sciendo

Proceedings on Privacy Enhancing Technologies ; 2022 (1):49–74

Mingyu Liang*, Ioanna Karantaidou, Foteini Baldimtsi, S. Dov Gordon, and Mayank Varia

(ϵ, δ)-Indistinguishable Mixing for Cryptocurrencies

Abstract: We propose a new theoretical approach for building anonymous mixing mechanisms for cryptocurrencies. Rather than requiring a fully uniform permutation during mixing, we relax the requirement, insisting only that neighboring permutations are similarly likely. This is defined formally by borrowing from the definition of differential privacy. This relaxed privacy definition allows us to greatly reduce the amount of interaction and computation in the mixing protocol. Our construction achieves $O(n \cdot \text{polylog}(n))$ computation time for mixing n addresses, whereas all other mixing schemes require $O(n^2)$ total computation across all parties. Additionally, we support a smooth tolerance of fail-stop adversaries and do not require any trusted setup. We analyze the security of our generic protocol under the UC framework, and under a stand-alone, game-based definition. We finally describe an instantiation using ring signatures and confidential transactions.

Keywords: Anonymous Mixing, Cryptocurrency, Differential Privacy

DOI 10.2478/popets-2022-0004
Received 2021-05-31; revised 2021-09-15; accepted 2021-09-16.

1 Introduction

Cryptocurrencies are the main application of blockchain technologies. Starting with Bitcoin back in 2008 [48], a large number of different types of cryptocurrencies have emerged, with the total number of distinct available cryptocurrencies today getting close to 3000. Existing cryptocurrencies share a market capital of over $1.5 trillion [20] and a total of over 1 million daily transactions [56]. However, despite the high adoption rates and the importance of cryptocurrencies in the current financial ecosystem, user privacy is repeatedly overlooked.

In any typical blockchain based cryptocurrency, all user transactions are public and stored in the blockchain forever. In systems like Bitcoin, the user identity is never posted on the blockchain; instead, users are represented by random looking addresses (or else *pseudonyms*), which in the early years of Bitcoin created the perception that Bitcoin transactions were anonymous. Today, however, we are well aware that this is not true. Bitcoin transactions are only *pseudoanonymous*, and they are *linkable* to each other. Not only can any observer of the blockchain trivially link together transactions involving a specific address, but they can also cluster together sets of addresses as belonging to the same user by relying on simple transaction patterns, such as receiving change amounts, merging addresses together, and so on. This linkability, when combined with external data, such as data from exchanges or ground truth data, or when aided by active attacks such as "dusting attacks" [22], can result in complete de-anonymization of users [2, 42, 52]. The trend of linking transactions and de-anonymizing users has followed in other cryptocurrencies that were built using the same mechanics as Bitcoin, such as Ethereum, Litecoin, BitcoinCash [21, 36] as well as cryptocurrencies constructed under very different principles such as Ripple [45].

One approach to providing anonymity is through the use of mixing protocols. The study of mixing protocols has a long history in cryptography, first in the context of anonymous messaging and electronic voting, and then with applications in secure multiparty computation. Broadly, and without attempting to be exhaustive, protocols for mixing fall into a few categories . In mix chains, each party takes a turn shuffling the messages. These are especially useful when there is some small set of servers that are trusted to contain an honest participant. However, when n parties are involved in the mix, the communication complexity is $O(n^2)$, and the round complexity is $O(n)$, as each party takes a turn, sequentially, with mixing the values and forwarding them along. Another approach to mixing is to use secure computation to obliviously evaluate a switching

***Corresponding Author: Mingyu Liang:** George Mason University, E-mail: mliang5@gmu.edu
Ioanna Karantaidou: George Mason University, E-mail: ikaranta@gmu.edu
Foteini Baldimtsi: George Mason University, E-mail: foteini@gmu.edu
S. Dov Gordon: George Mason University, E-mail: gordon@gmu.edu
Mayank Varia: Boston University, E-mail: varia@bu.edu

network. While we have efficient constant-round protocols for obliviously evaluating circuits, these protocols require $O(n^2)$ communication (independent of the circuit size).[1] Additionally, it is not immediately clear how to efficiently implement a switching network in the context of crypto-currencies, as "moving" coins requires knowledge of a secret signing key, and moving them obliviously would require creating signatures without anyone, including the key holder or the recipient, learning who signed and who received the coin. Obliviously accessing these keys at each step of the circuit would be prohibitively expensive.

In the setting of cryptocurrencies, most *decentralized* mixing mechanisms [44, 54] are designed as peer-to-peer (P2P) protocols, and allow a set of mutually distrusting peers to publish their messages anonymously without requiring any external party. Such protocols are often realized through the use of DC-nets [19], which require $O(n^2)$ computational cost, and a total of $O(n^2)$ communication when n parties are mixing their coins (dominated by the underlying anonymous broadcast). They are also highly interactive, requiring $O(n)$ rounds if there is a constant fraction of corrupted parties. Concretely, decentralized mixing protocols, such as Coinshuffle(++) [53, 54], require heavy coordination among the participating parties.

The Need for Large Scale Mixing. Summing up the previous discussion, all known, fully secure mixing protocols require $O(n^2)$ communication, and all that are amenable to mixing cryptocurrency have round complexity $O(n)$. The high computation and communication costs of decentralized mixing protocols has lead to the adoption of small scale mixing, in the hopes that a little privacy is better than no privacy. For example, as reported in [26], when the CoinJoin protocol was used on top of Bitcoin in the 2015-2017 period, it had a mean of 3.98 coins in each transaction, and a standard deviation of 1.72. At the same time, it has been shown that a small amount of auxiliary information, e.g. that two or more mixed coins belong to the same entity (potentially learned through cookie tracking), can lead to cluster intersection attacks, removing the privacy gained from mixing [26]. Thus, it has been demonstrated that small scale mixing can be useless. In fact, it is not hard to imagine a scenario where small scale mixing can do more harm than good. For example, imagine Alice believes that Bob is spending his coin at a particular local coffee shop, but this location accepts hundreds of coins on any given day. Alice then finds out that Bob has participated in a small mix involving a few thousand coins from around the world. If any coin from that mix is later spent at the location of interest, it almost certainly belongs to Bob, as it is unlikely that many other people in Bob's town participated in the same mixing protocol.

Although we do not make a formal claim along these lines, the intuition is clear: we need large-scale mixing. To *entirely* remove the value of auxiliary information, we must mix the full network of unspent coins (UTXOs) before any of those coins are spent again. Currently, the Bitcoin network has about 68 million UTXOs. Scaling current approaches to such a large number of coins seems prohibitive, and finding protocols with asymptotic improvement on this very old problem would represent a major breakthrough.

1.1 Our Contributions

We propose a new approach for mixing at scale. Rather than relaxing security by reducing the size of the mix, we propose to *relax the distribution over the shuffling permutations*. Inspired by the definition of differential privacy [23], we propose the definition of (ϵ, δ)-indistinguishability, which requires that for all "neighboring" permutations, any observed leakage from the mixing using these permutations should be similarly likely. That is, for all pairs of input addresses $(\mathsf{s_a}, \mathsf{s_b})$ and output addresses $(\mathsf{r_c}, \mathsf{r_d})$, an observer of the mixing protocol should learn little, statistically, about whether sender $\mathsf{s_a}$ pays receiver $\mathsf{r_c}$ and $\mathsf{s_b}$ pays $\mathsf{r_d}$, or whether $\mathsf{s_a}$ pays $\mathsf{r_d}$ and $\mathsf{s_b}$ pays $\mathsf{r_c}$. Although traditionally used to bound leakage when querying datasets, differential privacy has inspired privacy definitions in many new application domains, such as secure computation [30, 41], oblivious computation in a client/server model [5, 18, 34, 62], and anonymous messaging [38, 57, 60].

We build the first cryptocurrency mixing mechanism inspired by differential privacy, where two neighboring mixing permutations give rise to nearly the same set of transactions. Our protocol requires $O(n \log n)$ total communication when n users are mixing coins, has sub-quadratic total computation cost, and requires only constant or polylog(n) communication rounds (depending on which protocol variant is chosen). In addition to our constructions, we provide the first definitions for anonymous cryptocurrency mixing with leakage. We believe that this weakening of the anonymity definition

1 One exception is the work of Movahedi et al. [47], which uses quorums in the MPC to avoid the $O(n^2)$ term but still suffers from the efficient signing problem we mention.

may open the door to new protocol constructions, beyond the ones described here.

1.2 Overview of Our Construction

Our construction is a distributed mixing mechanism for n users, based on an anonymous transaction building block, which allows transferring a hidden value from an address s_p to an address r_p, such that the link between sender and receiver is obfuscated. We provide a rigorous definition of this functionality in Section 3. Such a building block can be realized via the use of ring signatures and confidential transactions, as done in Monero [43] (see Section 1.3), or via the use of generic ZK arguments. A naive solution for mixing, if we implemented anonymous transactions with ring signatures, would require every participant to create a spending transaction with ring size n (in order to ensure an anonymity set of size n). Although fully secure, such a naive solution has quadratic computation and verification costs, and $n \log n$ total communication cost (if a state of the art logarithmic-size ring signature [37, 63] is used).

As discussed above, in order to achieve sub-quadratic overall computation cost (i.e., sublinear for each mixing participant) we relax the level of privacy by allowing some leakage. We divide the mixing procedure into multiple rounds of smaller mixings (which is a typical pattern for various mixing protocols). In each round, every participant creates a transaction that mixes her coin with a small subset of the total set of participants. To determine the mix groups for each round, we rely on the structure of an n-size *butterfly network*. What is nice about butterfly networks is that they provide a path from each of the n input nodes to the n output nodes while ensuring that the fan-in, k, and the depth $d = \log_k(n)$ are sub-linear in n.

At a high level, our protocol (detailed in Section 4) works as follows. Each participant owns a pair of addresses: a source address in which their currency resides at the start of the protocol, and target address that will hold the currency after the mixing is complete: $(\mathsf{s}_p, \mathsf{r}_p)$. These source address is assigned a unique node at the input layer of the butterfly network, and the target is placed randomly in one of the output nodes of the network (possibly sharing that node with another target address). Participants transfer their currency from the source to the target by creating and placing public keys in the intermediate nodes, and making a series of anonymous transactions to these keys, one for each level of the tree, according to the network topology. In the typical use of butterfly networks, every item takes a disjoint path to its destination. In our construction, in order to avoid the need for further coordination, we allow multiple users to pass their currency through the same network nodes by allowing multiple parties to assign addresses in the same node (with the exception of the input layer). We therefore refer to those intermediate and output nodes as *buckets*.

Each participant only handles her own currency, sending it along the path from the source to the target. The use of anonymous transactions for moving the value through the network provides some privacy, but does not suffice for any formal guarantee: we also need to hide the number of addresses in each bucket, as this reveals information about the permutation being used. The number of addresses in each bucket can be viewed as a histogram, which is the canonical problem for the application of differential privacy. We add "noisy" addresses in every bucket to hide the number of real addresses. Noisy addresses will be contributed distributively, by all participants, and without the need of coordination, by using a noise distribution that is additive (i.e. negative binomial distribution/Polya distribution).

Protocol Complexity: We explore butterfly networks with different branch factors and depths. If we apply a butterfly network with only one intermediate layer of buckets (what we call the "one-layer case"), we achieve a sub-quadratic expected computation cost of $O(n^{1.5})$ for all participants. The communication cost, if instantiated with logarithmic size ring signature, is $O(n \log n)$ for all participants, as in the naive case. Using multi-layer butterfly networks can further reduce computation cost, but at the cost of increased communication cost. For example, by using a $\log n$ depth butterfly network (for the 2-ary case), we have $O(n \log^3 n)$ expected computational cost and $O(n \log^2 n \log \log n)$ expected communication cost. Recall that the naive construction requires $O(n^2)$ computation, and, as we will discuss in Section 1.3, the total computation cost for n transactions in Monero, or an n size mixing mechanism like Coinshuffle++ is quadatric in n.

Through a clever improvement in which we merge certain buckets in the one-layer construction, we manage to "reuse" some noise, substantially improving the parameters. Concretely, for privacy parameters $\epsilon = 2.303$ and $\delta = 10^{-4}$ (a common selection in the literature), we require an average of 37 noisy addresses in each bucket. By a proof-of-concept evaluation, we show that when $n > 605$, we achieve better concrete computational efficiency when compared with the naive solution. We also show, that the communication cost is bounded

within three times that of the naive approach when using log-size ring signatures. See Section 6 for details.

Security: In Section 3, we provide two security definitions for our mixing protocol: a simulation-based one in the universally composable (UC) security framework for the case of mixing coins of equal value, and a game-based definition that supports mixing of variable values and that explicitly specifies the adversary's advantage in distinguishing neighboring permutations. For our simulation-based definition, we design an anonymous mixing functionality $\mathcal{F}^{T}_{\mathsf{AnonMix}}$ that, after receiving a number of input and output addresses from participants, performs mixing at predetermined time slots T. The mixing functionality $\mathcal{F}^{T}_{\mathsf{AnonMix}}$ leverages existing functionalities for a bulletin board and a globally-accessible clock.

In Section 5 we show that our protocol is secure against malicious behavior with any number of corruptions (for both our definitions). Intuitively, since each participant handles her funds all the way to the destination, there is no way for a malicious party to steal money. Colluding users can decrease the anonymity set size by the number of colluding parties, but nothing else is revealed. Furthermore, our protocol is robust against aborts, unlike many existing coordination-based protocols that require abandoning the protocol if a malicious party aborts. A user can drop out at any time, and the remaining users can complete the protocol without restarting with a slightly reduced anonymity set.

Composition: A common concern with differentially private protocols is the deterioration of the privacy parameter through the composition of multiple executions. We prove that this is not an issue for our protocol. The input to our (ϵ, δ)-indistinguishable mixing mechanism is the random permutation used for mixing. Each time the protocol starts, we choose another uniform, independent permutation as input; the previous permutation is discarded and never used again for further mixing. Intuitive, repetitive mixing, even a weak one, adds randomness to the composed mixing results. We formalize this concept as an iterated butterfly network in Appendix G and prove that it provides *more* privacy, not less in there. Surprisingly, this iterated approach can also improve efficiency when epsilon is already small. Specifically, instead of relying on larger noise for better privacy guarantee, it is more efficient to keep the original noise magnitude but shuffle multiple times. In contrast, when differential privacy has been used in an application like anonymous messaging [57, 60], the input across executions is correlated in a dangerous way, since pairs of parties that communicate today are likely to communicate again tomorrow.

Moving Towards Practice: In Section 7, we provide a strengthening of our construction that handles transaction fees and provides resilience against DDoS attacks. In Appendix B, we discuss an instantiation of our generic mixing construction built upon any anonymous transaction functionality, such as Monero's ring signatures and confidential transactions. For our instantiation, we modify the ring signature content in order to support loop-payments, i.e. the ability for an input address to transfer amounts back to itself. Then, we describe the ring's structure as predefined by the parent buckets in the previous level and the output address for both real and noisy transactions, and we claim full indistinguishability between them. We present a variant of our construction in Section 7 that is compatible with current signature and fee requirements of Monero.

1.3 Related Work

The community has reacted to the privacy shortcomings of cryptocurrencies by providing a number of solutions that can be classified under two main categories: *mixing mechanisms* for existing, already deployed cryptocurrencies (e.g., [9, 31, 40, 44, 53–55]) that require $O(n^2)$ communication and computation as described earlier, and new *stand-alone private cryptocurrencies* [7, 43, 61, 64] that employ cryptographic mechanisms to hide the transaction value and participants. Compared to these works, our protocol relies only on standard assumptions, does not require a trusted setup phase, achieves sub-quadratic total computation and communication cost, protects each person's currency against abort attacks, and scales to support large scale mixing.

