Non-Interactive Anonymous Router[⋆]

Elaine Shi and Ke Wu

Carnegie Mellon University

Abstract. Anonymous routing is one of the most fundamental online privacy problems and has been studied extensively for decades. Almost all known approaches for anonymous routing (e.g., mix-nets, DC-nets, and others) rely on multiple servers or routers to engage in some *interactive* protocol; and anonymity is guaranteed in the *threshold* model, i.e., if one or more of the servers/routers behave honestly.

Departing from all prior approaches, we propose a novel *non-interactive* abstraction called a Non-Interactive Anonymous Router (NIAR), which works even with a *single untrusted router*. In a NIAR scheme, suppose that n senders each want to talk to a distinct receiver. A one-time trusted setup is performed such that each sender obtains a sending key, each receiver obtains a receiving key, and the router receives a *token* that "encrypts" the permutation mapping the senders to receivers. In every time step, each sender can encrypt its message using its sender key, and the router can use its token to convert the n ciphertexts received from the senders to n *transformed ciphertexts*. Each transformed ciphertext is delivered to the corresponding receiver, and the receiver can decrypt the message using its receiver key. Imprecisely speaking, security requires that the untrusted router, even when colluding with a subset of corrupt senders and/or receivers, should not be able to compromise the privacy of honest parties, including who is talking to who, and the message contents.

We show how to construct a communication-efficient NIAR scheme with provable security guarantees based on the standard Decision Linear assumption in suitable bilinear groups. We show that a compelling application of NIAR is to realize a Non-Interactive Anonymous Shuffler (NIAS), where an untrusted server or data analyst can only decrypt a permuted version of the messages coming from n senders where the permutation is hidden. NIAS can be adopted to construct privacy-preserving surveys, differentially private protocols in the shuffle model, and pseudonymous bulletin boards.

Besides this main result, we also describe a variant that achieves fault tolerance when a subset of the senders may crash. Finally, we further explore a paranoid notion of security called full insider protection, and show that if we additionally assume sub-exponentially secure Indistinguishability Obfuscation and as sub-exponentially secure one-way functions, one can construct a NIAR scheme with paranoid security.

[⋆] A full version of the paper can be found online [68].

1 Introduction

The Internet has become a platform that billions of users rely on in their daily lives, and protecting users' online privacy is a significant challenge we face. Anonymous communication systems provide a way for users to communicate without leaking their identities or message contents. There has been several decades of work dedicated to the design, implementation, and deployment of anonymous communication systems [8, 18, 34, 35, 38, 39, 42, 50, 64, 70, 71, 74], and numerous abstractions and techniques have been explored, including mix-nets [8, 18, 34], the Dining Cryptographers' nets [9, 35, 39], onion routing [29, 41, 42, 50], multi-party-computation-based approaches [13], multi-server PIR-write [38, 48, 60], as well as variants/improvements of the above [70, 71, 74]. We refer the readers to several excellent surveys on this rich line of work [40, 44, 69].

To the best of our knowledge, almost all known anonymous routing schemes rely on *multiple* routers or servers to engage in an *interactive protocol*, and moreover, security is guaranteed in the *threshold* model, i.e., assuming that one or more of the routers remain honest. For example, the mix-net family of schemes typically require each router along the way to shuffle the input ciphertexts and remove a layer of encryption; the DC-net family of schemes require multiple parties to engage in a cryptographic protocol, and so on.

Departing from all prior approaches which are interactive and rely on some form of threshold cryptography, we ask the following natural question:

> Can we achieve anonymous routing *non-interactively* on a *single untrusted* router?

1.1 Defining Non-Interactive Anonymous Router (NIAR)

Our first contribution is a conceptual one: we formulate a new abstraction called a non-interactive anonymous router (NIAR). The abstraction is in fact quite natural, and in hindsight, it may even be a little surprising why it has not been considered before.

