the recursions are expanded all the way (see Figure 1). We will deal with the indivisible case later in this section.

Main algorithms: we describe the main algorithms below, where each algorithm may in turn call a recursive subroutine, denoted $\mathsf{RecSetup}$, RecEnc, and RecRte, respectively. We will define these recursive subroutines subsequently in Section 4.2.

- **Setup**$(1^\lambda, n, \pi, \mathsf{len})$:
  - let $(\pi_{\text{mid}}, \{\mathsf{ek}_v\}_{v\in[2n]}, \{\mathsf{rk}^{|}_{j,\beta}\}_{j\in[C],\beta\in[m_j]}, \mathsf{tk}') \leftarrow \mathsf{RecSetup}(1^\lambda, 2n, L, 1)$, where
  $$\mathsf{tk}' := \Big(\{m_j\}_{j\in[C]}, \{\mathsf{tk}^{-}_i\}_{i\in[R]}, \{\mathsf{tk}^{|}_j\}_{j\in[C]}, \{\mathsf{rk}^{-}_{i,\alpha}\}_{i\in[R],\alpha\in[C\cdot Z]}\Big);$$
  note that since each sender encrypts a real element and a filler element, we may pretend that each sender acts as two virtual senders, and this is why we pass $2n$ to the recursive call $\mathsf{RecSetup}$;
  - for each $u \in [n]$, let $\mathsf{ek}_u := \{\mathsf{ek}_{2(u-1)+1}, \mathsf{ek}_{2u}\}$;
  - compute the complement permutation π' such that $\pi' \circ \pi_{\text{mid}} = \pi$;
  - let $\mathsf{rk}_1, \ldots, \mathsf{rk}_n := \pi'\Big(\{\mathsf{rk}^{|}_{j,\beta}\}_{j\in[C],\beta\in[m_j]}\Big)$ where $\{\mathsf{rk}^{|}_{j,\beta}\}_{j\in[C],\beta\in[m_j]}$ is flattened as a 1-dimensional array in the lexicographical ordering of (j, β);
  - let the router's token $\mathsf{tk} := \Big(\pi', \{m_j\}_{j\in[C]}, \{\mathsf{tk}^{-}_i\}_{i\in[R]}, \{\mathsf{tk}^{|}_j\}_{j\in[C]}, \{\mathsf{rk}^{-}_{i,\alpha}\}_{i\in[R],\alpha\in[C\cdot Z]}\Big)$;
  - output the sender and receiver keys $\{\mathsf{ek}_u, \mathsf{rk}_u\}_{u\in[n]}$, as well as the router token tk.
- **Enc**$(\mathsf{ek}_u, x_{u,t}, t)$:
  - parse $\mathsf{ek}_u = (\mathsf{ek}, \mathsf{ek}')$;
  - call $\mathsf{ct} \leftarrow \mathsf{RecEnc}(\mathsf{ek}, x, t, L)$ and $\mathsf{ct}' \leftarrow \mathsf{RecEnc}(\mathsf{ek}', \mathbf{0}, t, L)$; and
  - output $\mathsf{CT} := (\mathsf{ct}, \mathsf{ct}')$.
- **Rte**$(\mathsf{tk}, \mathsf{CT}_{1,t}, \ldots, \mathsf{CT}_{n,t})$:
  - let tk' be the same as tk but without the π' term;
  - for each $u \in [n]$, parse $\mathsf{CT}_{u,t} := (\mathsf{ct}_{2(u-1)+1}, \mathsf{ct}_{2u})$;
  - call $\mathsf{CT}'_1, \ldots, \mathsf{CT}'_n \leftarrow \mathsf{RecRte}(\mathsf{tk}', \mathsf{ct}_1, \ldots, \mathsf{ct}_{2n}, L, 1)$ and return $\pi'(\mathsf{CT}'_1, \ldots, \mathsf{CT}'_n)$.
- **Dec**$(\mathsf{rk}_u, \mathsf{CT}'_u)$: output $\mathsf{NIAR}.\mathbf{Dec}(\mathsf{rk}_u, \mathsf{CT}'_u)$.

4.2 Recursive Subroutines

Subroutine $\mathsf{RecSetup}(1^\lambda, n, \mathsf{len}, \mathsf{nLayer}, \mathsf{bFin})$**:** If $\mathsf{nLayer} = 1$, then

1. view the symobilic input 1R, 1F, 2R, 2F, ..., $\frac{n}{2}$R, $\frac{n}{2}$F as R buckets each of size Z such that each bucket has half real and half filler elements, and simulate a run of the $\mathsf{RowPerm}$ algorithm resulting in the permutation π — recall that the $\mathsf{RowPerm}$ algorithm may throw an $\mathsf{overflow}$ exception if any bucket receives more real elements than its capacity.
2. let $\left(\{\mathsf{ek}_v, \mathsf{rk}_v\}_{v\in[n]}, \mathsf{tk}\right) \leftarrow \mathsf{NIAR}.\mathbf{Setup}(1^\lambda, n, \pi, \mathsf{len})$, and return $\left(\{\mathsf{ek}_v, \mathsf{rk}_v\}_{v\in[n]}, \mathsf{tk}\right)$.

Else, continue with the following:

1. View the symbolic vector 1R, 1F, 2R, 2F, ..., $\frac{n}{2}$R, $\frac{n}{2}$F as a matrix containing $R \times C$ buckets each of size Z, where R is a global parameter defined earlier, and $C = n/(R \cdot Z)$.
   - If $\mathsf{bFin} = 1$, then for each column, simulate a run of the $\mathsf{ColPerm}$ algorithm which randomly permutes the column and moves real elements to the front; and let $m_1, m_2, \ldots, m_C$ be the number of real elements in each column after applying the row-wise permutations;
   - Else if $\mathsf{bFin} = 0$, then for each column, simulate a run of the $\mathsf{RowPerm}$ algorithm which assigns each real element to a random bucket, and uses the remaining filler elements at random to pad all buckets to its maximum capacity; furthermore, a random permutation is applied to within each bucket.