Monero is a Cryptonote-style protocol that uses ring signatures [27, 51] to obscure the sender of a transaction within a set of potential senders, plus homomorphic commitments and range proofs to hide transaction values [50]. If n Monero users wish to transact anonymously today, each of them must perform computation, verification, and communication that is linear in the size of the anonymity set, so the total costs are $O(n^2)$. As a result, Monero is only efficient for small scale mixes; earlier versions of Monero allowed ring sizes as small as 4, which led to sender deducibility attacks [8, 46]. It is worth mentioning that a formalized security analysis is included in [37], but is not compatible with the current version of the Monero protocol.

Zcash is based on succinct zero-knowledge arguments (zk-SNARKs) [28] and can offer private mixing at scale, but the underlying zk-SNARKs require trusted setup as well as strong (non-falsifiable) cryptographic assumptions[2]. The total computational cost of generating these zk-SNARKs for all n parties will grow as $O(n \log(t) \log(\log(t))$, where t denotes the (monotonically increasing) number of Zcash transactions ever made. QuisQuis [25] attempts to solve the problem of UTXO size, but requires similar computation costs to Monero: $O(n^2)$ for n users transacting anonymously.

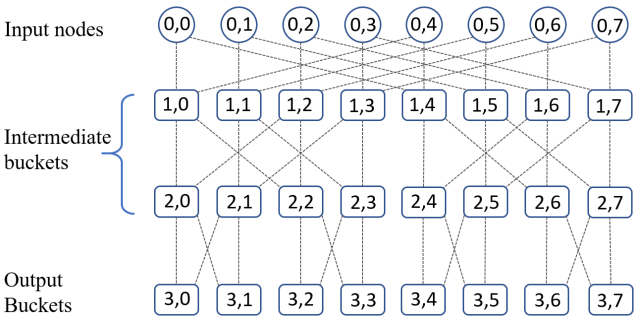

Fig. 1. 2-ary 8-size Butterfly Network, with nodes in $1, \ldots, d$ layers treated as buckets

2 Building Blocks

2.1 Butterfly Network Topology

A butterfly network [29] is a graph structure that facilitates routing. It provides a path from each of the n input nodes to the n output nodes while ensuring that both fan-in k and depth $d = \log_k(n)$ are sub-linear in n.

Definition 2.1 (k-ary n-size Butterfly Network)**.** *A k-ary n-size* butterfly network *consists of n nodes in each of the $d+1$ layers, where $d = \log_k n$. Let $\langle i, j \rangle$ denote the jth node in the ith layer with both indices starting with 0. A directed edge links two nodes $\langle i-1, j_1 \rangle$ and $\langle i, j_2 \rangle$ if and only if $j_1 = j_2$ or the base-k representation of j_1 and j_2 differ only in the ith most significant position. A* binary butterfly network *is a butterfly network with $k = 2$ and $d = \lg(n)$.*

Fig. 1 shows a binary butterfly network of size 8. For example, $\langle 1, 2 \rangle$ is connected to both $\langle 2, 0 \rangle$ and $\langle 2, 2 \rangle$ because the base-2 representations $0_2 = 000$ and $2_2 = 010$ only differ in the second most significant bit. Notice that in the binary butterfly network, a packet travels left if the ith significant bit of b's binary representation is 0, and travels right otherwise.

In this work, we refer to layer 0 nodes as *input nodes*, layer d nodes as *output buckets*, and all nodes in intermediate layers as *intermediate buckets*.

A butterfly network contains a unique path from every input node $\langle 0, a \rangle$ to every output bucket $\langle d, b \rangle$. The path can be calculated incrementally: at layer $i-1$, the packet travels through the rth leftmost outgoing edge (starting from 0), where r equals to the ith leftmost position's value of b's base-k representation. In other words, at layer i, the path proceeds through bucket $\langle i, c \rangle$ where c shares its i most significant symbols with b and the remaining symbols with a (for base k).

A *congestion* happens when more than one path pass through the same intermediate bucket. We will show later in Section 6.1 that a congestion with size $\Omega(\log n)$ only happens with negligible probability in our construction, unless a malicious adversary is conducting a (detectable) denial of service attack.

Our usage of the butterfly network draws inspiration from switching networks [1, 10]. A *novelty* of our construction is the use of buckets so that multiple transactions can be present at the same position in an intermediate layer.

2.2 (ϵ, δ)-Indistinguishable Mixing

We start by formally providing the definitions of neighboring permutations:

Definition 2.2. *Permutations π and π' are said to be* neighbors *if and only if they have a swap distance of 1, where the* swap distance *is the minimum number of times that pairs of elements of π must be swapped in order to produce π'.*

Intuitively, defining neighboring permutations through swap distance ensures that even an adversary who knows all the mappings between the mixing inputs and outputs, except any two pairs, still cannot gain a significant advantage in determining the mapping of these two pairs, even when given the leakage of the mixing mechanism.

2 Recent works propose zk-SNARKs with different types of setup and crypto assumptions, though all are still non-standard and the prover computation cost remains high [6, 12, 39].

We give the formal definition for (ϵ, δ)-indistinguishable mixing, inspired by differential privacy [23, 24]:

Definition 2.3 ((ϵ, δ)-Indistinguishable Mixing)**.** *Let $\mathcal{L}$ be a randomized algorithm that receives a permutation as input. We say that $\mathcal{L}$ is (ϵ, δ)-*indistinguishable *if for any two neighboring input permutations π and π' and for all $\mathcal{T} \subseteq Range(\mathcal{L})$:*

$$\Pr[\mathcal{L}(\pi) \in \mathcal{T}] \leq e^{\epsilon} \Pr[\mathcal{L}(\pi') \in \mathcal{T}] + \delta,$$

in which $Range(\mathcal{L})$ denotes the universe of all possible output of algorithm $\mathcal{L}$.

Throughout the text, we also refer to ϵ and δ as privacy parameters. Additionally, we say the mixing protocol is an (ϵ, δ)-indistinguishable mixing mechanism, if $\mathcal{L}$ is (ϵ, δ)-indistinguishable and it captures the whole view of any observer and the adversary of the mixing protocol.

Analogous to the definition of differential privacy for databases, our definition captures arbitrary background knowledge of the adversary. Just as in the classic definition, this follows by providing the adversary the ability to choose the pair of neighboring permutations. The fact that the inputs are constrained to being permutations can be viewed as a lower bound on this background knowledge: the adversary knows, for free, that all user inputs are unique (i.e. they constitute a permutation). The adversary's ability to choose the neighboring permutations captures further, arbitrary background knowledge.

Additionally, while constraining the user inputs to a permutation imposes a notion of "neighboring" that results from swapping two inputs, rather than dropping or modifying a single input value, it has been observed in the context of differential privacy for databases that such modifications to the definition of neighboring inputs do not have substantive impacts on the definition ([58]). Defining neighboring datasets by removing a single input value is appealing, because it captures the intuition that any individual's participation cannot be detected. Under our definition, the number of participants is fixed, so this intuition no longer holds. However, even if it is known that a particular user participated, privacy is guaranteed because their input could have been *any* arbitrary value from the domain.

Formally, we can also apply the same semantically flavored interpretation of (ϵ, δ)-indistinguishable mixing as in [32], modifying it slightly for our scenario: regardless of external knowledge, an adversary with access to our mixing protocol's leakage (which we characterize as the histogram of intermediate bucket weights) draws the same conclusions, regardless of whether my coin was swapped with that of another party.

Similar to differential privacy, the following lemma and theorem hold for Definition 2.3 and their proofs trivially follow the same arguments as used in differential privacy [24].

Lemma 2.4 (Post-Processing)**.** *Let $\mathcal{L}$ be a randomized algorithm that is (ϵ, δ)-indistinguishable. Let g be an arbitrary deterministic or randomized function. Then $g \circ \mathcal{L}$, the composition of functions g and $\mathcal{L}$, is also (ϵ, δ)-indistinguishable.*

Theorem 2.5 (Composition)**.** *Let $\mathcal{L}_i$ be a (ϵ_i, δ_i)-indistinguishable algorithm for $i \in [k]$. Define $\mathcal{L}_{[k]}(\pi) = (\mathcal{L}_1(\pi), \ldots, \mathcal{L}_k(\pi))$. Then $\mathcal{L}_{[k]}(\pi)$ is $(\sum_{i=1}^{k} \epsilon_i, \sum_{i=1}^{k} \delta_i)$-indistinguishable.*

Let Π denote the whole set of permutations for n elements. We consider a deterministic function $f : \Pi \to \mathbb{N}^m$ that takes a permutation and returns m natural numbers. As in standard differential privacy, we define the sensitivity of f as:

Definition 2.6 (ℓ_1-Sensitivity)**.** *The ℓ_1-sensitivity of f is:*

$$\Delta f = \max_{\forall \pi, \pi' \in \Pi} ||f(\pi) - f(\pi')||_1$$

where π, π' denote two neighboring permutations.

Δf captures the magnitude of f's output change between worst-case neighboring inputs.

In this work, we sample noise from the negative binomial distribution which models the number of successes in a series of independent and identically distributed (i.i.d) Bernoulli trials until r failures occur. Formally, we have:

Definition 2.7 (Negative Binomial Distribution)**.** *The negative binomial distribution is a non-negative discrete probability distribution with probability mass function:*

$$NB_{r,p}(x) = \binom{x + r - 1}{r - 1} \cdot (1 - p)^r \cdot p^x$$

where r is any positive integer and $p \in [0, 1]$. Additionally, its cumulative distribution function is:

$$F_{NB}(x; r, p) = \sum_{i=0}^{x} NB_{r,p}(x) = I_{1-p}(r, x + 1)$$

where $I.(\cdot, \cdot)$ is the regularized incomplete beta function.

We use $\mathrm{NB}(r, p)$ to refer to the negative binomial distribution itself. The negative binomial distribution satisfies an additive property: $\mathrm{NB}(r_1, p) + \mathrm{NB}(r_2, p) = \mathrm{NB}(r_1 + r_2, p)$.

The number of failures r in the negative binomial distribution can also be generalized to any positive real value, and the generalized distribution is often referred to as the Polya distribution:

Definition 2.8 (Polya Distribution)**.**

$$Pol_{r,p}(x) = \frac{\Gamma(x+r)}{x!\Gamma(r)} \cdot (1-p)^r \cdot p^x$$

where r is any positive real value and $p \in [0, 1]$, and $\Gamma(\cdot)$ is the Gamma function.

Notice that in the special case that r is an integer, $\frac{\Gamma(x+r)}{x!\Gamma(r)} = \frac{(x+r-1)!}{x!(r-1)!} = \binom{x+r-1}{x}$. Similarly, the Polya distribution also satisfies an additive property, let $\mathrm{Pol}(r_1, p)$ and $\mathrm{Pol}(r_2, p)$ be any two Polya distributions, then: $\mathrm{Pol}(r_1, p) + \mathrm{Pol}(r_2, p) = \mathrm{Pol}(r_1 + r_2, p)$.

The additive property of negative binomial distribution and Polya distribution allow all n participants to collectively sample noise from a negative binomial distribution $\mathrm{NB}(r, p)$ without cooperation. Each participant can simply sample noise from a Polya distribution $\mathrm{Pol}(r/n, p)$ and later sum up their noises.

2.3 UC Secure Computation

Cryptographically secure computation is formalized by demonstrating that an ideal world execution is indistinguishable from the real world protocol. At a high level, the intuition behind security in the universal composability (UC) framework [13] is that any adversary $\mathcal{A}$ attacking a protocol π should learn no more information than could have been obtained via the use of a simulator $\mathcal{S}$ that makes calls to the ideal functionality $\mathcal{F}$ instead. The UC formalism includes an environment $\mathcal{E}$ that represents "the rest of the world"; it provides inputs to all parties and also attempts to distinguish between the real and ideal executions. The UC framework includes a powerful composition theorem that supports modularity [13]; composition continues to hold in a "hybrid" setting in which both the real protocol and ideal functionality may call a common sub-routine. In this work, we require the strengthened model of UC with global setup [15] that has recently been instantiated as the default behavior within the UC framework [13].

Definition 2.9 (Universal Composability)**.** *Let $\mathcal{F}$ be an ideal functionality, and let Π be an interactive protocol for computing $\mathcal{F}$ among a set of parties P while making calls to an ideal functionality $\mathcal{G}$. Protocol Π is said to* UC realize *$\mathcal{F}$ in the $\mathcal{G}$-*hybrid model *if for every non-uniform PPT malicious adversary $\mathcal{A}$, there exists a non-uniform PPT adversary $\mathcal{S}$ in the ideal model such that for all non-uniform PPT environments $\mathcal{E}$ and all valid inputs $x_1, \ldots, x_p$, the following two distributions are computationally indistinguishable in κ:*

$$\mathsf{EXEC}_{\Pi^{\mathcal{G}},\mathcal{A},\mathcal{E}} \stackrel{c}{\equiv} \mathsf{EXEC}_{\mathcal{F}^{\mathcal{G}},\mathcal{S},\mathcal{E}}$$

The polynomial running times of the various interactive Turing machines are related; we refer to [13] for details.

2.4 UC Functionalities

We use three UC functionalities as building blocks. The first two have been extensively examined in prior work: a public bulletin board $\mathcal{G}_{\mathsf{bb}}$ [14, 16, 17] and a globally-accessible clock $\mathcal{G}_{\mathsf{clock}}$ [3, 16, 33, 59]. We include these functionalities in Appendix A for completeness.

Anonymous Transaction Functionality. In Figure 2, we provide a UC functionality $\mathcal{G}_{\mathsf{AnonTrans}}$ for partially-anonymous transactions on top of a ledger. This functionality utilizes the simple bulletin board functionality $\mathcal{G}_{\mathsf{bb}}$ and is meant to be a simplified version of existing UC models of blockchain ledgers [4, 35] that take a transaction amount from a sender and deliver the specified value to the receiver by updating the recorded balances. In order to focus on details that are more relevant to this work, our functionality intentionally lacks some features, such as clustering transactions into blocks, that are more realistic reflections of cryptocurrencies, but that are irrelevant to our constructions. At the same time, our transaction functionality more closely resembles that of an account based system as opposed to a transaction based one, i.e. the functionality is responsible for checking user balances before transactions.

On the other hand, our UC anonymous transaction functionality augments prior work that was mostly modeling linkable, Bitcoin style, ledgers in two ways: first, we provide confidential transactions, and second, we permit equivocation of the sender of any transaction within a small anonymity set S. This set must be specified by the (real) sender at the time of the transaction, and it will be publicly revealed to all parties. We caution that a small equivocation set may be private in a standalone setting but need not retain this privacy when composed with other transactions [46].

$\mathcal{G}_{\mathsf{AnonTrans}}[\mathcal{G}_{\mathsf{bb}}]$: **Anonymous Transaction functionality**

Registration: User i submits a pair of an address addr and a value v. If addr is not yet assigned to any other users, the functionality assigns addr to i and stores a record $(\mathsf{addr}, \mathsf{v})$ indexed with i.

Transfer: User i, or $\mathcal{F}^{T}_{\mathsf{AnonMix}}$ on its behalf, submits a transfer request with parameters $(S, j, \mathsf{addr}_r, \mathsf{v}_r)$ where S denotes a set of addresses $S = \{\mathsf{addr}_1, ..., \mathsf{addr}_\ell\}$ and j is an index $j \in \{1, \ldots, \ell\}$. The functionality first checks if addr_r is already registered. Then, it also checks if $\mathsf{addr}_1, ..., \mathsf{addr}_\ell$ are registered and whether addr_j is indexed with the user index i. The functionality checks if $\mathsf{v}_j \geq \mathsf{v}_r$ (i.e. the sending address carries enough funds). If any of the checks fail, it returns an "Error" message.
Then the functionality replaces $(\mathsf{addr}_r, \mathsf{v}'_r)$ with $(\mathsf{addr}_r, \mathsf{v}'_r + \mathsf{v}_r)$ and $(\mathsf{addr}_j, \mathsf{v}_j)$ with $(\mathsf{addr}_j, \mathsf{v}_j - \mathsf{v}_r)$ for addr_j that belongs to user i. Additionally, the functionality adds (S, addr_r) to $\mathcal{G}_{\mathsf{bb}}$. Else, it ignores the request.