Non-interactive anonymous router. Imagine that there are n senders and n receivers, and each sender wishes to speak with a distinct receiver. Henceforth let π denote the permutation that maps each sender to its intended receiver, i.e., each sender $i \in [n]$ wants to speak to receiver $\pi(i)$. A NIAR scheme has the following syntax:

- $(\{\mathsf{ek}_i, \mathsf{rk}_i\}_{i\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}(1^\kappa, n, \pi)$: First, we run a one-time trusted setup procedure that takes the security parameter 1^κ, the number of senders/receivers n, and the routing permutation π, and produces a *sender key* ek_i for each sender $i \in [n]$, and a *receiver key* rk_i for each receiver $i \in [n]$. Moreover, the setup procedure also produces a *token* tk for the router which encodes the secret permutation π. Note that the trusted setup can be decentralized using standard multi-party computation techniques.

- $\mathsf{ct}_{i,t} \leftarrow \mathbf{Enc}(\mathsf{ek}_i, x_{i,t}, t)$: With this one-time setup, we can allow the n senders to anonymously send T number of packets to their intended receivers. In every time step $t \in [T]$, each sender $i \in [n]$ encrypts its message $x_{i,t}$ using its secret key ek_i by calling $\mathbf{Enc}(\mathsf{ek}_i, x_{i,t}, t)$, and sends the resulting ciphertext $\mathsf{ct}_{i,t}$ to the router.
- $\{\mathsf{ct}'_{i,t}\}_{i\in[n]} \leftarrow \mathbf{Rte}(\mathsf{tk}, \{\mathsf{ct}_{i,t}\}_{i\in[n]})$: The untrusted router uses its token to convert the n ciphertexts collected from the senders into n *transformed ciphertexts*. This is accomplished by calling $\mathbf{Rte}(\mathsf{tk}, \{\mathsf{ct}_{i,t}\}_{i\in[n]})$. The router then forwards each transformed ciphertext $\mathsf{ct}'_{i,t}$ to the corresponding recipient i.
- $x_i \leftarrow \mathbf{Dec}(\mathsf{rk}_i, \mathsf{ct}'_{i,t})$: Finally, the recipients use their respective secret keys to decrypt the plaintexts by calling $\mathbf{Dec}(\mathsf{rk}_i, \mathsf{ct}'_{i,t})$.

At a very high level, we want that the untrusted router learns no information about the routing permutation π as well as the messages exchanged. Moreover, the scheme should offer robustness even when a (potentially majority) subset of the senders and/or receivers collude with the untrusted router. It turns out that defining robustness under collusion is non-trivial and the security requirements can vary from application to application — we will discuss the security definitions in more detail later.

Communication efficiency. The first naïve idea is to let each sender-receiver pair share a freshly and randomly chosen secret key during the setup. During each time step, each sender encrypts its messages using its secret key, and sends the ciphertext to the router. The router then forwards all n ciphertexts to each of the n receivers; and each receiver's secret key allows it to decrypt exactly one among the n ciphertexts received. This scheme protects the plaintext messages as well as the routing permutation π from the untrusted router; unfortunately, it incurs *quadratic* communication overhead in each time step[1].

Throughout the rest of the paper, we will require that the NIAR scheme preserve *communication efficiency*, that is, the communication blowup relative to sending the messages in the clear must be upper bounded by $\mathsf{poly}(\kappa)$ where κ is the security parameter. In other words, suppose, without loss of generality, that in each time step, each sender has one bit to send, then the total communication (among all senders and receivers) per time step must be upper bounded by $O(n) \cdot \mathsf{poly}(\kappa)$.

Non-interactive anonymous shuffler. One important application and special case of NIAR is to realize a non-interactive anonymous shuffler (NIAS). To understand what is a non-interactive anonymous shuffler, it helps to think of the following application. Suppose that during a pandemic, University X wants to implement a privacy-preserving daily check mechanism, where students and faculty each send a short message to report their health conditions every day, and whether they could have been exposed to the virus. To protect each individual's

[1] Furthermore, while this naïve scheme works for a private-messaging scenario, and does not work for the non-interactive anonymous shuffler application to be described later, due to the fact that a receiver colluding with the router can learn which sender it is paired with. We will elaborate on this point when we define security.

privacy, we want to shuffle the messages according to some randomly chosen permutation π, such that the history of an individual's reports is pseudonymous. In this scenario, we can employ a NIAR scheme, and give the data analyst the token tk as well as *all* n receiver keys. This ensures that the data analyst can decrypt only a shuffled list of the plaintexts, and moreover the permutation is hidden from the data analyst.