   Let $\pi^{|}_1, \ldots, \pi^{|}_C$ denote the resulting column-wise permutations.
2. For each $j \in [C]$, call
$$\left(\{\mathsf{ek}^{|}_{j,\beta}\}_{\beta\in[R\cdot Z]}, \{\mathsf{rk}^{|}_{j,\beta}\}_{\beta\in[R\cdot Z]}, \mathsf{tk}^{|}_j\right) \leftarrow \mathsf{NIAR}.\mathbf{Setup}(1^\lambda, R\cdot Z, \pi^{|}_j, \mathsf{len})$$
3. For each $i \in [R]$, recursively call
$$\left(\pi^{-}_i, \{\mathsf{ek}^{-}_{i,\alpha}\}_{\alpha\in[C\cdot Z]}, \{\mathsf{rk}^{-}_{i,\alpha}\}_{\alpha\in[C\cdot Z]}, \mathsf{tk}^{-}_i\right) \leftarrow \mathsf{RecSetup}(1^\lambda, C\cdot Z, \kappa\cdot\mathsf{len}, \mathsf{nLayer}-1, 0)$$
   where $1/\kappa$ denotes the rate of $\mathsf{NIAR}.\mathbf{Enc}$ (i.e., κ is the ciphertext size divided by the plaintext size).
4. Let π_{mid} be the effective permutation after applying the row-wise permutation π^{-}_i to each row $i \in [R]$, and applying the column-wise permutation $\pi^{|}_j$ to each column $j \in [C]$. In particular, if $\mathsf{bFin} = 1$ then π_{mid} denotes the permutation on only the real elements; else, π_{mid} denotes the permutation on all elements (including real and filler).
5. For $v \in [n]$, suppose that the element v corresponds to the initial position (i, α), and is routed to position (j, β) after the row-wise permutations[1], then, let $\mathsf{ek}_v := (\mathsf{ek}^{-}_{i,\alpha}, \mathsf{ek}^{|}_{j,\beta})$.
6. If $\mathsf{bFin} = 1$, then let $\mathsf{tk} := \left(\{m_j\}_{j\in[C]}, \{\mathsf{tk}^{-}_i\}_{i\in[R]}, \{\mathsf{tk}^{|}_j\}_{j\in[C]}, \{\mathsf{rk}^{-}_{i,\alpha}\}_{i\in[R],\alpha\in[C\cdot Z]}\right)$ and return $\left(\pi_{\text{mid}}, \{\mathsf{ek}_v\}_{v\in[n]}, \{\mathsf{rk}^{|}_{j,\beta}\}_{j\in[C],\beta\in[m_j]}, \mathsf{tk}\right)$.

   Else, let $\mathsf{tk} := \left(\{\mathsf{tk}^{-}_i\}_{i\in[R]}, \{\mathsf{tk}^{|}_j\}_{j\in[C]}, \{\mathsf{rk}^{-}_{i,\alpha}\}_{i\in[R],\alpha\in[C\cdot Z]}\right)$ and return $\left(\pi_{\text{mid}}, \{\mathsf{ek}_v\}_{v\in[n]}, \{\mathsf{rk}^{|}_{j,\beta}\}_{j\in[C],\beta\in[R\cdot Z]}, \mathsf{tk}\right)$.

[1] As before, position (i, α) refers to the α-th position of the i-th row, and position (j, β) refers to the β-th position of the j-th column.

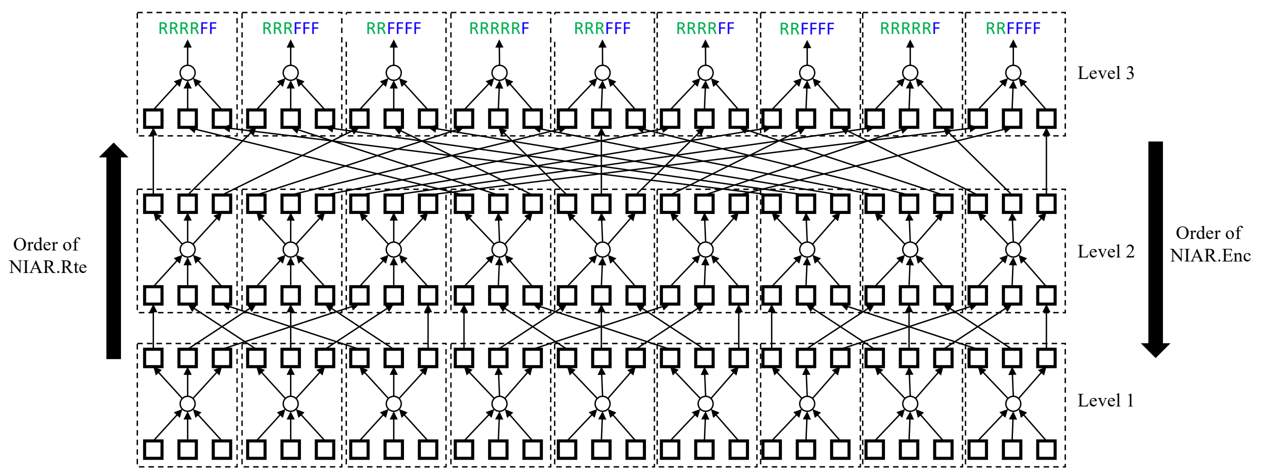

Figure 1: An R-way butterfly network when the recursion is fully expanded. In this example, $\mathsf{nLayer} = 3$, $2n/Z = 27$, and $R = (2n/Z)^{1/\mathsf{nLayer}} = 3$. The elements are being routed from level 1 to level L, and the onion layers of encryption are performed in the reverse order where level 1 is the outer-most layer. Each small box $\square$ denotes a bucket, and each dashed big box is either a $\mathsf{RowPerm}$ or a $\mathsf{ColPerm}$ instance. The last level is special and employs $\mathsf{ColPerm}$ instances which route the real elements to the front in a random order.

Subroutine $\mathsf{RecEnc}(\mathsf{ek}, x, t, \mathsf{nLayer})$**:** If $\mathsf{nLayer} = 1$, then let $\mathsf{CT} := \mathsf{NIAR}.\mathbf{Enc}(\mathsf{ek}, x, t)$ and return CT. Else,

1. parse $\mathsf{ek} := (\mathsf{ek}^{-}, \mathsf{ek}^{|})$;
2. let $\mathsf{ict} \leftarrow \mathsf{NIAR}.\mathbf{Enc}(\mathsf{ek}^{|}, x, t)$ and let $\mathsf{CT} \leftarrow \mathsf{RecEnc}(\mathsf{ek}^{-}, \mathsf{ict}, t, \mathsf{nLayer} - 1)$;
3. return CT.