Balance: User i submits a balance request with address addr. If user i has a registered record $(\mathsf{addr}_s, \mathsf{v}_s)$, then a delayed output is sent to user i with the private value v_s; otherwise, it returns "Error".

Output: The functionality sends (S, addr_r) to all the participants.

Fig. 2. Anonymous Transaction functionality

At the same time, a large equivocation set provides strong security but may be prohibitively expensive in many cryptocurrencies; in Monero for instance, transaction size and computation scale linearly with the size of the anonymity set. This work provides a way to bootstrap the privacy benefits of a large anonymity set, in a provable manner, while only calling the functionality $\mathcal{G}_{\mathsf{AnonTrans}}$ using small anonymity sets.

3 Definitions

3.1 A UC Definition for Secure and (ϵ, δ)-Indistinguishable Mixing

In this section, we define anonymous mixing in the UC setting by presenting our functionality $\mathcal{F}^{T}_{\mathsf{AnonMix}}$ in Figure 3. Our functionality will capture our mixing protocol, which uses a butterfly network topology that obscures the true permutation π which maps the source addresses to the receiver addresses.

In a high level, our $\mathcal{F}^{T}_{\mathsf{AnonMix}}$ functionality receives input and output addresses from participants, up until some fixed time T. When the mixing begins at time T, the functionality will follow steps that will resemble a mixing protocol utilizing the $\mathcal{G}_{\mathsf{AnonTrans}}$ functionality. It will additionally provide the leakage function $\mathcal{L}$ to the adversary as defined by the input and output addresses. The adversary will return a set of transactions for a k-ary butterfly network of size n including transactions made to intermediate addresses for each of the $\log_k n$ layers. The functionality will construct the transactions through calls to the $\mathcal{G}_{\mathsf{AnonTrans}}$ functionality and notify the adversary on completion.

Assumptions. The assumptions made by our ideal functionality are as follows:

- *Mixing Value:* We assume that all honest input addresses carry the same value v, i.e. all parties are mixing equal funds. Also each participant can specify multiple input and output addresses.
- *Sufficient Funds:* Given that our protocol does not have an upper bound on the number of parties that can mix in any given session, and that it will not begin prior to time T, there is no reason to check whether an input address carries a sufficient balance prior to T. If a user does not carry the fixed mixing amount v, it will be dropped by the mixing protocol.
- *Corruption Model:* The adversary can register any number of corrupted parties during registration.

Discussion. We note that an extension of our UC functionality in the multi-value mixing case is not trivial. The difficulty stems by the fact that a simulator needs to know the transaction values of each path of the butterfly network in order to be able to simulate them.

3.2 A Game-Based Definition for (ϵ, δ)-Indistinguishable Mixing

We also provide a game-based indistinguishability (GB-indistinguishability) definition to more clearly showcase the adversary's advantage in distinguishing between two neighbor mixing addresses permutations.

Definition 3.1. *Let Π_n be a mixing protocol involving n parties. Consider the following experiment Exp_{Mix} between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:*

$\mathcal{F}^{T}_{\mathsf{AnonMix}}[\mathcal{G}_{\mathsf{AnonTrans}}, \mathcal{G}_{\mathsf{bb}}, \mathcal{G}_{\mathsf{clock}}](v, k, \mathcal{L})$:
Anonymous Mixing Functionality

The AnonMix functionality interacts with the adversary, all parties, and three sub-functionalities. It is parameterized by some mixing value v, the fan-in parameter for the butterfly network k, and a leakage distribution $\mathcal{L}$. Mixing occurs at a specific, pre-defined absolute time T. A list L is initialized to $\emptyset$.

Register: Upon receiving $(\text{REGISTER}, \mathsf{addr}_i^{IN}, \mathsf{addr}_i^{OUT})$ from party P_i, first run *Check* and only continue if *Mix* is not invoked. Add $(P_i, \mathsf{addr}_i^{IN}, \mathsf{addr}_i^{OUT})$ to L, the list of participants in the current mixing session. The functionality posts addr_i^{IN} to $\mathcal{G}_{\mathsf{bb}}$.

Ping: This method has no parameters and can be called by anyone (even the environment). It invokes *Check* and returns execution to its caller.

Check: Send GETTIME to $\mathcal{G}_{\mathsf{clock}}$ and receive τ from $\mathcal{G}_{\mathsf{clock}}$. If $\tau > T$ then run the *Mix* process. Otherwise, return to the calling method.

Mix:
- The functionality randomly assigns an output bucket to each output address in L. Let $\vec{w}$ denote the vector of bucket weights, i.e., the number of output addresses contained in each bucket.
- The functionality determines π based on the sets $\mathsf{addr}^{IN}[]$, $\mathsf{addr}^{OUT}[]$, and the output bucket assignments. It computes $\mathcal{L}(\pi)$, and provides $(\mathcal{L}(\pi), \vec{w})$ to the adversary.
- The adversary returns a set of transactions for a k-ary butterfly network of size n, specifying for each transaction the values $(S, j, \mathsf{addr}_r, v)$ and which layer of the network it corresponds to.
- The functionality calls $\mathcal{G}_{\mathsf{AnonTrans}}()$ for all requested transactions.
- Notify Adversary that mixing was done.

Fig. 3. Anonymous Mixing Functionality

1. *The challenger sends* $S^{IN} = \{\mathsf{addr}_1^{IN}, \ldots, \mathsf{addr}_{n'}^{IN}\}$ *and* $S^{OUT} = \{\mathsf{addr}_1^{OUT}, \ldots, \mathsf{addr}_{n'}^{OUT}\}$ *to* $\mathcal{A}$. *The sets denote the input and output addresses of honest parties (respectively), and each set is of size* n'.
2. $\mathcal{A}$ *sends to* $\mathcal{C}$ *two chosen sets of input and output addresses* $\hat{S}^{IN}, \hat{S}^{OUT}$ *on behalf of the malicious parties, along with their secret keys (in order to allow* $\mathcal{C}$ *to run the mixing protocol itself). Additionally, the adversary might opt to corrupt some of the addresses sent by the challenger, in which case he is given their secret keys and these addresses are moved from set* S *to corresponding* $\hat{S}$. *Let* t *be the total number of adversarial parties and* n *the total number of parties.* $\mathcal{A}$ *selects two neighboring permutations* π_0, π_1 *(cf. Definition 2.2) over the unions of the challenger and the adversary's sets, i.e.,* $\pi_0, \pi_1 : S^{IN} \cup \hat{S}^{IN} \to S^{OUT} \cup \hat{S}^{OUT}$. *The permutations must agree on the adversarial addresses* $\pi_0|_{\hat{S}^{IN}} = \pi_1|_{\hat{S}^{IN}}$.
3. *The challenger flips a coin* $b \in \{0, 1\}$ *and runs the mixing protocol* Π_n *on input* π_b. *We define the output of the protocol as Out.*
4. *The adversary is given Out and guesses* b'. *He wins if* $b = b'$.

We say that Π_n *is a* (z, t)*-GB-indistinguishable mixing protocol for any PPT adversary who has corrupted up to* t *parties if the advantage of winning the game defined in* Exp_{Mix} *is bounded by some value* z *parameterized by our choice of privacy parameters.*

4 Construction

We construct an anonymous mixing protocol $\Pi^{\vec{s},\vec{v},\vec{r}}_{T,n,d,c}$ that allows n participants, where each participant p owns a sender address s_p and a receiver address r_p, to anonymously transfer a value v_p from s_p to r_p. To hide the mapping between sender and receiver addresses, we employ d rounds of mixing in which the participants mix with a subset of all participants in the round (these subsets are determined by the interconnections between two consecutive layers of buckets on a butterfly network). Each participant creates ephemeral addresses to hold the value in each round, except the last round where transactions are made to the receiver addresses.

In order to argue that the view of the protocol is (ϵ, δ)-indistinguishable, participants contribute to several "*noisy addresses*" that contain zero value via "noisy transactions" that carry zero value. We refer to the addresses that actually carry value as described in the above paragraph as *real addresses*. In particular, we rely on confidential transactions to ensure that a noisy transaction is individually indistinguishable from a real one, and we sample the number of noisy addresses from negative binomial distribution.

To determine the mixing participants, our protocol relies on using a blockchain as a public bulletin board where parties publish all necessary addresses in a layer by layer fashion. When a party wishes to have a transaction output mixed, it includes a relevant "to be mixed" flag. In the first phase of the mixing protocol, the sender addresses to be mixed will all be output transaction

addresses marked as "to be mixed" within some specific time period T (i.e. last 100 blocks). Similarly, any subsequent layers of transactions can rely on transaction outputs posted on the blockchain from the previous layer to form the subset of mixing parties in the current layer. We note that, similar to other privacy preserving cryptocurrencies, we require anonymous communication when users post their transactions on the bulletin board.

4.1 Building the Butterfly Network

Given n parties and an arbitrary choice of fan-in k, the transaction graph in our protocol follows a k-ary n-size butterfly network with depth $d = \log_k(n)$ in which the input layer nodes correspond to sender addresses and the output layer nodes correspond to receiver addresses. Each participant owns one pair of such nodes. We remark that most of our construction applies generically to any network topology that ensures unique paths from source to receiver addresses, however, we restrict our attention to the butterfly network in this work.

The sender addresses in the input layer are ordered by simply sorting them, where each address occupies one position in the input layer. The positions for receiver addresses in the output layer are independently chosen by each participant. For every address except the sender address, the participant includes its position in current layer in the transaction posted to blockchain.

In our protocol, each participant is solely responsible for transferring her own coins through the entire (unique) path in the butterfly network from her sender address to her receiver address. As a result, she must own an address for each layer along that transaction path. Since paths by different participants can cross through the same position in the intermediate $(1, \ldots, d-1)$ and output layers, our protocol allows for multiple addresses to reside in these positions and refer each position as a bucket.

We use s_p and r_p respectively to denote the sender address and receiver address of a participant p. We use v_p to denote the value that p wants to transfer from s_p to r_p. More generally, we denote each address by $\mathsf{a}_{p,l}^{\langle i,j \rangle}$. Here, the superscript $\langle i, j \rangle$ denotes that the address is located within the bucket in layer i at the jth position from the left. The left subscript indicates that this address is owned by participant p, and we use the counter $l \in \mathbb{N}$ as a unique index to distinguish between all addresses owned by participant p within bucket $\mathsf{a}_{p,l}^{\langle i,j \rangle}$. By convention, we reserve $l = 0$ to denote the one real address (if any) that p places in bucket $\langle i, j \rangle$, and we index noisy addresses (if any) with positive integers. We also use array notation to refer to all addresses meeting certain characteristics. Specifically, the array $\mathsf{a}_p[]$ contains all addresses made by participant p (in the whole network), and $\mathsf{a}[]$ contains all addresses for all participants while $\mathsf{s}[]$ contains all sender addresses.

4.2 Generating and Sending Addresses

We describe our full mixing protocol $\Pi_{T,n,d,c}^{\vec{s},\vec{v},\vec{r}}$ in Figure 4. The protocol involves n parties but given that it is symmetric, we opt to describe the protocol from the perspective of a single party p. Throughout the protocol, we let $\leftarrow\!\!\$$ refer to sampling value from certain distributions, or sampling a key from some key generation algorithm. At a high level, the protocol can be divided into an *offline* and an *online* phase.

1. Offline Phase. Each party posts a transaction indicating its intent to participate in a mix. In lines 2-7, each participant retrieves the collection of sender addresses from the ledger and sorts them to determine their starting position, source. Each participant then randomly samples an output layer position, target, and locally runs a sub-routine $\mathsf{Path}(\mathsf{source}, \mathsf{target})$ to determine her unique path from the sender address to the receiver address along the network. In lines 9-12, for each bucket on the participant's path through the intermediate layers, she creates a fresh "real address" which will be used to pass non-zero transactions from her sender address to the intended receiver address. In lines 13-17, each participant independently generates a certain number of noisy addresses, which could be zero, for each bucket in the intermediate layers.

In this phase, all real and noisy addresses except the sender address are freshly created by the participant and kept locally.

2. Online Phase. Next, the participants use the ledger in order to synchronize each layer of transactions. Depending on the size of n and the specifics of the underlying ledger (i.e. block/transaction size), a specific number of ledger blocks will be devoted for each mixing layer; only transactions posted within these blocks contribute toward this layer. In each round of these transactions, every participant is responsible to create transactions paid to all addresses she owned, real or noisy, in the current/lower layer.

Fig. 4. Protocol $\Pi^{\vec{s},\vec{v},\vec{r}}_{T,n,d,c}(s[])$. The protocol is symmetric, so we show the protocol from the view of a single participant p who sends v_p value from sender address s_p to receiver address r_p.

1 : $\mathcal{G}_{\mathsf{bb}}.\mathsf{Write}(\mathsf{s}_p, \mathrm{Mix}, T)$ // Post s_p to mix at time T

.....Offline Phase.....

2 : $\mathcal{G}_{\mathsf{bb}}.\mathsf{Read}(\mathsf{s}[])$ // Read all addresses posted at time T

3 : $\mathsf{Sort}(\mathsf{s}[])$

4 : $\mathsf{source} := \mathsf{Find}(\mathsf{s}_p, \mathsf{s}[])$

// Return location index of p's sender address.

5 : $\mathsf{target} := \leftarrow\!\!\$\, \mathsf{Uniform}(n)$

// Sample location index of p's receiver address.

6 : $\mathsf{a}_{p,0}^{\langle 0,\mathsf{source}\rangle} := \mathsf{s}_p$

7 : $\mathsf{a}_{p,0}^{\langle d,\mathsf{target}\rangle} := \mathsf{r}_p$

8 : $\mathsf{a}_p[].\mathsf{Append}(\mathsf{a}_{p,0}^{\langle 0,\mathsf{source}\rangle}, \mathsf{a}_{p,0}^{\langle d,\mathsf{target}\rangle})$

// Maintain an array of p's own addresses.

9 : $\mathsf{pathbuckets}[] := \mathsf{Path}(\mathsf{source}, \mathsf{target})$

// Find path from sender address to receiver address

10 : **for** $\mathsf{bucket}_{\langle i,j\rangle}$ **in** $\mathsf{pathbuckets}[]$ **do**

// Generate real addresses for buckets along the path

11 : $\mathsf{a}_{p,0}^{\langle i,j\rangle} \leftarrow\!\!\$\, \mathsf{AddressGen}(1^\kappa)$

12 : $\mathsf{a}_p[].\mathsf{Append}(\mathsf{a}_{p,0}^{\langle i,j\rangle})$

13 : **for** $\mathsf{bucket}_{\langle i,j\rangle}$ **in** $\mathsf{allbuckets}[]$ **do**

// Iterate for every bucket in the intermediate network

14 : $x \leftarrow\!\!\$\, \mathsf{Noise}(n, \epsilon, \delta)$

15 : **for** $l = 1..x$ **do** // Generate x noisy addresses

16 : $\mathsf{a}_{p,l}^{\langle i,j\rangle} \leftarrow\!\!\$\, \mathsf{AddressGen}(1^\kappa)$

17 : $\mathsf{a}_p[].\mathsf{Append}(\mathsf{a}_{p,l}^{\langle i,j\rangle})$

.....Online Phase.....