In other words, a Non-Interactive Anonymous Shuffler (NIAS) is a special case of NIAR where the router and all the receivers are a single party. In Section 1.4, we will present numerous applications of NIAR and NIAS. We point out that the NIAS special case in fact imposes some extra security requirements on top of our basic security notion for NIAR, in the sense that even a receiver cannot know which sender it is paired up with — we will discuss how to define security next.

1.2 Defining Security Requirements

If all receivers were fully trusted, then another naïve idea would be to have every sender encrypt its message along with its respective destination using a Fully Homomorphic Encryption (FHE) scheme. In this way, the untrusted router can accomplish the routing through homomorphic evaluation. However, all receivers must be given the FHE's secret key to decrypt the messages. Therefore, if even a single receiver colludes with the untrusted router, then all other honest players' anonymity would be broken. This is clearly unacceptable since in most applications of anonymous routing, anyone can become a sender or a receiver, including the owner of the router. Approaches that construct special-purpose homomorphic encryption schemes optimized for shuffling suffer from the same drawback [11].

We therefore require a security notion that provides robustness even when a subset of the senders and receivers can be corrupt, and potentially colluding with the untrusted router. It turns out that how to define robustness against collusion requires some careful thinking, since the security requirements can vary from application to application.

Basic notion. Our basic security notion is motivated by a private-messaging scenario, e.g., members of a secret society wish to send private emails without revealing their identities and their correspondence to the public. In this case, each player (i.e., either sender or receiver) knows who it is talking to. Therefore, if the adversary who controls the router additionally corrupts a subset of the senders and receivers, the adversary can learn who the corrupt senders and receivers are paired up with, as well as the messages received by corrupt receivers (from honest senders) in every time step. Our basic security notion requires that besides this natural leakage, the adversary should not learn any additional information. Observe that our communication-inefficient naïve solution that forwards all ciphertexts to every receiver would satisfy this basic notion.

Receiver-insider protection. The basic security notion, however, turns out to be insufficient for the NIAS application. In the NIAS application, a single entity acts as the router and all n receivers — for example, in our earlier "anonymous daily check-in" application, the data analyst has all receiver keys $\{\mathsf{rk}_i\}_{i\in[n]}$ as well as the token tk. To protect the users' pseudonymity, it is important that the data analyst does not learn which decrypted report corresponds to which user. In Section 1.4, we present more applications for NIAS, and all of them have the same security requirement.

We therefore propose a strengthened security notion, called receiver-insider protection, that is suitable and sufficient for NIAS-type applications. Here, we require that even a receiver does not learn which sender it is speaking with; however, a sender may learn which receiver it is speaking with. Now, if the adversary who controls the router additionally corrupts a subset of the senders and receivers, the adversary can learn the *corrupt-to-$*$* part[2] of the permutation π as well as the messages received by corrupt receivers in every time step. Besides this natural leakage, the adversary should not learn anything else.

Full insider protection. While receiver-insider protection seems sufficient for most applications including NIAS, we additionally explore a *paranoid* notion of security. Here, we want that every player has no idea who it is speaking with, including both senders and receivers. Nonetheless, the *corrupt-to-corrupt* part of the permutation is *inherently* leaked to the adversary and this leakage cannot be avoided: since a corrupt sender can always try encrypting some message and check whether any corrupt receiver received the corresponding message. Therefore, our most paranoid notion, which we call *full insider protection*, requires that an adversary controlling the router and a subset of corrupt senders and receivers learns only the corrupt-to-corrupt part of the permutation π, as well as the messages received by corrupt receivers in every time step, but nothing else.

In Section 1.4, we describe more applications of NIAR and NIAS, and at that point, the reader can see how different applications require different notions. Of course, one can always go for the most paranoid notion; but the weaker notions suffice for a wide range of natural applications. Therefore, differentiating between these notions can lead to more efficient constructions.

Equivalence between simulation- and indistinguishability-based notions. Later in the paper, we shall formalize the above security notions using two definitional approaches: *simulation-based* notions and *indistinguishability-based* notions. We then prove that in fact, *each simulation-based notion* (without insider protection, with receiver-insider protection, or with full insider protection) *is equivalent to the corresponding indistinguishability-based notion*. While the simulation-based notion more naturally captures the security requirements we want to express, the indistinguishability-based notions are often easier to work with in proofs.