Subroutine $\mathsf{RecRte}(\mathsf{tk}, \mathsf{CT}_1, \ldots, \mathsf{CT}_n, \mathsf{nLayer}, \mathsf{bFin})$**:**

- If $\mathsf{bFin} = 1$, then parse $\mathsf{tk} := \left(\{m_j\}_{j\in[C]}, \{\mathsf{tk}_i^{-}\}_{i\in[R]}, \{\mathsf{tk}_j^{|}\}_{j\in[C]}, \{\mathsf{rk}_{i,\alpha}^{-}\}_{i\in[R],\alpha\in[C\cdot Z]}\right)$; else parse $\mathsf{tk} := \left(\{\mathsf{tk}_i^{-}\}_{i\in[R]}, \{\mathsf{tk}_j^{|}\}_{j\in[C]}, \{\mathsf{rk}_{i,\alpha}^{-}\}_{i\in[R],\alpha\in[C\cdot Z]}\right)$.
- Let $R := (n/Z)^{1/\mathsf{nLayer}}$, $C := n/(R\cdot Z)$, and view $\mathsf{CT}_1, \ldots, \mathsf{CT}_n$ as an $(R \times C)$ matrix where entries are buckets of size Z. We shall use $\mathsf{CT}[i:]$ to denote the i-th row of the CT matrix.

  For each row $i \in [R]$, let $\overline{\mathsf{ict}}_{i,1}, \ldots, \overline{\mathsf{ict}}_{i,C\cdot Z} \leftarrow \mathsf{RecRte}(\mathsf{tk}_i^{-}, \mathsf{CT}[i:], \mathsf{nLayer} - 1, 0)$, and for each $\alpha \in [C\cdot Z]$, let $\mathsf{ict}_{i,\alpha} \leftarrow \mathsf{NIAR}.\mathbf{Dec}(\mathsf{rk}_{i,\alpha}^{-}, \overline{\mathsf{ict}}_{i,\alpha})$;
- View $\{\mathsf{ict}_{i,\alpha}\}_{i\in[R],\alpha\in[C\cdot Z]}$ also as a $(R \times C)$-matrix where each entry is a bucket of size Z — we shall use $\mathsf{ict}[:j]$ to denote the j-th column of this matrix.

  For each column $j \in [C]$, let $\mathsf{CT}'_{j,1}, \ldots, \mathsf{CT}'_{j,R\cdot Z} \leftarrow \mathsf{NIAR}.\mathbf{Rte}\left(\mathsf{tk}_j^{|}, \mathsf{ict}[:j]\right)$;
- If $\mathsf{bFin} = 1$, then view $\{\mathsf{CT}'_{j,\beta}\}_{j\in[C],\beta\in[m_j]}$ as a 1-dimensional array, and return the result.

  Else, then view $\{\mathsf{CT}'_{j,\beta}\}_{j\in[C],\beta\in[R\cdot Z]}$ as a 1-dimensional array, and return the result.

More general parameters. So far, we have assumed that $\left(\frac{2n}{Z}\right)^{1/L}$ is an integer. If not, we can let $R := \lceil\{\rceil \left(\frac{2n}{Z}\right)^{1/L}\}$, and this determines the structure of the routing network when the recursions are expanded all the way (see Figure 1). Moreover, in this indivisible case, each bucket will not all be of uniform capacity. It is not hard to ensure the invariant that every bucket's capacity is either Z or $Z+2$, and morever, all buckets' capacities are even. With the slightly modified bucket size, we need to modify the instance size for each NIAR instance accordingly. With this resulting algorithm, all of our analyses would still hold.

Efficiency. For our efficiency analysis, suppose that the underlying NIAR is instantiated with the construction of Shi and Wu [SW21]. We reviewed the asymptotic efficiency of the underlying NIAR scheme in Section 3.2. Recall that earlier, we used the notation $O_\lambda(\cdot)$ to hide $\mathsf{poly}(\lambda)$ parameters — in fact, this notation hides a multiplicative factor related to the length of each bilinear group element in the underlying NIAR. If we use $\kappa = \mathsf{poly}(\lambda)$ to denote the bit-length of a single bilinear group element in the underlying NIAR, then, $O_\lambda(\cdot)$ can be equivalently expressed as $O(\cdot)\cdot\kappa$. Note also that underlying NIAR's coding rate is $\Theta(\frac{1}{\kappa})$. i.e., the ratio of the ciphertext size and the plaintext size is $\Theta(\kappa)$.

Suppose the number of layers $L = O(1)$, and we now analyze the asymptotical performance bounds for our multi-layer NIDAR. Clearly, the receiver key is still $O(\kappa)$ in size like before. The sender key size is $O(\kappa \cdot R \cdot Z \cdot L) = O\left(\kappa \cdot \left(\frac{n}{Z}\right)^{1/L} \cdot Z\right)$. Each of the L layers incur a multiplicative κ factor blowup in the ciphertext size. Therefore, the per-sender ciphertext size as well as encryption runtime are $O(\kappa^L \cdot \mathsf{len})$. The Rte cost is $O(\kappa^L \cdot \frac{n}{R\cdot Z} \cdot (R\cdot Z)^2) = O\left(\kappa^L \cdot \left(\frac{n}{Z}\right)^{1/L} \cdot n \cdot Z\right)$.

Expanding the recursion. Recall that part of the goal of the recursive RecSetup algorithm is to sample the permutation π_{mid}, and this permutation is realized over multiple layers of routing. For ease of understanding as well as in our proofs, it is often be helpful to think about what actually happens when we fully expand the recursion out. The network structure looks like Figure 1 when we fully expand the recursion out. In this example, we assume that $2n/Z = 27$, and $R = 3$ in each level of the recursion. Each little box $\square$ represents a bucket. Each big dashed box represents a RowPerm or ColPerm instance.

The elements are routed from level 1 to level 3. Except for the last level which is a little special and uses ColPerm, for all other levels, each dashed box denotes a RowPerm instance. Inside each RowPerm instance, all real elements are thrown into random buckets, and if any bucket's load exceeds Z, simply throw an overflow exception. Next, for each remaining empty slot inside each bucket, we fill them with a random unconsumed filler element belonging to this instance. In the last level, each dashed box represents a ColPerm instance, which randomly permutes all elements moving all real elements to the front.

Our multi-layer NIDAR scheme is routing real elements as follows. In each RowPerm instance in levels 1 through $L-1$, the real elements are throw at random into buckets, and an overflow exception is thrown if any bucket receives more real elements than its capacity Z. In the last level, all the real elements are routed to the front arranged in a random order in each ColPerm instance.

The following lemma is a counterpart of Lemma 3.1 for the multi-layer case.