18 : **for** $i = 1, 2, \ldots, d$ **do**

// Synchronize each layer of transactions among all

19 : **for** $\mathsf{a}_{p,l}^{\langle i,j\rangle}$ **in** $\mathsf{a}_p[]$ **do**

// Iterate through every p's address in the current layer

20 : **if** $i == 1$

21 : $\mathsf{inputs}[] := \mathsf{Parent}(\langle i, j\rangle, s[])$

// Read parent addresses from the sender addresses array

22 : **else**

23 : $\mathsf{inputs}[] := \mathsf{ReadParent}(\langle i, j\rangle, \mathcal{G}_{\mathsf{bb}}.\mathit{Read})$

// Read parent addresses from the ledger

24 : $\mathsf{output} := \mathsf{a}_{p,l}^{\langle i,j\rangle}$

25 : **if** $l == 0$ **then** // Create real transaction

26 : $\mathcal{G}_{\mathsf{AnonTrans}}.\mathsf{Transfer}(\mathsf{inputs}[], \mathsf{output}, v_p)$

27 : **else** // Create fake transaction

28 : $\mathcal{G}_{\mathsf{AnonTrans}}.\mathsf{Transfer}(\mathsf{inputs}[], \mathsf{output}, 0)$

In lines 20-23, each participant determines the current set of input addresses from either the sorted sender addresses array or the ledger, depending on whether this is a first round transaction or not. For the latter, each output address in the previous layer must also include its bucket location on the ledger. In lines 24-28, for a real transaction paid to a real output address, the participant disguises the real input address among all other addresses from the input set. Notice that since this transaction is along the path that a real value follows, there must exist a real input address owned by the same participant in one of the parent buckets on previous layer. On the other hand, if it is a noisy transaction, there might not exist a parent input address owned by the same participant. Here, we simply assume that a participant can create a transaction with zero value without owning any address in the input set. Later, we show a slightly different construction that remove this assumption.

Sub-Routines. The full protocol $\Pi^{\vec{s},\vec{v},\vec{r}}_{T,n,d,c}$ in Figure 4 uses the following sub-routines. For some we have UC functionalities described in Section 2.4 and the others are straightforward enough that we omit detailed descriptions for brevity.

- $\mathcal{G}_{\mathsf{AnonTrans}}.\mathsf{Transfer}(\mathsf{inputs}[], \mathsf{output}, v)$: Transfers v coins to the output address from one of the addresses in $\mathsf{inputs}[]$, as specified by the partially-anonymous transaction ledger functionality defined in Figure 2.
- $\mathsf{AddressGen}(1^\kappa)$: Generates an ephemeral address. Notice that both a real address and fake address are generated in the same way.
- $\mathsf{Find}(arr[], value)$: Return the index of the value in the array.
- $\mathsf{Path}(\mathsf{source}, \mathsf{target})$: Return an array of buckets (denoted by $\langle i, j\rangle$) along the unique path from *source* to *target*.
- $\mathsf{Sort}(arr[])$: Sort the array in increasing order. We use this function to sort the broadcasted array of participants' addresses. If each address is generated randomly, then the sort itself can be viewed as random permutation.
- $\mathsf{Noise}(n, \epsilon, \delta)$: Sample noise that satisfies the (ϵ, δ)-indistinguishable guarantee. Since the total noise is contributed by all n participants, the amount of individual noise required is also a function of n. In particular, to ensure the total noise follows a negative binomial distribution $\mathrm{NB}(r, p)$, each participant samples noise independently from a Polya distribution $\mathrm{Pol}(r/n, p)$.

- $\mathsf{Parent}(\langle i, j\rangle, s[])$: Return the set of addresses resided in the parent buckets of $\langle i, j\rangle$ from all sender addresses $s[]$ where i is the layer number and j is the bucket location within the current layer. The parent relationship is defined by the network topology.
- $\mathsf{ReadParent}(\langle i, j\rangle, \mathcal{G}_{\mathsf{bb}}.Read)$: Return the set of addresses from the parent buckets of $\langle i, j\rangle$ from $\mathcal{G}_{\mathsf{bb}}$.

Communication Cost. As the participants rely on published transactions on blockchain to establish the network construction, it suffices to only consider the on-chain communication cost. (We store on-chain all information that is needed for future transaction verification i.e., the sequence of the transactions). On-chain communication happens in Steps 26 and 28 and consists of an expected total of $n(1+\lambda) \times d$ messages for all participants. Here λ is the average amount of noisy addresses per bucket. Concretely messages are transactions with one output address and their size depends on the input set size and specific instantiation (discussed in Appendix B). As mentioned earlier, on-chain communication requires anonymous communication channels (i.e. Tor or Atom without the need of a bulletin board).

Handling Address Overflow. Having many addresses in a parent bucket (either through malicious behavior or excessive noisy addresses from honest sampling) forces the child bucket/node to create anonymous transactions of larger sizes, and increases the overall computation. We formally describe such a scenario in Section 6, and provide a formal bound. As a countermeasure, the participants can set an upper bound on the number of total address allowed in one bucket, and agree to abort the protocol (or simply halt the transactions that connect to this bucket) when any bucket exceeds this bound. This prevents inefficient executions, but could lead to denial of service attacks where an adversary maliciously adds noise that exceed the bound. In Section 7 we explain how to avoid such attacks.

User Aborts. Our discussion accounts for both honest user aborts and malicious aborts during the protocol execution. As opposed to previous work on cryptocurrency mixing [53, 54], our protocol makes no attempt to detect and handle user aborts. This is mainly because an aborted user does not hinder the normal transaction flow for the rest of the users. In terms of privacy loss, we assume the worst case scenario where all aborts are caused by t malicious parties. Aborts can happen in any phase of the protocol, and without loss of generality we assume that in the end all malicious parties publicly reveal all their real and noisy addresses. The real address paths of malicious parties will be identified, and cause the reduction of anonymity set to $n-t$. The noisy addresses disclosed will reduce the effective noise magnitude to $(n-t)/n$ of the original for each bucket. To achieve the target anonymity set size and privacy parameter, both issues can be easily addressed by setting a larger required number of mixing parties $n' = n + t$.

Instantiation of Our Protocol Using Ring Signatures. For compatibility with Monero, in Appendix B, we provide an instantiation of our protocol using ring signatures over confidential transactions (CTs).

5 Privacy and Security Analysis

5.1 Privacy Analysis

We now rigorously prove that the leakage in our mixing protocol $\Pi^{\vec{s},\vec{v},\vec{r}}_{T,n,d,c}$ is (ϵ, δ)-indistinguishable. We build up to this proof via a series of lemmas that first analyze the negative binomial mechanism generally and then apply this analysis toward the noise distribution within $\Pi^{\vec{s},\vec{v},\vec{r}}_{T,n,d,c}$.

Lemma 5.1. *Let $f_1 : \Pi \to \mathbb{N}$ be a function with ℓ_1-sensitivity 1. Let $\mathcal{M}_1(\pi)$ be the mechanism that output the sum of $f_1(\pi)$ and some noisy value sampled from $NB(r, p)$ with $r \geq 1$. Then $\mathcal{M}_1$ is (ϵ, δ)-indistinguishable for $\epsilon \geq \ln(1/p)$ and $\delta = I_{1-p}(r, k+1) - e^{\epsilon} I_{1-p}(r, k)$, where $k = \left\lfloor \frac{p(r-1)}{e^{\epsilon}-p} \right\rfloor$.*

We defer the proof of this lemma to Appendix C.

Lemma 5.2. *Let $f_2 : \Pi \to \mathbb{N}^2$ be a function such that for any pair of neighboring inputs π and π' (Definition 2.2), with $f_2(\pi) = (x_1, x_2)$, $f_2(\pi')$ equals to either $(x_1 + 1, x_2 - 1)$ or $(x_1 - 1, x_2 + 1)$. Let $\mathcal{M}_2$ be a mechanism that perturbs the output of f_2 by independently adding $NB(r, p)$ noise to each element. Then, $\mathcal{M}_2$ is $(2\epsilon, \delta)$-indistinguishable with ϵ, δ defined as in Lemma 5.1.*

We defer the proof of this lemma to Appendix C.

Next, we consider the (randomized) leakage function $\mathcal{L} : \Pi \to \mathbb{N}^{n \times (d-1)}$ as a map from an input/output permutation to a matrix representing the intermediate bucket weights in the network. We informally state that $\mathcal{L}$ captures the whole view of the adversary (and leave the formal argument to Section 5.2) from our mixing protocol, so it suffices to prove $\mathcal{L}$ is indistinguishable to

show that our mixing protocol is an indistinguishable mixing mechanism. Note that in Fig. 3, the set of output weights are given. We first argue that the output weight itself does not leak any information regarding the permutation chosen, i.e., the output buckets weights are independent of the permutation (which is why we do not place any noisy addresses in the output layer). To see this, consider neighboring permutations, π_1 and π_2, that have swapped targets in the source/target pairs $(s_1, t_1), (s_2, t_2)$. For any fixed sequence of weights $\vec{w}$ in the output layer, its probability of occurrence is identical under both permutations, as the probability of swapping the locations of t_1 and t_2 such that they are consistent with π_2 is identical to the probability of placing them in the locations consistent π_1. As a result, our analysis of Lemma 5.3 and Theorem 5.4 are conditioned on any fixed sequence of weights in the output layer.

We begin with the following simple observation.

Lemma 5.3 (Swap Sensitivity)**.** *For any fixed sequence of output weights $\vec{w}$, in each layer of the intermediate buckets, a swap from permutation π to a neighboring permutation π' can result in one of the following two cases: (a) the number of real addresses in two buckets each increase by one and the number of addresses in another two buckets each decrease by one, or (b) the number of real addresses in each of this layer's buckets remain unchanged.*

Proof. This is an immediate consequence of the fact that the sender and receiver addresses in π or π' imply a unique path and one weight to each bucket it passes. □

Finally, the following theorem holds for any output layer bucket weights.

Theorem 5.4. *For any fixed sequence of output weights $\vec{w}$, the leakage function $\mathcal{L} : \Pi \to \mathbb{N}^{n\times(d-1)}$ is $(\bar{\epsilon}, \bar{\delta})$-indistinguishable, where $\bar{\epsilon} = 4(d-1)\epsilon$, $\bar{\delta} = 2(d-1)\delta$, and ϵ, δ are defined as in Lemma 5.1.*

Proof. According to Lemma 5.3, there are at most $2(d-1)$ pairs of buckets that increase and decrease by one across all $d-1$ intermediate layers. As our protocol essentially applies noise sampled from negative binomial distribution on all buckets, it can be decomposed into $2(d-1)$ parallel run of $\mathcal{M}_2$ on each pair of buckets. Each run of $\mathcal{M}_2$ is $(2\epsilon, \delta)$-indistinguishable, according to Lemma 5.2. Therefore, composition (Theorem 2.5) yields the desired result. □

5.2 Security Analysis

UC Security. Our protocol satisfies the anonymous mixing functionality $\mathcal{F}^{T}_{\mathsf{AnonMix}}$ defined in Section 3.1. In Appendix D, we describe a simulator that internally emulates an execution of the real protocol with black-box access to the real adversary.

Theorem 5.5. *Given the probabilistic, polynomial time function $\mathcal{L}$ defined in the previous section, the protocol described in Figure 4 UC-realizes $\mathcal{F}^{T}_{\mathsf{AnonMix}}$ in the ($\mathcal{G}_{\mathsf{AnonTrans}}$, $\mathcal{G}_{\mathsf{bb}}$, $\mathcal{G}_{\mathsf{clock}}$)-hybrid world with static corruption, with $\mathcal{L}$ leakage and $(\kappa, \bar{\epsilon}, \bar{\delta})$-privacy for $\bar{\epsilon}$ and $\bar{\delta}$ as specified in Theorem 5.4.*

We defer the proof of this theorem to Appendix D.

Game-Based Security.

Theorem 5.6. *Protocol $\Pi^{\vec{s},\vec{v},\vec{r}}_{T,n,d,c}$ is a $(\frac{\bar{\epsilon}}{2} + \bar{\delta}, t)$-GB-indistinguishable mixing protocol for any PPT adversary, under the condition that the noise magnitude in the mixing protocol is multiplied by a factor of $n/(n-t)$, where t is the number of corruption parties.*

The proof is in Appendix E. It relies on the post-processing property (Lemma 2.4).

6 Complexity of Our Construction

6.1 One Layer Butterfly Network

In this subsection, we discuss a practical setup of our protocol, using a $\sqrt{n}$-ary Butterfly network. In the rest of this section, we also simply refer to this as the "one-layer case," as there is only one intermediate layer.

Asymptotic Analysis. Under the assumption that the computational cost of a single anonymous transaction is linear in the size of the input set, we can calculate the overall computational and communication costs of the protocol by counting the number of edges connecting addresses in the butterfly network. We stress that for any address, there is an edge between it and every address in its parent buckets. In other words, the in-degree of an address is equal to the number of addresses in its parent buckets, not the number of its parent buckets.

Theorem 6.1. *The expected number of edges for one layer butterfly network is $O(n^{1.5})$. Additionally, the number of edges is upper bounded by $O(n^{1.5} \log n \log\log n)$ except for negligible probability.*

We defer the proof to Appendix F.

If an anonymous transaction is implemented using log-size ring signatures [37, 63], then the size of a single anonymous transaction is logarithmic in the size of the input set. In our one-layer case, the total size is dominated by the first round of transactions (larger number of transactions with smaller rings compared to the 2nd round). Let μ denote the expected number of noisy addresses in each bucket, the expected size of all ring signatures is: $(1+\mu)n \cdot \log(\sqrt{n}) \in O(n \log n)$, and the size is bounded by $O(n \log^2 n \log\log n)$ except with negligible probability.

Merged Buckets. The asymptotic analysis neglects the inefficiency caused by the noise in practise. Concretely, the noisy addresses end up constituting a majority of all addresses in the intermediate layer (in average, μ times the number of real addresses), which significantly increases the number of edges. Here we propose an optimized network topology based on the butterfly network that yields better computational and space costs in practice.

The essential idea is to "merge" some of the intermediate buckets together. The motivation here is to reduce the number of buckets that we need to add noise too, given that the noise magnitude for each bucket is invariant to n, and independent of the specific network topology use. However, merging arbitrary buckets could potentially increase the number of incoming edges and outgoing edges of each address in the bucket (Recall that we require addresses in the same bucket share the same set of parent addresses and child addresses). To avoid this, we choose buckets such that some of them already share their parent addresses or child addresses, so that the in-degree and out-degree increase sub-linearly with the number of merging buckets.

Concretely, we denote the intermediate buckets from left to right as $\mathsf{Bucket}_0, \ldots, \mathsf{Bucket}_{n-1}$. For any two buckets, Bucket_a and Bucket_b, their indices can be represented as $a = a_1\sqrt{n} + a_0$ and $b = b_1\sqrt{n} + b_0$. According to Definition 2.1, if $a_1 = b_1$, then Bucket_a and Bucket_b share the same set of output buckets in the butterfly network. Correspondingly, if $a_0 = b_0$, Bucket_a and Bucket_b share the same set of input nodes. The size of these sets are both $\sqrt{n}$. We merge the buckets μ-wise to form a total of $\frac{n}{\mu}$ merged buckets. We will show in Claim 6.3 that by choosing μ as the number of buckets to merge, the expected number of edges is minimized. For ease of analysis, we assume n is divisible by μ. Let $\mathsf{MBucket}_0, ..., \mathsf{MBucket}_{\frac{n}{\mu}-1}$ denote the $\frac{n}{\mu}$ merged buckets. For each $\mathsf{MBucket}_j$, with $j = j_1\sqrt{n/\mu} + j_0$, then $\mathsf{MBucket}_j$ is constructed from merging the following set of buckets: $\{\mathsf{Bucket}_a \mid j_1\sqrt{\mu} \le a_1 < (j_1+1)\sqrt{\mu}, j_0\sqrt{\mu} \le a_0 < (j_0+1)\sqrt{\mu}\}$. Through this construction, each $\mathsf{MBucket}_j$ connects to $\sqrt{\mu}$ input node sets and $\sqrt{\mu}$ output bucket sets, which translate to $\sqrt{\mu n}$ of input nodes and $\sqrt{\mu n}$ of output buckets.