Remark 1 (NIAR/NIAS requires no network-layer anonymity protection). We point out that whenever a NIAR or NIAS scheme is deployed, one advantage is that *we would no longer need any network-layer anonymity protection* (e.g.,

[2] Here, $*$ denotes a wildcard; thus the corrupt-to-$*$ part of the permutation includes who every corrupt sender is speaking with.

Tor [42] or DC nets [35]). This is in contrast to a vast line of works that leverage cryptographic techniques such as zero-knowledge proofs for anonymity protection, e.g., E-Cash [32, 33], e-voting [10, 62], anonymous credentials [14, 20], ZCash [21], and others [53, 54] — in these cases, an Internet Service Provider controlling the network routers can completely break anonymity despite the cryptographic techniques employed.

1.3 Our Results

Main Construction: NIAR with Receiver-Insider Protection and NIAS To situate our results in context, it helps to first think of the following naïve construction based on a virtual-blackbox (VBB) obfuscator. During setup, we publish the public key pk of a public-key encryption (PKE) scheme, and moreover, we give each sender-receiver pair a symmetric encryption key. During each routing step, each sender uses its symmetric key to encrypt its respective message, resulting what we henceforth call an *inner ciphertext*. The sender then encrypts the inner ciphertext with the public key encryption scheme, resulting in an *outer ciphertext*. During the setup, we give the router a VBB obfuscation of the following program: use the PKE's secret key to decrypt each sender's outer ciphertext, obtain a list of n inner ciphertexts, and then apply the permutation π to the n inner ciphertexts and output the result. Now, during each routing step, the router can simply apply its VBB obfuscated program to the list of n outer ciphertexts collected from the senders, and the result would be n permuted inner ciphertexts. The i-th inner ciphertext is then forwarded to the i-th receiver where $i \in [n]$. Note that in this VBB-based solution, the program obfuscation hides the secret key of the PKE scheme as well as the secret permutation π. One can verify that indeed, this VBB-based construction satisfies security with *receiver-insider protection*; but it does *not* provide full insider protection. Specifically, a corrupt sender $i^* \in [n]$ colluding with the router can simply plant a random inner ciphertext c, and see which of the receivers receives c at the end — this must be the receiver i^* is speaking with[3].

The drawback with this naïve solution is obvious: it is well-known that VBB obfuscation is impossible to attain for general functions if one-way functions exist [17]. We therefore ask,

Can we construct a NIAR scheme from standard cryptographic assumptions?

We construct a NIAR scheme that achieves security with receiver-insider protection, relying on the Decisional Linear assumption in suitable bilinear groups. Our scheme satisfies communication efficiency: in each time step, each player sends or receives only $\mathsf{poly}(\kappa)$ bits of data (assuming, without loss of generality, that each sender wants to send one bit during each time step). Furthermore, the

[3] In general, achieving full insider security appears much more challenging than our basic notion or receiver-only insider protection. Indeed, we will discuss this in further detail later on.

public and secret key sizes are $\mathsf{poly}(n, \kappa)$; and yet the scheme can support an *unbounded* number of time steps.

At a high level, in our construction, each sender creates an inner encryption of its message using a symmetric key shared with its receiver, and then encrypts the inner ciphertext again using a special outer encryption scheme. With an appropriately constructed token, the router can output a permuted list of inner ciphertexts. We state the aforementioned result in the following theorem:

Theorem 1 (NIAR with receiver-insider protection). *Assume that the Decisional Linear assumption holds in certain bilinear groups. Then, there exists a NIAR scheme with receiver-insider protection, where the public key and secret key sizes are at most* $\mathsf{poly}(n, \kappa)$ *bits, and the per-player communication cost in each routing step is only* $\mathsf{poly}(\kappa)$ *assuming that each sender has one bit to send per time step. Further, the scheme supports an unbounded number of time steps.*

The above theorem also implies a NIAS scheme with the same performance bounds. Although our work should primarily be viewed as an initial exploration of NIAR, the constructions that led to Theorem 1 is potentially implementable.