Lemma 4.1. *The output of the the above random process outputs a permutation of the real elements, and moreover, the resulting permutation has statistical distance at most $O(nL)\cdot\exp(-\Omega(Z))$ from a uniform random permutation.*

Proof. The proof is essentially identical to that of Lemma 3.1. $\square$

4.3 Proofs

Theorem 4.2 (L-layer NIDAR)**.** *Let $L \geq 2 = O(1)$ be the number of layers, and let $\mathcal{A}$ be an arbitrary non-uniform p.p.t. adversary that controls ρ fraction of the senders. Suppose that $(1-\rho)Z \geq \Theta(\log \frac{1}{\delta})$ where $\Theta(\cdot)$ hides a suitably large constant; further, suppose that the underlying* NIAR *scheme is SIM-secure. Then, there exists a negligble function* $\mathsf{negl}(\cdot)$*, for any S,*

$$\Pr[\mathsf{NIDAR\text{-}Expt}^0 \in S] \leq e^{\epsilon} \cdot \Pr[\mathsf{NIDAR\text{-}Expt}^1 \in S] + \delta'$$

where

$$\epsilon = \frac{\sqrt{Z \cdot \min((1-\rho),\rho)\log\frac{1}{\delta}} + \log\frac{1}{\delta}}{(1-\rho)Z}, \quad \delta' = O(n\cdot\delta) + \mathsf{negl}(\lambda)$$

The remainder of this section will be dedicated to the proof of Theorem 4.2. The proof of the multi-layer scheme is an extension of the two-layer proof. Recall that our multi-layer construction is recursive. If a NIAR instance is instantiated in a recursive call when the variable $\mathsf{nLayer} = \ell$, we say that this is a *level-ℓ* NIAR instance. Suppose that $R := (n/Z)^{1/L}$ is an integer. If one fully expands the recursion out, then there are R number of level-L instances, R^2 number of level-$(L-1)$ instances, and so on, and finally, there are R^L number of level-1 instances.

4.3.1 Sequence of Hybrids

We first define a sequence of hybrids.

Experiment $\mathsf{NIDAR\text{-}Expt}^0$. Same as the original $\mathsf{NIDAR\text{-}Expt}^0$ experiment as defined in Section 2. Henceforth, we may assume that during the experiment $\mathsf{NIDAR\text{-}Expt}^0$ that interacts with $\mathcal{A}$, the experiment samples all the randomness needed in all instances of $\mathsf{RowPerm}$ and $\mathsf{ColPerm}$ upfront for the entire recursion. In this way, it will be determined at the beginning of the experiment where all elements will be routed to in each instance of $\mathsf{RowPerm}$ or $\mathsf{ColPerm}$ when the recursion is fully expanded.

Experiment Hyb^0_L. Almost the same as $\mathsf{NIDAR\text{-}Expt}^0$ except that each level-L NIAR instance is replaced now with with a NIAR simulator. Recall that the NIAR's simulated **Setup** algorithm needs to know the destinations of all corrupt sources, and the NIAR's simulated **Enc** algorithm needs to know what message each corrupt destination receives in each time step. As mentioned earlier, since we assume that the experiment chooses all random coins needed by all $\mathsf{RowPerm}$ and $\mathsf{ColPerm}$ instances upfront, therefore, it is possible to pass to the NIAR simulator which are the corrupt sources in each level-L NIAR instances, and which are their destinations.

Claim 4.3. *Suppose that the* NIAR *scheme is SIM-secure. The adversary's views in* Hyb^L_1 *and* $\mathsf{NIDAR\text{-}Expt}^0$ *are computationally indistinguishable.*

Proof. Follows in a straightforward fashion due to the SIM-security of the underlying NIAR scheme and the hybrid argument. □

Experiment Hyb^0_{L-k} for $k \in [L-1]$. Hyb^0_{L-k} is almost identical to Hyb^0_{L-k+1}, except that in all level-$(L-k)$ instances, we replace each NIAR instance with a NIAR simulator.

Claim 4.4. *Suppose that the* NIAR *scheme is SIM-secure. The adversary's views in* Hyb^0_{L-k} *and* Hyb^0_{L-k-1} *are computationally indistinguishable for $k \in \{0, 1, \ldots, L-2\}$.*

Proof. Follows in a straightforward fashion due to the SIM-security of the underlying NIAR scheme and the hybrid argument. □

Experiment $\mathsf{Hyb}^0_\star$. $\mathsf{Hyb}^0_\star$ is a rewrite of Hyb^0_1, where we change how we sample the random coins. In $\mathsf{Hyb}^0_\star$, we introduce an *initial sampling phase* where a subset of the random coins and events are sampled. Then, based on the outcomes of these partial random coins and events, we invoke a *simulator* that completes the rest of the experiment including interactions with the adversary.

Henceforth, it is often helpful to think of the recursive algorithm that chooses the permutation π_{mid} as fully expanded out. Earlier in Section 4, we described what things look like when the recursion is fully expanded out, and how the permutation π_{mid} is chosen over multiple layers of routing.

Below we first describe the initial sampling phase.

1. Sample $m_1, m_2, \ldots, m_C$, by throwing n balls into $2n/(R \cdot Z)$ bins, and counting the bin loads.
2. Sample the complement permutation π' at random, and compute $\pi_{\text{mid}} := (\pi')^{-1} \circ \pi^{(0)}$, which is the permutation chosen by $\mathsf{RecSetup}$ (specifically, the simulated version where all NIAR instances are replaced with NIAR simulators). At this moment, the following random coins are fully determined:
   - which bucket each real element uR (either real or filler) should land in during each $\mathsf{RowPerm}$ instance in the expanded recursion, and if any bucket's load exceeds Z, return $\mathsf{overflow}$ just like before; and
   - the destination of each real element uR during each $\mathsf{ColPerm}$ instance.
3. Sample the number of corrupt filler elements for all the buckets in all $\mathsf{RowPerm}$ instances in the expanded recursion. To sample these random variables, we can go from level 1 to level $L-1$ in the expanded recursion, and in each level, for each $\mathsf{RowPerm}$ instance, recall that the destinations of the real elements have already been fixed when we sampled π'. We can now sample the destinations for all the filler elements. After this, we calculate the number of corrupt filler elements that land in each bucket during each $\mathsf{RowPerm}$ instance, and throw away the rest of the information we have sampled since they will be resampled again freshly by the simulator, in the next stage.