We give the following claims regarding the practical efficiency of the merged bucket invariant and defer all their proofs to Appendix F.

Claim 6.2. *The expected number of edges in one-layer merged bucket network is* $4n \cdot \sqrt{\mu n} + n - \sqrt{\mu n}$.

Notice that the same claim works for size, if we assume a linear size anonymous transaction.

Claim 6.3. *For the one-layer merged bucket network, the expected number of edges is minimized when the buckets are merged* μ*-wise.*

Claim 6.4. *If a log-size anonymous transaction is used and* $n > 2\mu$*, the total size of the transactions using the one layer merged bucket is bounded by* $3n \log n$.

Proof-of-Concept Evaluation With Ring Signatures. We now provide a proof of concept evaluation of our protocol for the one-layer case, when anonymous transactions are instantiated using the Concise Linkable Spontaneous Anonymous Group (CLSAG) signature [27] which is currently used by Monero.

We conduct our experiment on a Intel Core 2.90GHz processor using the library https://crates.io/crates/nazgul[3] that implements CLSAG. In Table 1, for each ring size listed, we report the average signing and verification costs (in ms) over 5 iterations of experiment. For a naive comparison, we consider the case where all users use a ring size as large as the number of mixing parties.

Using the negative binomial distribution for our noise sampling mechanism, we give the comparisons of signing/verification costs between the naive approach and our one layer merged bucket protocol under different privacy parameters. In particular, we estimate the average ring size for each transaction and interpolate the result of our experiment to calculate the signing/verification cost. Figure 5 shows the expected signing cost per party while Figure 6 shows the expected verification cost of all parties.

3 We post an issue on their GitHub and apply a small fix in our experiment.

Ring size	Sign	Verify	Ring size	Sign	Verify
2	4.4	3.2	256	446.6	441.2
4	8	7	512	888.2	885.8
8	15	14.2	1024	1777.8	1773.4
16	29.2	27.6	2048	3558.4	3542
32	57.2	56	4096	7098.2	7063.8
64	113	111	8192	14179.2	14132
128	224.6	221.4	16384	28406.6	28242.2

Table 1. Signing and verification times (ms) for CLSAG

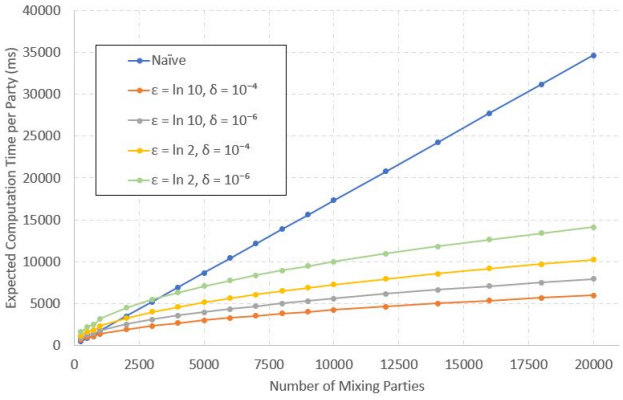

Fig. 5. Expected signing time per party for naive and one layer merged bucket with different privacy parameters.

Concretely, if we set $\epsilon = \ln 10$ and $\delta = 10^{-4}$ (a selection consistent with the literature, i.e. in Stadium [57]), we sample an average of 37 noisy addresses in each bucket. Thus, if the number of mixing parties is greater than 605, our protocol is computationally more efficient than the naive approach.

We note that the verification time is extremely high (both for our solution, and the for the naive construction). When amortized over the number of transactions supported (2500-20,000), this might be deemed reasonable, but we also envision several ways of reducing this cost in the future. Batched verification of ring signatures is an active area of research [49]. Additionally, the task of verifying could be divided up among miners, at random, with sufficient redundancy to ensure that every signature is verified by at least one honest miner.

6.2 Multi Layer Butterfly

Our protocol also works on a multi-layer butterfly network which can reduce computation costs at the price of increased communication costs.

Asymptotic Analysis. Using multiple layers of intermediate buckets introduces more leakage than the one

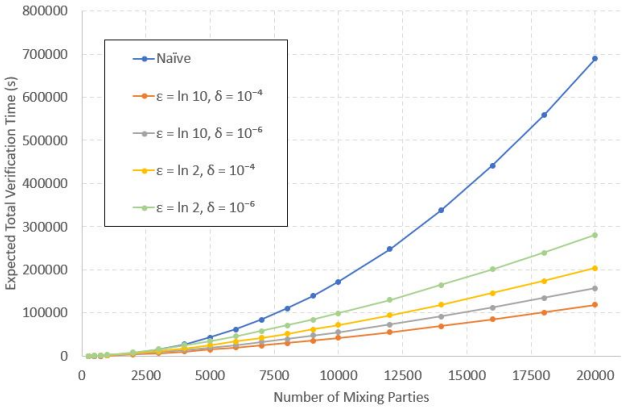

Fig. 6. Expected total verification time for naive and one layer merged bucket with different privacy parameters.

layer case. Specifically, the overall sensitivity is multiplied by $(d-1)$. In particular, a 2-ary butterfly network has $\log n - 1$ times larger sensitivity than the sensitivity of the one layer case. Naturally, an increase in noise magnitude is required to cope with the increased sensitivity. Let μ_m denote the expected noise required in each bucket. Similar to the analysis for one-layer case in Section 6.1, we can capture the expected computation cost by counting the expected number of edges. We give the following theorem:

Theorem 6.5. *The expected number of edges for 2-ary butterfly network is* $O(n \log n \cdot \mu_m^2)$.

We defer the proof to Appendix F.

Noise Choices. We also give an empirical analysis on choosing μ_m and defer it to Appendix H.

Finally, we point out that the merged bucket technique for the one-layer case does not work well for a 2-ary butterfly network, and it is generally inefficient for any butterfly network with larger than constant depth. For a high level intuition, the small branch factor in such cases limits the number of buckets that already share the same set of parent and child buckets before the merge. As a result, we can not claim a similar sublinear increase for the in-degree and out-degree of the merged bucket, ruining the efficiency gain.

7 Fees and DDoS Prevention

In our construction, as the noise responsibility at all buckets is evenly divided, a participant can potentially generate noisy addresses that does not lie on any of her

paths. This is problematic as the participant does not own a real address in the parent buckets, hence cannot create a valid transaction. In addition to that, most cyptocurrencies require the users to pay a transaction fee for each transaction. A participant also needs to deposit some funds to the noisy address's parent address, allowing it to pay the transaction fee. Without a separate address that is unlinkable to her input address to pay the fee, she inevitably needs to diverge the fee from her input address, which exposes links between her input address and her noisy addresses. Also, it is worth noting that a naive alternative of evenly splitting fee paying responsibility among all participants will facilitate a DDoS attack, as a malicious party can flood noise into a bucket with a relatively small cost, causing the protocol to halt to prevent the computation penalty of making large rings.

We present a variant construction for the one-layer case to allow participants to fund their noisy transactions with safe leakage. In particular, we add one more layer of n buckets between the input layer and the intermediate layer and call it "inserted layer". Each participant generates a real address in the inserted layer bucket directly below its input node. For every noisy address she sampled in the intermediate layer, she creates an address in any one of its parent buckets with uniform probability. For the rest of this subsection, we refer to this address in the parent bucket simply as a noisy parent address. Apparently, if this parent address contains enough funds, it can make a valid transaction to the noisy address in the intermediate layer, and the transaction itself can be completely indistinguishable to a real transaction if it ropes-in the same set of addresses as another transaction. It remains to see how this parent address should receive funds.

The solution is quite simple yet counter-intuitive. The participant simply makes one multi-output transaction between the input layer and the inserted layer. While she transfers the majority of her mixing value to the real address directly below, she also allocates a certain amount of values to each noisy parent address to cover the fee later. This makes our protocol satisfying fee requirement, and posts a financial penalty for flooding noises. In addition, as the amount of noise each participant contributed is open, all honest participants can agree on a threshold to exclude those flooding noise.

Now we informally argue why the extra inserted layer and exposing the interconnection between the input layer and inserted layer (notice we do not require the participant to rope in other input addresses) do not degrade our previous privacy guarantee. We show that it is possible to simulate both the inserted layer bucket weight and the interconnection between the first two layers given the bucket weight of intermediate layers. First, let us consider any subset of intermediate buckets that are parented by the same set of inserted layer buckets. The number of real addresses within any set of buckets is equal to the number of parent buckets, as each inserted layer bucket only contains one real address. By summing the intermediate buckets' weight, we can also get the number of total noisy addresses. To simulate the view, it suffice to create an address corresponding to a real address in each one of the inserted layer bucket. And we randomly position the parent noisy addresses with each address uniformly choose a parent bucket. Next, to complete the interconnection between the input layer and inserted layer, we can assign the ownership through following measure: First, we determine the probability that s noisy addresses are contributed by t participant each with s_i noisy address. We first sample a set of s_i. Due to symmetry of each participant generating noise, we can randomly attribute s_i to any t of them. Finally, we connect each participant's input address and the inserted layer addresses accordingly.

8 Acknowledgments

The authors thank the anonymous reviewers for their insightful comments and valuable feedback on this work. This material is supported by a ZCash Foundation Grant [65]. S. Dov Gordon is supported by NSF Grants CNS-1942575 and CNS-1955264, by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under Contract No. N66001-15-C-4070, by the Blavatnik Interdisciplinary Cyber Research Center at Tel-Aviv University and Israel National Cyber Directorate (INCD), and by a Google faculty award. Mayank Varia is supported by the DARPA SIEVE program under Agreement No. HR00112020021, DARPA and the Naval Information Warfare Center (NIWC) under Contract No. N66001-15-C-4071, and the National Science Foundation under Grants No. 1414119, 1718135, 1739000, 1801564, 1915763, and 1931714. Foteini Baldimtsi is supported by NSF Grant CNS-01717067, by NSA Grant 204761 (under a CMU Subcontract No. 1990713-40018), by an IBM faculty award and by a Facebook faculty award.

References

[1] Masayuki Abe. Mix-networks on permutation networks. In Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, *Advances in Cryptology – ASIACRYPT'99*, volume 1716 of *Lecture Notes in Computer Science*, pages 258–273, Singapore, November 14–18, 1999. Springer, Heidelberg, Germany.

[2] Elli Androulaki, Ghassan Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In *FC*, 2013.

[3] Michael Backes, Praveen Manoharan, and Esfandiar Mohammadi. TUC: time-sensitive and modular analysis of anonymous communication. In *CSF*, 2014.

[4] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *CRYPTO*, 2017.

[5] Amos Beimel, Kobbi Nissim, and Mohammad Zaheri. Exploring differential obliviousness. *CoRR*, abs/1905.01373, 2019.

[6] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR ePrint*, 2018.

[7] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE-SP*, 2014.

[8] Alex Biryukov and Sergei Tikhomirov. Deanonymization and linkability of cryptocurrency transactions based on network analysis. In *EuroS&P*, 2019.

[9] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A. Kroll, and Edward W. Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In *FC*, 2014.

[10] Elette Boyle, Saleet Klein, Alon Rosen, and Gil Segev. Securing abe's mix-net against malicious verifiers via witness indistinguishability. In Dario Catalano and Roberto De Prisco, editors, *SCN 18: 11th International Conference on Security in Communication Networks*, volume 11035 of *Lecture Notes in Computer Science*, pages 274–291, Amalfi, Italy, September 5–7, 2018. Springer, Heidelberg, Germany.

[11] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*, pages 423–443. Springer, 2020.

[12] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE-SP*, 2018.

[13] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. http://eprint.iacr.org/2000/067.

[14] Ran Canetti. Universally composable signature, certification, and authentication. In *CSFW-17*, 2004.

[15] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany.

[16] Ran Canetti, Kyle Hogan, Aanchal Malhotra, and Mayank Varia. A universally composable treatment of network time. In *CSF*, 2017.

[17] Ran Canetti, Daniel Shahaf, and Margarita Vald. Universally composable authentication and key-exchange with global PKI. In *PKC*, pages 265–296, 2016.

[18] T.-H. Hubert Chan, Kai-Min Chung, Bruce M. Maggs, and Elaine Shi. Foundations of differentially oblivious algorithms. In *SODA*, 2019.

[19] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *J. Cryptology*, 1:65–75, 1988.

[20] Coinmarketcap https://coinmarketcap.com/. Accessed 2/25/2021.

[21] Adrian Zmudzinski (Cointelegraph). Chainalysis rolls out real-time threat detector for 15 major cryptos), 2019. Accessed 9/23/2020.

[22] Joshua Mapperson (Cointelegraph). Understanding litecoin's dusting attack: What happened and why), 2019. Accessed 9/23/2020.

[23] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284, New York, NY, USA, March 4–7, 2006. Springer, Heidelberg, Germany.

[24] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9:211–407, August 2014.

[25] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. In *Asiacrypt*, 2019.

[26] Steven Goldfeder, Harry A. Kalodner, Dillon Reisman, and Arvind Narayanan. When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies. *Proc. Priv. Enhancing Technol.*, 2018(4):179–199, 2018.

[27] Brandon Goodell, Sarang Noether, and RandomRun. Concise linkable ring signatures and forgery against adversarial keys. Cryptology ePrint Archive, Report 2019/654, 2019. https://eprint.iacr.org/2019/654.

[28] Jens Groth. On the size of pairing-based non-interactive arguments. In *Eurocrypt*, pages 305–326. Springer, 2016.

[29] Andrey Gubichev. Online-routing on the butterfly network: probabilistic analysis, 2008. Accessed 9/26/2020.

[30] Xi He, Ashwin Machanavajjhala, Cheryl J. Flynn, and Divesh Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1389–1406, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

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

[32] Shiva Prasad Kasiviswanathan and Adam D. Smith. A note on differential privacy: Defining resistance to arbitrary side information. *CoRR*, abs/0803.3946, 2008.

[33] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *TCC*, 2013.

[34] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Accessing data while preserving privacy. *CoRR*, abs/1706.01552, 2017.

[35] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In *EUROCRYPT*, 2016.

[36] Robin Klusman and Tim Dijkhuizen. Deanonymisation in ethereum using existing methods for bitcoin, 2018. Technical Report, https://delaat.net/rp/2017-2018/p61/report.pdf.

[37] Russell WF Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. Omniring: Scaling up private payments without trusted setup. *IACR Cryptology ePrint Archive*, 2019:580, 2019.

[38] David Lazar, Yossi Gilad, and Nickolai Zeldovich. Karaoke: Distributed private messaging immune to passive traffic analysis. In *USENIX*, 2018.

[39] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updateable structured reference strings. *IACR Cryptology ePrint Archive*, 2019:99, 2019.

[40] Felix Konstantin Maurer, Till Neudecker, and Martin Florian. Anonymous coinjoin transactions with arbitrary values. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 522–529. IEEE, 2017.

[41] Sahar Mazloom and S. Dov Gordon. Secure computation with differentially private access patterns. In *ACM-CCS*, 2018.

[42] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *IMC*, 2013.

[43] Monero, https://getmonero.org/home. Accessed 2/25/2021.