NIAR with Full Insider Protection The receiver-insider protection achieved by Theorem 1 is sufficient for most application scenarios including NIAS. Nonetheless, it is interesting to ask whether one can achieve full insider protection. As mentioned, full insider protection is the strongest security notion one can hope for in the context of NIAR, since here we leak only the inevitable. Achieving full insider security, however, appears much more challenging. The reason is that we do not even want a corrupt sender to learn which honest receiver it is talking to. However, in our schemes so far (even the aforementioned VBB-based construction), a corrupt sender $i^* \in [n]$ colluding with the router can choose a random inner ciphertext c and just check which receiver receives c. In this way, the adversary can learn the corrupt-to-$*$ part of the permutation π.

Again, it is instructive to first consider how to achieve full insider protection using VBB obfuscation. To achieve such paranoid security, one way is to modify the previous VBB-based scheme such that inside the VBB, we decrypt the n input ciphertexts, permute them, and then *reencrypt* them under the receivers' keys, respectively. To defeat the aforementioned attack, it is important that the reencryption step produces *random* transformed ciphertexts. In fact, one useful insight we can draw here is that for any scheme that provides full insider protection, if the adversary controlling a corrupt sender $\widetilde{i} \in [n]$ switches $\widetilde{i}$'s input ciphertext, the transformed ciphertexts corresponding to *all* receivers output by the **Rte** procedure must all change.

We show how to achieve full insider protection by additionally relying on sub-exponentially secure indistinguishability obfuscation and sub-exponentially secure one-way functions.

Theorem 2 (NIAR with full insider protection). *Assume the existence of sub-exponentially secure indistinguishability obfuscator, sub-exponentially secure* *one-way functions, and that the Decisional Linear assumption (with standard polynomial security) holds in certain bilinear groups. Then, there exists a NIAR scheme with full insider protection, and whose key sizes and communication cost match those of Theorem 1.*

Notably, a flurry of very recent works [27,47,56,73] show that sub-exponentially secure indistinguishability obfuscator can be constructed under a variety of assumptions some of which are considered well-founded.

Extension: Fault-Tolerant NIAR Similar to the line of work on Multi-Client Functional Encryption (MCFE) [2, 37, 37, 51, 66], a drawback with the present formulation is that a single crashed sender can hamper liveness. Basically, the router must collect ciphertexts from all senders in each time step to successfully evaluate the **Rte** procedure. To the best of our knowledge, fault tolerance has been little investigated in this line of work.

We therefore formulate a variation of our basic NIAR abstraction, called *fault-tolerant NIAR*. In a fault-tolerant NIAR, if a subset of the senders have crashed, the remaining set of senders can encrypt their messages in a way that is aware of the set of senders who are known to be still online (henceforth denoted $\mathcal{O}$). Similarly, the router will perform the **Rte** procedure in a way that is aware of $\mathcal{O}$, too. In this way, the router can continue to perform the routing, without being stalled by the crashed senders. Similar to our basic notion, we define receiver-insider protection and full-insider protection for our fault-tolerant NIAR abstraction, and show that the most natural simulation-based and indistinguishability-based notions are equivalent.

We show that our previous NIAR constructions of Theorem 1 and Theorem 2 can be extended to the fault-tolerant setting, and the result is stated in the following theorem.

Theorem 3 (Informal: fault-tolerant NIAR). *Suppose that the Decisional Linear assumption holds in suitable bilinear groups. Then, there exists a fault-tolerant NIAR scheme that leaks only the (corrupt+crashed)-to-$*$ part of the permutation as well as messages received by corrupt receivers, but nothing else (see the the online full version [68] for formal security definitions).*

Suppose that the Decisional Linear assumption (with standard polynomial security) holds in suitable bilinear groups, and assume the existence of sub-exponentially secure indistinguishability obfuscation and one-way functions. Then, there there exists a fault-tolerant NIAR scheme that leaks only the inherent leakage, that is, the (corrupt+crashed)-to-corrupt part of the permutation as well as messages received by corrupt receivers, but nothing else (see the the online full version [68] for formal security definitions).

Furthermore, both schemes achieve the same key sizes and communication efficiency as in Theorem 1.