At this point, imagine we run the following simulator which continues to interact with the adversary:

Input:

1. set of corrupt senders $\mathcal{K}_S \subseteq [n]$ and set of corrupt receivers $\mathcal{K}_R \subseteq [n]$;
2. for every $u \in [n]$, if $(\pi^{(0)})^{-1}(u) \in \mathcal{H}_S$, what message the corrupt receiver u receives from some honest sender in each time step, based on the $\{x^{(0)}_{u,t}\}_{u,t}$ values.
3. the destination of every corrupt sender $u \in \mathcal{K}_S \subseteq [n]$ based on $\pi^{(0)}$;
4. $m_1, m_2, \ldots, m_C$;
5. π';
6. how many corrupt filler elements land in each bucket during each $\mathsf{RowPerm}$ instance in the expanded recursion.

Simulator algorithm. The simulator now performs the following:

- Consider the expanded recursion. Starting from level-1 to $L-1$, for every $\mathsf{RowPerm}$ instance, based on how many corrupt filler elements are to be received in each bucket during the $\mathsf{RowPerm}$ instance, randomly assign the corrupt filler elements belonging this $\mathsf{RowPerm}$ instance to the buckets;

  Recall that which bucket each corrupt real element should go during each $\mathsf{RowPerm}$ instance was already determined during the initial sampling phase. Therefore, at this time, the simulator knows which bucket each corrupt element (including real and filler) lands in during each $\mathsf{RowPerm}$ instance.
- For each bucket in each $\mathsf{RowPerm}$ instance, the simulator picks a random unconsumed position for each corrupt element that is supposed to go into this bucket during this $\mathsf{RowPerm}$ instance. At this moment, it is fully determined where all corrupt elements go during all $\mathsf{RowPerm}$ instances.
- For each $j \in [C]$, consider the $\mathsf{ColPerm}$ instance of column j in the last level of the recursion: for all the corrupt filler elements in column j belonging to this $\mathsf{ColPerm}$ instance, pick a random (non-overlapping) position among the last $R \cdot Z - m_j$ positions to be its destination. At this moment, the routes of all corrupt elements during all $\mathsf{RowPerm}$ and $\mathsf{ColPerm}$ instances are fully determined.
- At this moment, it is not hard to see that the simulator can accomplish the interactions with the adversary, since it knows all the inputs needed for calling the NIAR's simulators.

Claim 4.5. *The adversary's views in* Hyb_1^0 *and* $\mathsf{Hyb}_\star^0$ *are identically distributed.*

Proof. It is not difficult to check that $\mathsf{Hyb}_\star^0$ is simply a rewrite of Hyb_1^0, where the random coins are sampled in a different manner, by sampling a subset of the random coins and events first in an initial sampling stage, and then having a simulator accomplish the remaining. □

Experiment $\mathsf{Hyb}_\star^1$. $\mathsf{Hyb}_\star^1$ is almost identical to $\mathsf{Hyb}_\star^0$ except the following changes.

1. During the initial sampling stage: let $\pi_{\text{mid}} := (\pi')^{-1} \circ \pi^{(1)}$.
2. Part of the inputs to the simulator is changed to the following:
   - for every $u \in [n]$, if $(\pi^{(1)})^{-1}(u) \in \mathcal{H}_S$, what message the corrupt receiver u receives from some honest sender in each time step, based on the $\{x_{u,t}^{(1)}\}_{u,t}$ values;
   - the destination of every corrupt sender $u \in \mathcal{K}_S \subseteq [n]$ based on $\pi^{(1)}$.

Due to the admissibility rule on the adversary, for the above inputs to the simulator, it does not matter whether we use $\{x_{u,t}^{(0)}\}_{u,t}, \pi^{(0)}$ or $\{x_{u,t}^{(1)}\}_{u,t}, \pi^{(1)}$ — the outcomes are the same. In this sense, the only real difference in $\mathsf{Hyb}_\star^0$ and $\mathsf{Hyb}_\star^1$ is that π_{mid} is now computed as $(\pi')^{-1} \circ \pi^{(1)}$.

Lemma 4.6. *Suppose that* $(1-\rho)Z \geq \Theta(\log \frac{1}{\delta})$ *where* $\Theta(\cdot)$ *hides some appropriately large constant, where* ρ *is the fraction of corrupt senders. For any* S,

$$\Pr[\mathsf{view}_{\mathcal{A}}(\mathsf{Hyb}_\star^0) \in S] \leq e^\epsilon \cdot \Pr[\mathsf{view}_{\mathcal{A}}(\mathsf{Hyb}_\star^1) \in S] + \delta'$$

where $\mathsf{view}_{\mathcal{A}}(\mathsf{Hyb}_3^b)$ *denotes the adversary's view in experiment* $\mathsf{Hyb}_\star^b$ *for* $b \in \{0,1\}$, *and*

$$\epsilon = O(1) \cdot \frac{\sqrt{Z \cdot \min((1-\rho), \rho) \log \frac{1}{\delta}} + \log \frac{1}{\delta}}{(1-\rho)Z}, \quad \delta' = O(n\delta)$$

Proof. The full proof of this lemma is deferred to Section 4.3.2. □

Experiment Hyb^1_{L-k} for $k \in [L-1] \cup \{0\}$. Same as Hyb^0_{L-k} except that $\pi^{(1)}$ and $\{x^{(1)}_{u,t}\}_{u,t}$ are used in place of $\pi^{(0)}$ and $\{x^{(0)}_{u,t}\}_{u,t}$.

Claim 4.7. *Suppose that the* NIAR *scheme is SIM-secure. The adversary's views in* Hyb^1_{L-k} *and* Hyb^1_{L-k-1} *are computationally indistinguishable for* $k \in \{0, 1, \ldots, L-2\}$.

Proof. Symmetric to the proof of Claim 4.4. □

Experiment $\mathsf{NIDAR\text{-}Expt}^1$. Same as the original $\mathsf{NIDAR\text{-}Expt}^1$ experiment as defined in Section 2.

Claim 4.8. *Suppose that the* NIAR *scheme is SIM-secure. The adversary's views in* Hyb^1_L *and* $\mathsf{NIDAR\text{-}Expt}^1$ *are computationally indistinguishable.*

Proof. Symmetric to the proof of Claim 4.3. □

Proof of Theorem 4.2. The proof of Theorem 4.2 due to the standard hybrid lemma and Claims 4.3, 4.4, 4.5, 4.7, and 4.8, and Lemma 3.5.

4.3.2 Differential Privacy Lemma

Let $R = (2n/Z)^{1/\mathsf{nLayer}}$, and consider an R-way butterfly network like the one depicted earlier in Figure 1. Now, consider the following experiment DPExpt^b where $b \in \{0, 1\}$, in which the adversary's view capatures the simulator's input in the earlier hybrid experiment $\mathsf{Hyb}^b_\star$.