[44] Pedro Moreno-Sanchez, Tim Ruffing, and Aniket Kate. Pathshuffle: Credit mixing and anonymous payments for ripple. *PoPETS*, 2017.

[45] Pedro Moreno-Sanchez, Muhammad Bilal Zafar, and Aniket Kate. Listening to whispers of ripple: Linking wallets and deanonymizing transactions in the ripple network. *PoPETs*, 2016(4):436–453, 2016.

[46] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. An empirical analysis of traceability in the monero blockchain. *PoPETs*, 2018(3):143–163, 2018.

[47] Mahnush Movahedi, Jared Saia, and Mahdi Zamani. Secure multi-party shuffling. In Christian Scheideler, editor, *Structural Information and Communication Complexity*, pages 459–473, Cham, 2015. Springer International Publishing.

[48] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[49] Sarang Noether and Brandon Goodell. Triptych: Logarithmic-sized linkable ring signatures with applications. In Joaquin Garcia-Alfaro, Guillermo Navarro-Arribas, and Jordi Herrera-Joancomarti, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 337–354, Cham, 2020. Springer International Publishing.

[50] Shen Noether, Adam Mackenzie, et al. Ring confidential transactions. *Ledger*, 1:1–18, 2016.

[51] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *ASIACRYPT*, 2001.

[52] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In *FC*. Springer, 2013.

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

[54] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. P2P mixing and unlinkable bitcoin transactions. In *NDSS*, 2017.

[55] Amitabh Saxena, Janardan Misra, and Aritra Dhar. Increasing anonymity in bitcoin. In *FC*, pages 122–139. Springer, 2014.

[56] Various sources (Coin Metrics). Average number of daily cryptocurrency transactions in 1st quarter of 2019, by type (in thousand transactions), 2019. Accessed 9/23/2020.

[57] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *SOSP*, pages 423–440, New York, NY, USA, 2017. ACM.

[58] Salil P. Vadhan. The complexity of differential privacy. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography*, pages 347–450. Springer International Publishing, 2017.

[59] István Vajda. On the analysis of time-aware protocols in universal composability framework. *Int. J. Inf. Sec.*, 15(4):403–412, 2016.

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

[61] Nicolas van Saberhagen. Cryptonote v 2.0. *Whitepaper*, 2013.

[62] Sameer Wagh, Paul Cuff, and Prateek Mittal. Differentially private oblivious RAM. *PoPETs*, 2018(4):64–84, 2018.

[63] Tsz Hon Yuen, Shi-Feng Sun, Joseph K. Liu, Man Ho Au, Muhammed F. Esgin, Qingzhao Zhang, and Dawu Gu. Ringct 3.0 for blockchain confidential transaction: Shorter size and stronger security. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security*, pages 464–483, Cham, 2020. Springer International Publishing.

[64] Zcash, https://z.cash/. Accessed 2/25/2021.

[65] ZCash Foundation. GrantProposals-2018Q2. https://github.com/ZcashFoundation/GrantProposals-2018Q2, 2018.

A Additional UC Functionalities

We present a bulletin functionality $\mathcal{G}_{\mathsf{bb}}$ in Figure 7 and a globally-accessible clock functionality $\mathcal{G}_{\mathsf{clock}}$ in Figure 8.

B Protocol Instantiation

Below we present an instantiation of our protocol based on ring signatures and confidential transactions. We start by discussing some background on ring signatures.

B.1 Linkable Ring Signatures and Confidential Transactions

Ring signatures allow a user to dynamically choose a set of public keys (including her own) and to sign messages on behalf of the set, without revealing her identity [51]. *Confidential Transactions* (CT) hide the amounts and replace that information with a proof of balance. Our instantiation assumes basic background on ring signatures and Confidential transactions which can be found in [50] due to page limitations.

To present an instantiation of our overal protocol, we start by proposing an instantiation of our anonymous transaction functionality using ring signatures over confidential transactions (CTs), as defined in Section B.1, for compatibility with Monero. We use a ring signature scheme in order to hide the real sender and therefore the real path within a ring of "possible" senders. CTs are mainly needed to hide the distinction between real transactions along the payment's real path (which transfer non-zero amounts of money) and the transactions containing zero balance (noisy ones that hide the true transaction paths). CTs also allow participants to mix different amounts, without revealing the connection between the input and output addresses.

In protocol 4, addresses are instantiated with ring signature key pairs. An address registration is a coin generation, represented by a public key, accompanied by its secret key knowledge and some amount-loading mechanism (this is usually taken care of by prior CT instances). Address sorting throughout the protocol corresponds to public key ordering.

$\mathcal{G}_{\mathsf{bb}}$: Global Setup Bulletin Board Functionality

Upon initialization, it creates an empty list L.
Write: Upon receiving (WRITE, x) from $\mathcal{G}_{\mathsf{AnonTrans}}$, it appends the record x to L and sends (WRITE, x) to adversary. Ignore requests sent by any other entity.
Read: Upon receiving READ from any party P, it returns L to the requester.

Fig. 7. Global Setup Bulletin Board Functionality

$\mathcal{G}_{\mathsf{clock}}$: Global Clock Functionality

$\mathcal{G}_{\mathsf{clock}}$ maintains an integer C corresponding to the absolute time. When created, the box initializes $C = 0$.

IncrementTime: Upon receiving INCREMENT-TIME from the environment, update $C = C + 1$ and send an "ok" message to the environment. Ignore the request if sent by any other entity.

GetTime: Return C to the requester.

Fig. 8. Global Ideal functionality representing absolute time $\mathcal{G}_{\mathsf{clock}}$.

B.2 Instantiation

We focus on transaction-based systems (UTXO setting) as confidentiality and anonymity in the account-based setting [11] relies on extra tools such as locks and epochs. In the next paragraph, we explain how $\mathcal{G}_{\mathsf{AnonTrans}}$ functionality (Section 2.4) can be converted to capture this setting. Monero is an example of a currency in the UTXO setting that is equipped with ring signatures over CTs.

We instantiate $\mathcal{G}_{\mathsf{AnonTrans}}.\mathsf{Transfer}()$ as follows: A successful call corresponds to a valid ring signature with S being the corresponding signature ring $\mathcal{R}$ and the message signed being the transaction information, including the transferred value, which is hidden under a commitment as part of the CT. (Signing is also used to prevent transaction information tampering by other users.) As in Monero, we will assume *linkable* ring signatures, where each key can be only used once for signing, in order to avoid double-spending attacks. To capture this transaction-based style, we would have to modify $\mathcal{G}_{\mathsf{AnonTrans}}$ to delete the record of the performer of a successful transfer. The registration check during transfer corresponds to the knowledge of a secret key for the to-be-spent coin (proved through signing). The combined use of CT guarantees balance preservation. In transaction-based systems with one-time spendable coins, this corresponds to a proof that input amounts exceed output amounts. It also realizes the receiver's $\mathsf{v}'_r + \mathsf{v}_r$ aggregate balance, in a sense that two different CTs with the same receiver can be merged under a new one and be spent as a whole. This is how anonymity mechanisms work on Monero [43] if we ignore one-time addresses used for recipient unlinkability, i.e. we immediately use the receiver's public key instead of some

other key derived from it. In mixing protocols, we can do this because users pay themselves.

Given the above instantiation of $\mathcal{G}_{\mathsf{AnonTrans}}.\mathsf{Transfer}()$, we now present how our main protocol (Figure 4) can be instantiated. We instantiate terminal/intermediate addresses in phases 1 and 2 as pairs of ring signature keys. We note that both real and noisy addresses are instantiated the same way, and the moment they get broadcast, they do not yet carry any value. The synchronized transactions phase (phase 2 of the protocol) involves the following step, repeated for every layer of the network: "each participant is responsible to create transactions paid to all addresses she created in that layer". This is done with calls to $\mathcal{G}_{\mathsf{AnonTrans}}.\mathsf{Transfer}()$ as described in lines 29,31 of Figure 4. Instead of choosing the rings, senders construct rings according to the addresses of the network. The anonymity set S (or else the ring $\mathcal{R}$) is the set of all possible payers (called parents) according to the network topology. The calls are identical for transactions to real and noisy addresses. In other words, these transactions can either carry real amount to real addresses, or be *noisy transactions*, carrying a zero payment amount to a noisy address.

Multi-Layer Instantiation. In the case of noisy transactions, the transaction sender might not know any secret for the predefined input ring. This is the case when the sender has to make a noisy transaction that, due to the butterfly network structure, originates from a bucket in which the sender does not own a key. Note however, that the sender will always own the secret key that corresponds to the receiving address of the transactions. Thus, we make a slight modification in the usual ring signature content by allowing the receiving address (which is essentially a public key) to be part of the sending ring. More precisely, if a a key pair $(\mathsf{sk}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{out}})$ exists in bucket $\langle i, j \rangle$, the input Ring $\mathcal{R}$ will be $(\mathsf{Parent}(\langle \mathsf{i}, \mathsf{j} \rangle, \mathsf{pk}_{\mathsf{out}}).\mathsf{append}(\{\mathsf{pk}_{\mathsf{out}}\}))$ and the output will be pk_{out}.

This modification of including the output in the ring allows for what we call a *loop payment*: a payment from pk_{out} to pk_{out}. While this action is not forbidden by Monero, we remark that it has little value outside of our protocol. Specifically, in systems that use UTXOs (or transaction-based cryptocurrencies), each coin can be spent only once. This implies that a loop-transaction is a coin that cannot be used in the future. In our instantiation, we only use loop payments for noisy transactions (of value zero), thus we do not care about moving such an amount further in the network. However, we still care about indistinguishability between real and noisy transactions. More specifically, we have the following two types of transactions:

1. type 1: input key = output key, i.e. zero-amount transaction that is not used to transfer amounts to the next layer
2. type 2: input key $\neq$ output key, i.e. real payment transaction that is used to transfer amounts to the next layer

Our protocol keeps type 1 and type 2 transactions indistinguishable and eventually hides the payments' true paths. An observer cannot tell if for two sequential transactions, it was the case of two type 2 payments signed by the same participant (real path) or any other combination of type 1/type 2 payments signed by the same or different participants (that own a key in the ring). The two types of transactions can become distinguishable in the case where there is no noise or real keys in the parent buckets. This happens with a very small probability and the real path can still be hidden in the rest of the layers.

Confidential Transactions (CTs) are usually implemented using commitments to the address' amount (in Monero for example) or more generally with a binding function (as defined in [37]) and a proof of balance between inputs and outputs aggregated. By using the modified ring signature format we proposed above, the input ring now includes a new key that is not bound to any amount yet. In other words, it is not the output of any previous valid transaction. Fortunately, it is expected that these new keys will be distinguishable and known to carry zero amount, therefore users can post intermediate keys along with a commitment and a proof that it opens to zero (a technique already suggested in [25]). The moment this address is used as the transaction output, its commitment immediately changes (new randomness is added no matter if this was a zero-amount or a real transaction).

C Proof of Lemmas 5.1 and 5.2

Proof of Lemma 5.1. Let $x = f_1(\pi)$. It suffices to separately prove the following two inequalities: For arbitrary $\mathcal{S} \subseteq \mathrm{Range}(\mathcal{M}_1)$,

$$\Pr[\mathcal{M}_1(x) \in \mathcal{S}] \leq e^{\epsilon} \Pr[\mathcal{M}_1(x+1) \in \mathcal{S}] + \delta \quad (1)$$

$$\Pr[\mathcal{M}_1(x+1) \in \mathcal{S}] \leq e^{\epsilon} \Pr[\mathcal{M}_1(x) \in \mathcal{S}] \quad (2)$$

In particular, for inequality (2), we will show the difference could be bounded by e^{ϵ} alone whenever $\epsilon \geq \ln(1/p)$.

Let y denote the noise that $\mathcal{M}_1$ adds; it is sampled from the negative binomial distribution $\mathrm{NB}(r, p)$. And let $\mathrm{NB}_{r,p}(y)$ denote the probability to sample y. To show that inequality (1) holds, it is equivalent to showing, for arbitrary $Y \subseteq \{0, 1, 2, \dots\}$, that

$$\sum_{y \in Y} \mathrm{NB}_{r,p}(y) \leq e^{\epsilon} \sum_{y \in Y} \mathrm{NB}_{r,p}(y-1) + \delta$$

In particular, we show the maximum additive difference between $\sum_{y \in Y} \mathrm{NB}_{r,p}(y)$ and $e^{\epsilon} \sum_{y \in Y} \mathrm{NB}_{r,p}(y-1)$ over all choice of S is upper bounded by δ. First, for all $y \geq \frac{p(r-1)}{e^{\epsilon}-p}$, we have

$$\begin{aligned} \frac{\mathrm{NB}_{r,p}(y)}{\mathrm{NB}_{r,p}(y-1)} &= \frac{\binom{y+r-1}{y} \cdot (1-p)^r \cdot p^y}{\binom{y+r-2}{y-1} \cdot (1-p)^r \cdot p^{y-1}} \\ &= \frac{p(y+r-1)}{y} \leq e^{\epsilon} \end{aligned}$$

Hence, it suffices to focus on the case that $y < \frac{p(r-1)}{e^{\epsilon}-p}$ to bound the additive difference. Recall that we let $k = \left\lfloor \frac{p(r-1)}{e^{\epsilon}-p} \right\rfloor$. Moreover, the maximum additive difference is achieved when Y contains all $y \leq k$. Hence, we have:

$$\begin{aligned} \delta &= \sum_{y \in [0,k]} (\mathrm{NB}_{r,p}(y) - e^{\epsilon} \mathrm{NB}_{r,p}(y-1)) \\ &= \sum_{y \in [0,k]} \mathrm{NB}_{r,p}(y) - e^{\epsilon} \sum_{y \in [0,k-1]} \mathrm{NB}_{r,p}(y) \\ &= F_{\mathrm{NB}}(k; r, p) - e^{\epsilon} F_{\mathrm{NB}}(k-1; r, p) \\ &= I_{1-p}(r, k+1) - e^{\epsilon} I_{1-p}(r, k), \end{aligned}$$

where the second equality holds because $\mathrm{NB}_{r,p}(-1) = 0$. Recall $I.(\cdot, \cdot)$ is the regularized incomplete beta function.

Now assume $\epsilon \geq \ln(1/p)$ for the rest of the proof, to show that inequality (2) holds, it is equivalent to show for arbitrary $Y \subseteq \{0, 1, 2, \dots\}$, we have:

$$\sum_{y \in Y} \mathrm{NB}_{r,p}(y-1) \leq e^{\epsilon} \sum_{y \in Y} \mathrm{NB}_{r,p}(y)$$

And it suffices to show for any single $y \in \{1, 2, 3, \dots\}$:

$$\mathrm{NB}_{r,p}(y-1) / \mathrm{NB}_{r,p}(y) \leq e^{\epsilon}$$

This holds as

$$\begin{aligned} \frac{\mathrm{NB}_{r,p}(y-1)}{\mathrm{NB}_{r,p}(y)} &= \frac{\binom{y+r-2}{y-1} \cdot (1-p)^r \cdot p^{y-1}}{\binom{y+r-1}{y} \cdot (1-p)^r \cdot p^y} \\ &= \frac{y}{p(y+r-1)} \leq \frac{1}{p} \leq e^{\epsilon} \end{aligned}$$

where the second to last inequality is due to $r \geq 1$.

Proof of Lemma 5.2. We prove that for arbitrary $\mathcal{S} = \{(z_1, z_2)\} \in \mathrm{Range}(\mathcal{M}_2)$, the following inequality holds:

$$\begin{aligned} &\Pr[\mathcal{M}_2(x_1, x_2) \in \mathcal{S}] \\ &\leq e^{2\epsilon} \Pr[\mathcal{M}_2(x_1 - 1, x_2 + 1) \in \mathcal{S}] + \delta. \end{aligned}$$

The proof for the other case with $f_2(\pi') = (x_1 + 1, x_2 - 1)$ follows from symmetry.

Let $\mathcal{S}_1 = \{z_1 \mid \exists z_2 : (z_1, z_2) \in \mathcal{S}\}$, let $\mathcal{S}_2(z_1) = \{z_2 \mid (z_1, z_2) \in \mathcal{S}\}$. We have:

$$\begin{aligned} &\Pr[\mathcal{M}_2(x_1, x_2) \in \mathcal{S}] \\ &= \sum_{z_1 \in \mathcal{S}_1} \Pr[\mathcal{M}_1(x_1) = z_1] \Pr[\mathcal{M}_1(x_2) \in \mathcal{S}_2(z_1)] \\ &\leq \sum_{z_1 \in \mathcal{S}_1} \Pr[\mathcal{M}_1(x_1) = z_1](e^{\epsilon} \Pr[\mathcal{M}_1(x_2 + 1) \in \mathcal{S}_2(z_1)] \\ &\quad + \delta) \\ &= \sum_{z_1 \in \mathcal{S}_1} e^{\epsilon} \Pr[\mathcal{M}_1(x_1) = z_1] \Pr[\mathcal{M}_1(x_2 + 1) \in \mathcal{S}_2(z_1)] + \delta \\ &\leq \sum_{z_1 \in \mathcal{S}_1} e^{2\epsilon} \Pr[\mathcal{M}_1(x_1 - 1) = z_1] \Pr[\mathcal{M}_1(x_2 + 1) \in \mathcal{S}_2(z_1)] \\ &\quad + \delta. \\ &= e^{2\epsilon} \Pr[\mathcal{M}_2(x_1 - 1, x_2 + 1) \in \mathcal{S}] + \delta \end{aligned}$$

The first inequality is due to (1) and the second inequality is due to (2).

D Proof of Theorem 5.5

Since we have already shown that $\mathcal{L}$ preserves $(\bar{\epsilon}, \bar{\delta})$ privacy, it only remains to show that the main protocol presented above (Figure 4) UC-realizes the ideal functionality $\mathcal{F}^{T}_{\mathsf{AnonMix}}$, presented in Figure 3. We now describe a simulator and argue that it emulates the real world execution up to statistical error $\bar{\delta}$.

Our simulator locally executes the code of all honest parties, it interfaces with the real environment, adversary, and global functionalities $\mathcal{G}_{\mathsf{AnonTrans}}$, $\mathcal{G}_{\mathsf{bb}}$ and $\mathcal{G}_{\mathsf{clock}}$. The simulator relays and monitors all traffic sent between the emulated parties/functionalities, the adversary, and the external environment. Some of these messages cause the simulator to perform actions in the ideal world with $\mathcal{F}^{T}_{\mathsf{AnonMix}}$, with the (potentially dummy) adversary, or with the environment, as detailed below.

Address Registration. The simulator reads $\mathcal{G}_{\mathsf{bb}}$ to collect the input addresses of corrupted parties that will participate in the next mixing period. The simulator

then chooses output addresses for the malicious parties arbitrarily, and submits these input-output pairs to $\mathcal{F}^T_{\mathsf{AnonMix}}$'s registration routine. The simulator also forwards to the adversary any input addresses received from the functionality that were registered by honest parties during registration.

Leakage Function $\mathcal{L}$. At Time T, the simulator queries the ideal functionality using the Ping command, and it receives in response the output $\mathbf{O}$ from the leakage function $\mathcal{L}$ and the bucket weights $\vec{w}$. The simulator forwards the full list of registered addresses to the adversary.

Recall that $\mathbf{O}$ contains the integer weight for the real + noisy addresses contained within each bucket of the butterfly network, where bucket sizes at the output layer are publicly known (as defined in Section 5.1). The simulator's goal is to construct a butterfly network consistent with $\mathbf{O}$.

To do this, the simulator must create a new matrix $\mathbf{O}^*$ that removes the real and noisy addresses that the leakage contains from the corrupted parties. To remove the real addresses, the simulator removes one node from each bucket along the path between the input and output address that it registered for each adversarial party. To remove the noisy addresses, the simulator subtracts m samples of the negative binomial distribution from each entry in $\mathbf{O}$, where m denotes the number of adversarial parties. (Recall from the theorem statement that we assume that the set of dishonest parties remains static throughout the mixing protocol.) The resulting matrix $\mathbf{O}^*$ is a perfect simulation of the honest parties' bucket weights in the real world because the negative binomial distribution is additive.

Intermediate Addresses. The simulator generates intermediate addresses (these include both noisy and real addresses) on behalf of the honest parties, ensuring that the number of keys is consistent with $\mathbf{O}^*$. He forwards these to the adversary and awaits a list of intermediate addresses in response. Any corrupted party that fails to post their intermediate addresses in the ledger is marked locally by the simulator as having aborted the protocol. Any party that was previously marked as having aborted is ignored if it sends intermediate addresses.

Creating Synthetic Transactions. The simulator uniformly samples a 1-to-1 mapping from the honest input keys to the honest output keys, and creates a path through the butterfly network for each pair in the mapping. Note that such a path exists with probability at least $1 - \bar{\delta}$ since an empty path in the butterfly network is the "failure" case in which someone's privacy has been breached.

Posting Transactions. For each edge in the butterfly network, the simulator sends a transaction carrying value v to the ideal functionality. Additionally, Sim creates sufficiently many "dummy" transactions carrying 0 value until the weight of each node matches its value in $\mathbf{O}^*$.

The simulator posts all of these transactions to the anonymous transaction functionality, and Sim also sends these transactions within the emulated world's mixing protocol.

Communication With the Environment. All messages from the adversary to the environment are faithfully forwarded by the simulator.

To conclude our proof, as per Definition 2.9, we need to argue why our simulator provides a statistical emulation of the real world protocol. During *address registration* the honest addresses are created exactly as in the real world, so the views of the adversary are identical. The adversary can also choose its own own adversarial addresses and the decision of whether to abort will also faithfully follow the same decision paths as in the real world. The synthetically-chosen transactions are also sampled from the appropriate negative binomial distribution as long as the matrix $\mathbf{O}^*$ permits at least 1 node in each of the buckets along the uniformly-chosen real path, which happens with probability at least $1-\bar{\delta}$. The *leakage function* is determined exactly the same way as in the real world so this is indistinguishable from the point of view of the adversary. The simulator picks the *intermediate* addresses from the same distribution as in the real protocol and ensures that the number of addresses per bucket is picked in accordance to $\mathcal{L}$ thus the view of the adversary is again identical to the real protocol. The most critical part is to guarantee indistinguishability during the *posting transactions* phase. We first note that the interaction of the $\mathcal{F}^T_{\mathsf{AnonMix}}$ functionality with the other functionalities is specifically designed to capture the exact use of these functionalities in the real world, and all messages from the adversary to the environment are faithfully forwarded. During *transaction posting* the simulator does not know the true permutation of input/output addresses that was sampled by the functionality and samples his own 1-to-1 mapping between honest input and output addresses. We note however that because all honest parties mix the *same value* v, the environment (or adversary) cannot determine whether the correct permutation was used: everyone terminates with the right balance, even if they may receive v from the wrong sender.

E Proof of Theorem 5.6

We start by recognizing that the adversary can only passively observe the protocol. In particular, the *Out* given to the adversary is captured the leakage function defined in Theorem 5.4. Still, due to the potential loss of noisy transactions by the adversarial parties, $n/(n-t)$ times of original noise magnitude is used. Eventually, this allow us to argue the leakage from $\Pi^{\vec{s},\vec{v},\vec{r}}_{T,n,d,c}$ is still $(\bar{\epsilon}, \bar{\delta})$-indistinguishable as proven in Theorem 5.4.

We prove the bound of the adversary advantage by contradiction. We assume the adversary can win with advantage greater than $\frac{\bar{\epsilon}}{2} + \bar{\delta}$. In particular, we regard the adversary as a function f that takes the output of the leakage function and output a guess bit b'. We use the notation like $\Pr[b' = 0|\pi_0]$ to denote the probability that f output 0 conditioned on the challenger choosing π_0. The following inequality shows an adversary with advantage greater than $\frac{\bar{\epsilon}}{2} + \bar{\delta}$:

$$\begin{aligned}&\Pr[b' = 0|\pi_0] + \Pr[b' = 1|\pi_1] \\ &- \Pr[b' = 0|\pi_1] - \Pr[b' = 1|\pi_0] > 2(\bar{\epsilon}/2 + \bar{\delta})\end{aligned}$$

Note that:

$$\begin{aligned}&\Pr[b' = 0|\pi_0] - \Pr[b' = 0|\pi_1] \\ &= (1 - \Pr[b' = 1|\pi_0]) - (1 - \Pr[b' = 1|\pi_1]) \\ &= \Pr[b' = 1|\pi_1] - \Pr[b' = 1|\pi_0] \\ &> \bar{\epsilon}/2 + \bar{\delta}.\end{aligned}$$

Clearly, at least one of the $\Pr[b' = 0|\pi_1]$ and $\Pr[b' = 1|\pi_0]$ is smaller than $1/2$. Without loss of generality, assume $\Pr[b' = 0|\pi_1] < 1/2$. Therefore:

$$\begin{aligned}\Pr[b' = 0|\pi_0] - \Pr[b' = 0|\pi_1] &> \frac{\bar{\epsilon}}{2} + \bar{\delta} \\ \Pr[b' = 0|\pi_0] &> \Pr[b' = 0|\pi_1] + \frac{\bar{\epsilon}}{2} + \bar{\delta} \\ &> \Pr[b' = 0|\pi_1] + \bar{\epsilon}\Pr[b' = 0|\pi_1] + \bar{\delta} \\ &> (1 + \bar{\epsilon})\Pr[b' = 0|\pi_1] + \bar{\delta} \\ &= e^{\bar{\epsilon}}\Pr[b' = 0|\pi_1] + \bar{\delta} \text{ (if } \bar{\epsilon} = 0),\end{aligned}$$

where the third inequality is because $\Pr[b' = 0|\pi_1] < 1/2$. Notice that this inequality violates the post-processing property (Lemma 2.4). This concludes our proof.

F Proof in Section 6

Proof of Theorem 6.1

Let $X_1, ..., X_n$ denote the random variables for the number of real addresses in the intermediate layer buckets. Clearly, each X_i is not independent of the others; one of the constraints is $\sum_{i=1}^{n} X_i = n$. Let $Y_1, ..., Y_n$ denote the independent and identically distributed random variables for the number of noisy addresses sampled from a negative binomial distribution with mean μ in each bucket.

We first consider the expected edge count. The edges between the input layer and the intermediate layer can be easily calculated as in degree for every address is fixed, i.e.,$\sqrt{n}$. Thus, the expected number of edges in between the first two layers is $\mathrm{E}[\sqrt{n}\sum_{i=1}^{n}(X_i + Y_i)] = (1 + \mu)n^{1.5}$.

For the edges between the intermediate layer and the output layer, we group every $\sqrt{n}$ buckets that share the same "butterfly", i.e. they connect to the same set of $\sqrt{n}$ output buckets. Let $S_1, ..., S_{\sqrt{n}}$ (resp. $T_1, ..., T_{\sqrt{n}}$) denote the random variable for the number of real (resp. noisy) addresses in each of these groups. Notice that $\mathrm{E}[S_i] = \sqrt{n}$, $\mathrm{E}[T_i] = \sqrt{n}\mu$ and the variance $\mathrm{Var}[S_i] = n \cdot (1/\sqrt{n} \cdot (1 - 1/\sqrt{n})) = \sqrt{n} - 1$, as S_i follows binomial distribution with $p = 1/\sqrt{n}$.

The expected number of edges in between can be calculated as:

$$\mathrm{E}\left[\sum_{i=1}^{\sqrt{n}}(S_i + T_i)S_i\right] = \sum_{i=1}^{\sqrt{n}}\mathrm{E}[S_i^2] + \mathrm{E}[T_i \cdot S_i]$$

$$= \sum_{i=1}^{\sqrt{n}}(\mathrm{E}[S_i]^2 + \mathrm{Var}[S_i] + \mathrm{E}[T_i] \cdot E[S_i])$$

(T_i and S_i are independent to each other.)

$$= \sum_{i=1}^{\sqrt{n}}(n + \sqrt{n} - 1 + \sqrt{n} \cdot \mu\sqrt{n}) = (1 + \mu)n^{1.5} + n - \sqrt{n}.$$

Finally, due to the linearity of expectation, the total expected edges is $2(1 + \mu)n^{1.5} + n - \sqrt{n} \in O(n^{1.5})$.

We now prove that the total edges count is bounded by $O(n^{1.5} \log n \log\log n)$ except with negligible probability. To do this, we show that X_i is bounded by $\log n$ and Y_i is bounded by $\log n \log\log n$.

For X_i, we have:

$$\Pr[X_i \geq k] \leq \binom{\sqrt{n}}{k}\left(\frac{1}{\sqrt{n}}\right)^k \leq \frac{1}{k!} \leq \left(\frac{e}{k}\right)^k,$$

which is negligible when $k = \log n$.

For $Y_i \sim \mathrm{NB}(r, p)$, we first divide into r i.i.d random variables, $Z_1, \ldots, Z_r \sim \mathrm{NB}(1, p)$, for each Z_j, we have:

$$\Pr[Z_j > k/r] = (1-p)^{k/r+1},$$

which is negligible when $k = \log n \log\log n$. Using the union bound, the probability that there exists a random variable from $Z_1, \ldots, Z_r$ such that one of them is greater than k/r is also negligible. This suggests the probability that $\Pr[Y_i \geq k]$ is also negligible.

Proof of Claim 6.2 The number of real addresses in the intermediate layer of buckets is n, while the expected amount of noise addresses is $\mu \cdot \frac{n}{\mu} = n$. As the in-degree for each merged bucket is $\sqrt{\mu n}$, the total number of edges between the input and intermediate layer is $2n \cdot \sqrt{\mu n}$. For the edge count between the intermediate layer and output layer, we can apply a similar analysis as in the original one-layer case and the expected number of edges is $2n \cdot \sqrt{\mu n} + n - \sqrt{\mu n}$. So the total expected number of edges is $4n \cdot \sqrt{\mu n} + n - \sqrt{\mu n}$.

Proof of Claim 6.3 Assume each merged bucket contains r original buckets, so the total number of merged buckets is $\frac{n}{r}$. The expected number of addresses in the whole intermediate layer is $\mu \cdot \frac{n}{r}$ noisy addresses plus n real addresses. Using the same strategy to pick the merging buckets, we can achieve merged buckets with a $\sqrt{rn}$ in-degree and out-degree. The total number of edges is thus $2 \cdot (n(1+\mu/r)) \cdot \sqrt{rn} = 2n^{1.5} \cdot (\sqrt{r} + \mu/\sqrt{r})$. And this term is minimum when $r = \mu$.

Proof of Claim 6.4 For the one-layer case, the users need to perform two rounds of transactions. The first round corresponds to the upper half of the network, in which we have an average of $2n$ intermediate addresses, each requiring one transaction. All of them have input size $\sqrt{\mu n}$. For the lower half, we have n output addresses, each with an average input size $2\sqrt{\mu n}$. As a result, with log-size anonymous transactions, the total size of all transactions are $2n \cdot \log\sqrt{\mu n} + n \cdot \log(2\sqrt{\mu n}) = \frac{3}{2} n(\log n + \log \mu) + n$. If $n > 2c$, then the cost is bounded by $3n \log n$.