DPExpt^b

1. The adversary submits two neighboring permutations $\pi^{(0)}$ and $\pi^{(1)}$, i.e., the two permutations are otherwise identical except for swapping the destinations of two honest senders.
2. The challenger randomly chooses a set of corrupt senders $\mathcal{K}_S$ and tells the adversary the set $\mathcal{K}_S$.
3. The challenger samples $m_1, \ldots, m_C$ at random as mentioned before and tells the adversary these values.
4. The challenger samples a complement random π'. The challenger computes $\pi_{\text{mid}} = (\pi')^{-1} \circ \pi^{(b)}$. Tell the adversary the complement permutation π'.
5. At this moment, it is fully determined that in the each RowPerm instance in R-way butterfly network, which bucket each real element will land. If any bucket exceeds Z real elements, abort throwing overflow.
6. For all RowPerm instances, simulate how many corrupt filler elements land in each bucket during this instance. Tell the adversary the number of corrupt filler elements that land in each bucket for all RowPerm instances.
7. Return the adversary's view.

Lemma 4.9. *Suppose that the total number of levels* $L = O(1)$. *Lemma 3.10 still holds for the new definition of* DPExpt^0 *and* DPExpt^1.

Proof. Like in the proof of Lemma 3.10, we define three types of coins, corruption coins, routing coins, and filler coins. Their definitions are the same as before.

Once we fix the routing coins and corruption coins, the number filler elements in all buckets are fixed. Henceforth let $\widetilde{\mathbf{M}}^b_{\ell,r}$ denote the filler loads in all buckets of the r-th RowPerm instance in level ℓ, after this RowPerm instance finishes its routing in DPExpt^b. For each level $\ell \in [L-1]$, it must be that there are at most four r's where $\widetilde{\mathbf{M}}^0_{\ell,r} \neq \widetilde{\mathbf{M}}^1_{\ell,r}$. Henceforth, for each RowPerm instance indexed by (ℓ, r) such that $\widetilde{\mathbf{M}}^0_{\ell,r} \neq \widetilde{\mathbf{M}}^1_{\ell,r}$, we call such an instance a *distinguishing* instance. For each distinguishing instance (ℓ, r), there are the following possible scenarios:

1. there exists j_0 and j_1 such that the filler load vector becomes $\{M_1, M_2, \ldots, M_{j_b}+1, \ldots, M_R\}_{j\in[R]}$ in this RowPerm instance in DPExpt^b for $b \in \{0,1\}$;
2. there exists j_0 the filler load vector becomes $\{M_1, M_2, \ldots, M_{j_0}+1, \ldots, M_R\}_{j\in[R]}$ in this RowPerm instance in DPExpt^0, and becomes $\{M_1, M_2, \ldots, M_{j_0}, \ldots, M_R\}_{j\in[R]}$ in this RowPerm instance in DPExpt^1;
3. there exists j_1 the filler load vector becomes $\{M_1, M_2, \ldots, M_{j_1}+1, \ldots, M_R\}_{j\in[R]}$ in this RowPerm instance in DPExpt^1, and becomes $\{M_1, M_2, \ldots, M_{j_1}, \ldots, M_R\}_{j\in[R]}$ in this RowPerm instance in DPExpt^0.

Henceforth, we may assume that for every (ℓ, r), $\widetilde{\mathbf{M}}^b_{\ell,r} = \mathbf{M}_{\ell,r} + \mathbf{\Delta}^b_{\ell,r}$ for some $\mathbf{M}_{\ell,r}$ and $\mathbf{\Delta}^b_{\ell,r}$ such that

1. for every RowPerm instance (ℓ, r) that is not distinguishing, $\mathbf{\Delta}^0_{\ell,r} = \mathbf{\Delta}^1_{\ell,r} = \mathbf{0}$;
2. for a distinguishing RowPerm instance (ℓ, r), it must be one of the following cases:
   (a) for $b \in \{0,1\}$, there is a j_b such that $\Delta^b_{\ell,r,j_b} = 1$; all other coordinates in $\mathbf{\Delta}^b_{\ell,r}$ are 0;
   (b) $\mathbf{\Delta}^1_{\ell,r} = \mathbf{0}$; moreover, there is a j_0 such that $\Delta^0_{\ell,r,j_0} = 1$, and all other coordinates in $\mathbf{\Delta}^0_{\ell,r}$ are 0;
   (c) $\mathbf{\Delta}^0_{\ell,r} = \mathbf{0}$; moreover, there is a j_1 such that $\Delta^1_{\ell,r,j_1} = 1$, and all other coordinates in $\mathbf{\Delta}^1_{\ell,r}$ are 0.

Below is the new counterpart of Claim 3.11. Henceforth we will often index a bucket by a tuple (ℓ, r, j) meaning that it is the j-th bucket in the RowPerm instance indexed by (ℓ, r).

Claim 4.10. *Suppose $(1-\rho)Z \geq \Theta'(\log \frac{1}{\delta})$ where $\Theta'(\cdot)$ hides a sufficiently large constant. It must be that for any fixed bucket indexed by (ℓ, r, j), with probability at least $1-\delta$, $M_{\ell,r,j} = \Theta(Z)$ where $\Theta(\cdot)$ hides an appropriately large constant.*

Proof. Due to a straightforward application of the Chernoff bound. □

The following claim will be used as a counterpart of Claim 3.12 and Lemma 3.13 in the multi-layer case.

Claim 4.11. *Suppose that $(1-\rho)Z \geq \Theta_1(\log \frac{1}{\delta})$ where $\Theta_1(\cdot)$ hides a sufficiently large constant and $L = O(1)$. For every fixed bucket in level 1, the fraction of filler elements that are honest among fillers is at least $\Theta_2(1-\rho)$, except with $\frac{1}{\delta}$ probability over the choice of the corruption coins.*

Further, suppose that in some RowPerm instance denoted (ℓ, r), the fraction of filler elements that are honest is $\Theta_3(1-\rho)$, and suppose that every coordinate in $\mathbf{M}_{\ell,r}$ satisfies the good event of Claim 4.10. Then, for any fixed bucket within this RowPerm instance denoted (ℓ, r, j), with $1-\frac{1}{\delta}$ probability over the choice of the filler coins of this RowPerm instance,

- *$M_{\ell,r,j} - \mu_{\ell,r,j} \geq \Theta_4((1-\rho)Z)$ where $\mu_{\ell,r,j}$ is the number of corrupt filler elements that land in the bucket indexed by $_{\ell,r,j}$ during the* RowPerm *instance indexed by (ℓ, r);*
- *among the filler elements that land in the bucket indexed by $_{\ell,r,j}$ during the* RowPerm *instance indexed by (ℓ, r), the fraction of honest elements is at least $\Theta_5(1-\rho)$;*
- *$\mu_{\ell,r,j} < M_{\ell,r,j} - 1$.*

Proof. The statement about the first level follows due to a straightforward application of the Chernoff bound. The rest of the claim can be proven in a similar fashion as that of Lemma 3.13 due to negative association and the Chernoff bound, and observing that since the constants are blown up over only $O(1)$ levels, they remain constants. $\square$

Henceforth, we use rc to denote some specific choice of the routing and corruption coins. We use $\boldsymbol{\mu}_{1:\ell}$ to denote some specific choice of the corrupt filler load vectors of all levels from 1 to ℓ.