Proof of Theorem 6.5 To calculate the expected number of edges between the lth and $l+1$ layers, let $S_{l,1}, ..., S_{l,n/2}$ (resp. $T_{l,1} ..., T_{l,n/2}$) denote the random variable for the number of real (resp. noisy) addresses in each of the current mixing groups. In addition to that, let $T'_{l,1}, ..., T'_{l,n/2}$ denote the number of noisy address in each group in the $l+1$ layer. (Here the group is determined by the connection between l and $l+1$ layers so that $T'_{l,1}$ and $T_{l+1,1}$ refer to two different random variables, hence the different notations.) Notice that $\mathrm{E}[S_{l,i}] = 2$, $\mathrm{E}[T_{l,i}] = \mathrm{E}[T'_{l,i}] = 2\mu_m$ and the variance $\mathrm{Var}[S_{l,i}] = 2^{l+1} \cdot 2/n \cdot (1 - 2/n) \leq 2$, as $S_{l,i}$ follows binomial distribution with number of trials $n_{trial} = 2^{l+1}$ and $p = 2/n$. Hence, the number of edges can be calculated as:

$$\begin{aligned}
&\mathrm{E}\left[\sum_{i=1}^{n/2} (S_{l,i} + T_{l,i})(S_{l,i} + T'_{l,i})\right] \\
&= \sum_{i=1}^{n/2} \mathrm{E}[S_{l,i}^2] + \mathrm{E}[S_{l,i} \cdot T'_{l,i}] + \mathrm{E}[T_{l,i} \cdot S_{l,i}] + \mathrm{E}[T_{l,i} \cdot T'_{l,i}] \\
&= \sum_{i=1}^{n/2} \mathrm{E}[S_{l,i}]^2 + \mathrm{Var}[S_{l,i}] + \mathrm{E}[S_{l,i}]\mathrm{E}[T'_{l,i}] + \mathrm{E}[T_{l,i}]\mathrm{E}[S_{l,i}] \\
&+ \mathrm{E}[T_{l,i}]\mathrm{E}[T'_{l,i}] \\
&\leq \sum_{i=1}^{n/2} 4 + 2 + 4\mu_m + 4\mu_m + 4\mu_m^2 = O(\mu_m^2 n).
\end{aligned}$$

With $\log n$ layers, the expected number of edges is $O(n \log n \cdot \mu_m^2)$.

G Iterated Butterfly Network

In this section, we demonstrate that the repetitive run of our mixing protocol can achieve a better privacy guarantee. In particular, we define an iterated butterfly network as containing multiple butterfly networks in a sequence, with each network's (except the last one's) output addresses served as the input addresses of the next network. Each butterfly network also independently picks an uniform permutation.

On a very high level, each run of a butterfly network contributes some randomness to the composed permutation, so the repetitive run of butterfly networks makes it harder for the adversary to guess what the final composed permutation is. Without loss of generality, we prove this intuition in the case that we have two butterfly networks.

Formally, let $\mathcal{L}_1, \mathcal{L}_2 : \Pi \to \mathbb{N}^{n \times (d-1)}$ be the leakage functions for the two consecutive butterfly networks. We define the overall leakage function for the whole mixing as $\mathcal{L} : \Pi \to \mathbb{N}^{n \times (d-1)} \times \mathbb{N}^{n \times (d-1)}$ and interpret it as follows: given input permutation π, $\mathcal{L}$ uniformly picks a π_1 and set $\pi_2 = \pi\pi_1^{-1}$ (so that $\pi = \pi_2\pi_1$), and output $\mathcal{L}_1(\pi_1), \mathcal{L}_2(\pi_2)$. Notice this interpretation is equivalent to the original statement.

Theorem G.1. *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be (ϵ_0, δ_0)-indistinguishable. Then $\mathcal{L}$ is (ϵ, δ)-indistinguishable with $\epsilon = \ln(\frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}}) \leq \epsilon_0$ and $\delta = (e^{\epsilon_0} - \frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}} + \frac{\delta_0}{2}) \cdot \delta_0 \leq \delta_0$.*

Proof. Let n be the number of participants. Let π denote the permutation composed by two consecutive mixings, and π' be one of its neighboring permutations. We use $\mathbb{O}$ to be short of $\mathbb{N}^{n\times(d-1)} \times \mathbb{N}^{n\times(d-1)}$ and let $\mathbf{O}$ be any subset of $\mathbb{O}$. Throughout the proof, we use the simplified notation of conditional probability such as $\Pr[\mathbf{O}|\pi]$ rather than $\Pr[\mathcal{L}(\pi) \in \mathbf{O}]$. Given that both rounds of mixing are (ϵ_0, δ_0)-indistinguishable. Our goal is to find ϵ and δ such that for arbitrary $\mathbf{O} \in \mathbb{O}$:

$$\Pr[\mathbf{O}|\pi] \leq e^{\epsilon} \Pr[\mathbf{O}|\pi'] + \delta \tag{3}$$

Let us consider all pair of permutation π_1, π_2 such that $\pi = \pi_2\pi_1$. Since there exists exactly one π_2 for every π_1 that satisfies this equation, there are a total of $n!$ pairs. The probability $\Pr[\mathbf{O}|\pi]$ is essentially averaged (with uniformly sample permutation) across all such pairs, π_1, π_2.

$$\Pr[\mathbf{O}|\pi] = \sum_{\forall \pi_1, \pi_2} \Pr[\pi_1, \pi_2|\pi] \cdot \Pr[\mathbf{O}|\pi_1, \pi_2].$$

According to Bayes rules,

$$\begin{aligned}\Pr[\pi_1, \pi_2|\pi] &= \Pr[\pi|\pi_1, \pi_2] \cdot \Pr[\pi_1, \pi_2] / \Pr[\pi] \\ &= 1 \cdot \frac{1}{n!n!} / \frac{1}{n!} = \frac{1}{n!}\end{aligned}$$

Let π_1' be the neighboring permutation of π_1 such that the two swapped elements correspond to those swapped in π and π'. Similarly, let π_2' be the neighboring permutation of π_2 that also satisfies $\pi_2(\pi_1(x)) = \pi_2'(\pi_1'(x))$. ($\pi_2'$ basically reverse the swap from π_1 to π_1').

As a result, we partition all possible $n!$ pair of permutations for π into $n!/2$ pairs of neighboring pair permutations and rewrite the probability as the following summation:

$$\Pr[\mathbf{O}|\pi] = \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} (\Pr[\mathbf{O}|\pi_1, \pi_2] \Pr[\pi_1, \pi_2|\pi] \tag{4}$$

$$+ \Pr[\mathbf{O}|\pi_1', \pi_2'] \Pr[\pi_1', \pi_2'|\pi]) \tag{5}$$

$$= \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} \frac{1}{n!} (\Pr[\mathbf{O}|\pi_1, \pi_2] + \Pr[\mathbf{O}|\pi_1', \pi_2']) \tag{6}$$

$$= \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} \sum_{\forall o \in \mathbf{O}} \frac{1}{n!} (Pr[o|\pi_1, \pi_2] + \Pr[o|\pi_1', \pi_2']) \tag{7}$$

Similarly, $\Pr[\mathbf{O}|\pi']$ can be written as:

$$\Pr[\mathbf{O}|\pi'] = \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} \frac{1}{n!} (\Pr[\mathbf{O}|\pi_1', \pi_2] + \Pr[\mathbf{O}|\pi_1, \pi_2']) \tag{8}$$

$$= \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} \sum_{\forall o \in \mathbf{O}} \frac{1}{n!} (\Pr[o|\pi_1', \pi_2] + \Pr[o|\pi_1, \pi_2']) \tag{9}$$

Given that each mixing is (ϵ_0, δ_0)-indistinguishable, we have the following inequality for all $\mathbf{O}_1 \subseteq \mathbb{N}^{n\times(d-1)}$ and $\mathbf{O}_2 \subseteq \mathbb{N}^{n\times(d-1)}$:

$$\begin{aligned}\Pr[\mathbf{O}_1|\pi_1] &\leq e^{\epsilon_0} \Pr[\mathbf{O}_1|\pi_1'] + \delta_0 \\ \Pr[\mathbf{O}_2|\pi_2] &\leq e^{\epsilon_0} \Pr[\mathbf{O}_2|\pi_2'] + \delta_0\end{aligned}$$

For any $o_1 \in \mathbb{N}^{n\times(d-1)}$, let $\mu(o_1) = \max(\Pr[o_1|\pi_1] - e^{\epsilon_0} \Pr[o_1|\pi_1'], 0)$ and notice that $\sum_{o_1 \in \mathbb{N}^{n\times(d-1)}} \mu(o_1) \leq \delta_0$ follows from the indistinguishability definition. Given a fixed $\mathbf{O}$, let $\mathbf{O}_1$ denote the set of o_1 such that there exist some o_2 and $(o_1, o_2) \in \mathbf{O}$. Let $\mathbf{O}_2(o_1)$ denote the set of all o_2 such that $(o_1, o_2) \in \mathbf{O}$. using the newly defined notations, we have:

$$\begin{aligned}(4) = &\sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} \sum_{\forall o_1 \in \mathbf{O}_1} \frac{1}{n!} (\Pr[o_1|\pi_1] \Pr[\mathbf{O}_2(o_1)|\pi_2] \\ &+ \Pr[o_1|\pi_1'] \Pr[\mathbf{O}_2(o_1)|\pi_2'] \\ (8) = &\sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} \sum_{\forall o_1 \in \mathbf{O}_1} \frac{1}{n!} (\Pr[o_1|\pi_1'] \Pr[\mathbf{O}_2(o_1)|\pi_2] \\ &+ \Pr[o_1|\pi_1] \Pr[\mathbf{O}_2(o_1)|\pi_2']\end{aligned}$$

For any quadruple $\pi_1, \pi_1', \pi_2, \pi_2'$, we substitute the four conditional probabilities with simple notations: let p, q, p', q' denotes $\Pr[o_1|\pi_1]$, $\Pr[\mathbf{O}_2(o_1)|\pi_2]$, $\Pr[o_1|\pi_1']$, $\Pr[\mathbf{O}_2(o_1)|\pi_2']$ respectively. Without loss of generality, we assume $p \geq p'$ and $q \geq q'$. Clearly, $pq + p'q' \geq pq' + p'q$. Hence, (8) $\leq$ (4). Moreover, let $p = e^{\epsilon_1(o_1)}p' + \delta_1(o_1)$ and $q = e^{\epsilon_2(\mathbf{O}_2(o_1))}q' + \delta_2(\mathbf{O}_2(o_1))$. Here we abuse the notations of ϵ and δ and let them denote functions to specify the exact multiplicative and additive differences for each case of o_1 and $\mathbf{O}_2(o_1)$. Note that according to the indistinguishable definitions, all $\epsilon_1, \epsilon_2 \leq \epsilon_0$ and $\delta_1, \delta_2 \leq \delta_0$. We can substitute and rewrite as follows by neglecting the function parenthesis for legibility reasons:

$$\begin{aligned}(4) = &\frac{1}{n!} \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} \sum_{\forall o_1 \in \mathbf{O}_1} ((e^{\epsilon_1 + \epsilon_2} + 1)p'q' + \delta_1\delta_2 \\ &+ e^{\epsilon_2} q'\delta_1 + e^{\epsilon_1} p'\delta_2) \\ (8) = &\frac{1}{n!} \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} \sum_{\forall o_1 \in \mathbf{O}_1} ((e^{\epsilon_2} + e^{\epsilon_1})p'q' + q'\delta_1 + p'\delta_2)\end{aligned}$$

Notice that the ratio between the $e^{\epsilon_1+\epsilon_2}+1$ and $e^{\epsilon_1}+e^{\epsilon_2}$ can be maximized when $\epsilon_1 = \epsilon_2 = \epsilon_0$ as the partial derivative are non-negative for $\epsilon_1, \epsilon_2 \geq 0$. We use this to relax both equations and multiply (8) with term $\frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}}$ to match (4) :

$$(4) \le \frac{1}{n!} \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} \sum_{\forall o_1 \in \mathbf{O}_1} ((e^{2\epsilon_0}+1)p'q' + \delta_1\delta_2 + e^{\epsilon_0}q'\delta_1 + e^{\epsilon_0}p'\delta_2)$$

$$\frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}} \cdot (8) \le \frac{1}{n!} \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} \sum_{\forall o_1 \in \mathbf{O}_1} ((e^{2\epsilon_0}+1)p'q' + \frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}} \cdot q'\delta_1 + \frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}} \cdot p'\delta_2)$$

Now, the additive gap is maximized when all $\delta_1(o_1) = \mu(o_1), \delta_2(\mathbf{O}_2(o_1)) = \delta_0$. Let $s' = \sum_{\forall o_1 \in \mathbf{O}_1} p'$ and separate the additive terms, we can rewrite them as:

$$(4) = \frac{1}{n!} \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} (e^{2\epsilon_0}+1)s'q' + \frac{1}{n!} \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} (\delta_0^2 + e^{\epsilon_0}\delta_0(s'+q'))$$

$$\frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}} \cdot (8) = \frac{1}{n!} \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} (e^{2\epsilon_0}+1)s'q' + \frac{1}{n!} \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} (\frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}}\delta_0(s'+q'))$$

Notice the the summation is over $n!/2$ quadruple of $\pi_1, \pi_1', \pi_2, \pi_2'$, and the additive difference between this two terms can be calculated as:

$$\begin{aligned}
&(4) - \frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}} \cdot (8) \\
&= \frac{1}{n!} \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} ((\delta_0^2 + (e^{\epsilon_0} - \frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}})\delta_0(s'+q')) \\
&= \frac{\delta_0^2}{2} + (e^{\epsilon_0} - \frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}})\delta_0 \cdot \frac{1}{n!} \sum_{\forall \pi_1, \pi_1', \pi_2, \pi_2'} (s'+q') \\
&\le (e^{\epsilon_0} - \frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}} + \frac{\delta_0}{2}) \cdot \delta_0
\end{aligned}$$

Let $e^{\epsilon} = \frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}}$ and $\delta = (e^{\epsilon_0} - \frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}} + \frac{\delta_0}{2}) \cdot \delta_0$. Specifically, consider the case if we want to roughly half the ϵ_0 and δ_0, when $\epsilon_0 \le 1.219$, $e^{\epsilon} \le e^{\epsilon_0/2}$. Also when $\epsilon_0 \le 0.481$, $e^{\epsilon_0} - \frac{e^{2\epsilon_0}+1}{2e^{\epsilon_0}} \le 0.5$ and given that δ_0 is small, the last term is easily compensated by setting the requirement of ϵ_0 a little smaller. □

The result shows that two consecutive indistinguishable mixings are equivalent to one indistinguishable mixing with strictly no worse privacy parameters. Concretely, if the original ϵ_0 of individual shuffle has $\epsilon_0 < 0.481$, we can reduce both ϵ and δ by at least half. Additionally, our result also generalizes to t iterated mixing through recursively applying the theorem, given that t is a power of 2.

H Noise Mechanisms for Multi Layer Butterfly

Empirically, to keep the same privacy parameter, the noise magnitude μ_m roughly scales linearly with the sensitivity, which in turns scales linearly with the number of intermediate layers. In particular, for a 2-ary butterfly network, $\mu_m = O(\log n)$. As a result, the total computation cost is $O(n \log^3 n)$, and the total communication cost is $O(n \log^2 n \log\log n)$ with the assumption of log-size signature. Notice while the computation cost is asymptotic smaller compared to one-layer case, in practice it is less efficient in the range of n we care about (no more than hundreds of thousands). Additionally, the communication cost is strictly worse than the one-layer case, which has an upper bound of $3n \log n$. Intuitively, this is because with log-size anonymous transactions, it is more efficient to have fewer transactions with larger input size, rather than more transactions of smaller input size.