Lemma 4.12. *Fix some good choice of rc that satisfies the good events of Claims 4.10 for all buckets. Further, consider an arbitrary $\ell \in [L-1]$ and fix some good choice of $\boldsymbol{\mu}_{1:\ell}$ such that given rc and $\boldsymbol{\mu}_{1:\ell}$, the good events of Claim 4.11 hold for all buckets in levels 1 to ℓ. Now, consider some distinguishing* RowPerm *instance indexed by (ℓ, r), taking probability over the filler coins of the instance (ℓ, r), we have that*

$$\frac{\Pr_0[\boldsymbol{\mu}_{\ell,r} | rc, \boldsymbol{\mu}_{1:\ell-1}]}{\Pr_1[\boldsymbol{\mu}_{\ell,r} | rc, \boldsymbol{\mu}_{1:\ell-1}]} \leq e^{\epsilon} \quad \textit{where } \epsilon := O''\left(\frac{\sqrt{Z \cdot \min((1-\rho),\rho)\log\frac{1}{\delta}} + \log\frac{1}{\delta}}{(1-\rho)Z}\right)$$

Proof. Recall that there are three cases for a distinguishing instance.

For case (a), the lemma follows due to the same analysis as Lemma 3.14 and the proof of Lemma 3.10. For cases (b) and (c), we will instead use Lemma 4.13 (to be proven later) in place of Lemma 3.10. For case (b), we have

$$\frac{\Pr_0[\boldsymbol{\mu}_{\ell,r} | rc, \boldsymbol{\mu}_{1:\ell-1}]}{\Pr_1[\boldsymbol{\mu}_{\ell,r} | rc, \boldsymbol{\mu}_{1:\ell-1}]} \leq \frac{1 - \frac{\sum_j \mu_{\ell,r,j}}{\sum_j M_{\ell,r,j}+1}}{1 - \frac{\mu_{\ell,r,j_0}}{M_{\ell,r,j_0}+1}}$$

For case (c), we have

$$\frac{\Pr_0[\boldsymbol{\mu}_{\ell,r} | rc, \boldsymbol{\mu}_{1:\ell-1}]}{\Pr_1[\boldsymbol{\mu}_{\ell,r} | rc, \boldsymbol{\mu}_{1:\ell-1}]} \leq \frac{1 - \frac{\mu_{\ell,r,j_1}}{M_{\ell,r,j_1}+1}}{1 - \frac{\sum_j \mu_{\ell,r,j}}{\sum_j M_{\ell,r,j}+1}}$$

In both cases (b) and (c), plugging in Claims 4.10 and 4.11, it is not hard to see that the same calculation steps in the proof of Lemma 3.10 — specifically, Equation (♠) — still hold here. Thus we arrive at the statement claimed. $\square$

Now, consider an arbitrary set $S = \{\boldsymbol{\mu}\}$ of choices of $\boldsymbol{\mu}$'s, where $\boldsymbol{\mu}$ denotes the vector of corrupt filler loads of all buckets. Let rc be some choice of routing and corruption coins. We use $\mathsf{G}(rc, \boldsymbol{\mu})$ to denote the event that given rc the resulting $\mathbf{M}_{\ell,r}$'s and $\boldsymbol{\mu}$ satisfy the good events of Claims 4.10 and 4.11.

We have

$$\begin{aligned}
&\Pr_0[S] = \sum_{\boldsymbol{\mu}\in S}\sum_{rc}\Pr_0[\boldsymbol{\mu}, rc] \\
=&\sum_{\boldsymbol{\mu}\in S}\sum_{rc:\mathsf{G}(rc,\boldsymbol{\mu})}\Pr_0[\boldsymbol{\mu}, rc] + \sum_{\boldsymbol{\mu}\in S}\sum_{rc:\neg\mathsf{G}(rc,\boldsymbol{\mu})}\Pr_0[\boldsymbol{\mu}, rc] \\
=&\sum_{\boldsymbol{\mu}\in S}\sum_{rc:\mathsf{G}(rc,\boldsymbol{\mu})}\Pr[rc]\cdot\Pr_0[\boldsymbol{\mu}_1|rc]\cdot\Pr_0[\boldsymbol{\mu}_2|rc,\boldsymbol{\mu}_{1:1}]\cdot\Pr_0[\boldsymbol{\mu}_3|rc,\boldsymbol{\mu}_{1:2}]\ldots\cdot\Pr_0[\boldsymbol{\mu}_{L-1}|rc,\boldsymbol{\mu}_{1:L-2}] + O(n)\cdot\log\frac{1}{\delta} \\
\le&\sum_{\boldsymbol{\mu}\in S}\sum_{rc:\mathsf{G}(rc,\boldsymbol{\mu})} e^{2(L-1)\cdot\epsilon}\cdot\Pr[rc]\cdot\Pr_1[\boldsymbol{\mu}_1|rc]\cdot\Pr_1[\boldsymbol{\mu}_2|rc,\boldsymbol{\mu}_{1:1}]\ldots\cdot\Pr_1[\boldsymbol{\mu}_{L-1}|rc,\boldsymbol{\mu}_{1:L-2}] + O(n)\cdot\log\frac{1}{\delta} \\
\le& e^{4(L-1)\cdot\epsilon}\cdot\Pr_1[S] + O(n)\cdot\log\frac{1}{\delta}
\end{aligned}$$

Note that in the last but second inequality, the $e^{4(L-1)\cdot\epsilon}$ term comes from the fact that there are $L-1$ levels, and moreover, for each level, there are at most 4 distinguishing instances. Furthermore, the $O(n)$ factor in the $O(n)\cdot\log\frac{1}{\delta}$ term comes from taking a union bound over all buckets.

□

Analysis of the sampling mechanism (variant). We consider the same sampling mechanism as before, except that 1) the attributes are chosen from $[R]$ rather than $[C]$; and 2) the two neighboring database are now $\mathsf{DB} := (M_1, M_2, \ldots, M_k, \ldots, M_R)$ and $\mathsf{DB}' := (M_1, M_2, \ldots, M_k + 1, \ldots, M_R)$.

Lemma 4.13. *Given any $(\mu_1, \ldots, \mu_R)$ vector where $\mu_j \le M_j$ for $j \in [R]$,*

$$\frac{\Pr_{\mathsf{DB}}[\mu_1,\ldots,\mu_R]}{\Pr_{\mathsf{DB}'}[\mu_1,\ldots,\mu_R]} = \frac{1-\frac{\mu_k}{M_k+1}}{1-\frac{s}{M+1}}$$

Proof. We have

$$\Pr_{\mathsf{DB}}[\mu_1,\ldots,\mu_R] = \frac{\binom{M_k}{\mu_k}\cdot\Pi_{j\in[R],j\neq k}\binom{M_j}{\mu_j}}{\binom{M}{s}}$$

$$\Pr_{\mathsf{DB}'}[\mu_1,\ldots,\mu_R] = \frac{\binom{M_k+1}{\mu_k}\cdot\Pi_{j\in[R],j\neq k}\binom{M_j}{\mu_j}}{\binom{M+1}{s}}$$

where $M := \sum_{j\in[R]} M_j$ and $s = \sum_{j\in[R]}\mu_j$.

Therefore,

$$\frac{\Pr_{\mathsf{DB}}[\mu_1,\ldots,\mu_R]}{\Pr_{\mathsf{DB}'}[\mu_1,\ldots,\mu_R]} = \frac{\binom{M_k}{\mu_k}\cdot\binom{M+1}{s}}{\binom{M_k+1}{\mu_k}\cdot\binom{M}{s}} = \frac{M+1}{M+1-s}\cdot\frac{M_k+1-\mu_k}{M_k+1} = \frac{1-\frac{\mu_k}{M_k+1}}{1-\frac{s}{M+1}}$$

□

Acknowledgments

We would like to thank Bruce Maggs for helpful discussions about permutation networks and oblivious routing. We also thank Aron Laszka for helpful discussions.

References

[Abe99] Masayuki Abe. Mix-networks on permutation networks. In *ASIACRYPT*, 1999.

[AIVG20] Kinan Dak Albab, Rawane Issa, Mayank Varia, and Kalman Graffi. Batched differentially private information retrieval. Cryptology ePrint Archive, Report 2020/1596, 2020. https://ia.cr/2020/1596.

[APY20] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder: Mpc based scalable and robust anonymous committed broadcast. In *ACM CCS*, 2020.

[BCG$^+$14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE S & P*, 2014.

[BG12] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Eurocrypt*, volume 7237, pages 263–280, 2012.

[BHKP16] Michael Backes, Amir Herzberg, Aniket Kate, and Ivan Pryvalov. Anonymous ram. In *ESORICS*, 2016.

[CBM15] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *S & P*, 2015.

[CFN90] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *CRYPTO*, 1990.

[CGF10] Henry Corrigan-Gibbs and Bryan Ford. Dissent: Accountable anonymous group messaging. In *CCS*, page 340–350, 2010.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, February 1981.

[Cha82] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO*, 1982.

[Cha88] David L. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, March 1988.

[CM06] Kamalika Chaudhuri and Nina Mishra. When random sampling preserves privacy. In *CRYPTO*, 2006.

[DD08] George Danezis and Claudia Diaz. A survey of anonymous communication channels. Technical Report MSR-TR-2008-35, Microsoft Research, 2008.

[Dia21] Benjamin E Diamond. Many-out-of-many proofs and applications to anonymous zether. In *IEEE S & P*, 2021.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, 2004.

[DR14] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[EY09] Matthew Edman and Bülent Yener. On anonymity in an electronic society: A survey of anonymous communication systems. *ACM Comput. Surv.*, 42(1), December 2009.

[GIKM00] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *J. Comput. Syst. Sci.*, 60(3), 2000.

[GRS99] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42:39–41, 1999.

[HAB$^+$17] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *NDSS*, 2017.

[HKP20] Yuncong Hu, Sam Kumar, and Raluca Ada Popa. Ghostor: Toward a secure data-sharing system from decentralized trust. In *NSDI*, 2020.

[HMPS14] Susan Hohenberger, Steven A. Myers, Rafael Pass, and Abhi Shelat. ANONIZE: A large-scale anonymous survey system. In *IEEE S & P*, 2014.

[HOWW19] Ariel Hamlin, Rafail Ostrovsky, Mor Weiss, and Daniel Wichs. Private anonymous data access. In *Eurocrypt*, 2019.

[LGZ18] David Lazar, Yossi Gilad, and Nickolai Zeldovich. Karaoke: Distributed private messaging immune to passive traffic analysis. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 711–725, USA, 2018. USENIX Association.

[LYK$^+$19] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *CCS*, 2019.

[OS97] Rafail Ostrovsky and Victor Shoup. Private information storage (extended abstract). In *STOC*, pages 294–303, 1997.

[RMSK14] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. Coinshuffle: Practical decentralized coin mixing for bitcoin. In *ESORICS*, 2014.

[RMSK17] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. P2p mixing and unlinkable bitcoin transactions. In *NDSS*, 2017.

[SSA$^+$18] Fatemeh Shirazi, Milivoj Simeonovski, Muhammad Rizwan Asghar, Michael Backes, and Claudia Diaz. A survey on routing in anonymous communication protocols. 2018.

[SW21] Elaine Shi and Ke Wu. Non-interactive anonymous router. In *Eurocrypt*, 2021.

[TDG16] Raphael R. Toledo, George Danezis, and Ian Goldberg. Lower-cost epsilon-private information retrieval. *Proc. Priv. Enhancing Technol.*, 2016(4):184–201, 2016.

[TGL$^+$17] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *SOSP*, 2017.

[Vad17] Salil Vadhan. The complexity of differential privacy. 2017.

[vdHLZZ15] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *SOSP*, 2015.

[ZZZR05] Li Zhuang, Feng Zhou, Ben Y. Zhao, and Antony Rowstron. Cashmere: Resilient anonymous routing. In *NSDI*, 2005.