1.4 Applications of NIAR and NIAS

NIAR adds to the existing suite of primitives [8, 18, 34, 35, 38, 39, 42, 50, 70, 71, 74] that enable anonymous routing. In comparison with prior works, NIAR adopts a different trust model since it does not rely on threshold cryptography. Arguably it also has a somewhat simpler abstraction than most existing primitives, partly due to the non-interactive nature.

We discuss two flavors of applications for NIAR, including 1) using NIAR in private messaging, which is the more classical type of application; and 2) using NIAR as a non-interactive anonymous shuffler (NIAS). We will use these applications to motivate the need for the different security notions, without insider protection, with receiver-insider protection, or with full insider protection. We shall begin with NIAS-type applications since some of these applications are of emerging interest.

Using NIAR as a Non-Interactive Anonymous Shuffler NIAR can serve as *a non-interactive anonymous shuffler* (NIAS), which shuffles n senders' messages in a non-interactive manner, such that the messages become unlinkable to their senders. This allows the senders to publish messages under a pseudonym, and the pseudonymity does not have to rely on the network layer being anonymous. In a non-interactive shuffler type of application, typically a single entity acts as the router and all n receiver — therefore, typically these applications require *receiver-insider* protection. To understand what is a non-interactive anonymous shuffler, it is most instructive to look at some example applications.

Anonymous bulletin board or forum. Imagine that a group of users want to post messages *pseudonymously* to a website every day, e.g., to discuss some sensitive issues. The users act as the NIAR senders and encrypt their messages every day. The server, which acts as both the router and all the receivers in NIAR, decrypts a permuted list of the messages and posts them on the website. In this way, the untrusted server can mix the n senders' messages, and the pseudonymity guarantee need not rely on additional network-layer anonymity protection. In other words, even a powerful attacker controlling all routers in the world as well as the server cannot break the pseudonymity guarantees.

Since the server takes the role of the router and all n receivers, we would need a NIAR scheme that provides *receiver-insider protection*. This way, even when all the receivers are in the control of the adversary, the adversary cannot deanonymize honest senders.

Distributed differential privacy in the shuffle model. There has been a growing appetite for large-scale, privacy-preserving federated learning, especially due to interest and investment from big players such as Google and Facebook. Unlike the classical "central model" where we have a trusted database curator [43], in a federated learning scenario, the data collector is not trusted, and yet it wants to learn interesting statistics and patterns over data collected by multiple users' mobile phones, web browsers, and so on. This model is often referred to the "local model". It is understood that without any additional assumptions and without cryptographic hardness, mechanisms in the local model incur a utility loss [19, 31, 67] that is significantly worse than the central model (given a fixed privacy budget).

Recently, an elegant line of work [15, 16, 23, 36, 45, 49] emerged, and showed that if there exists a shuffler that randomly shuffles the users' input data, then we can design (information-theoretic) distributed differential privacy mechanisms that are often competitive to the central model. This is commonly referred to as the "shuffle model".

NIAR can be potentially employed to implement a shuffler for the shuffle model. In particular, it is suited for a setting like Google's RAPPOR project [46], where data was repeatedly collected from the users' Chrome browsers on a daily basis. In this scenario, the data collector acts as the NIAR router and all the receivers too; therefore, we also need the NIAR scheme to satisfy receiver-insider protection. Again, we do not need network-level anonymity protection.

Privacy-preserving "daily check" during a pandemic. This application was described earlier in this section. We additionally point out an interesting variation of the same application: we can *create the inner layer of encryption using not symmetric-key encryption, but rather, a predicate encryption scheme* [7, 25, 26, 52, 61, 65]. In this way, a data analyst can be granted special tokens that would permit her to decrypt the data, only if some predicate is satisfied over the user's encrypted daily report (e.g., the user has come in contact with an infected person and needs to be quarantined).

Pseudonymous survey systems. Another application is to build a pseudonymous survey system. For example, we can allow students to pseudonymously and regularly post course feedback to an instructor throughout the semester, or ask questions that they would otherwise feel embarrassed to ask. We can also create periodic surveys and allow members of an underrepresented minority group to pseudonymously report if they have been the victims of discrimination or harassment. Similar applications have been considered and implemented in the past [10, 54]. However, in such existing mechanisms [10, 54], the cryptographic protection alone is insufficient, and one must additionally rely on the network layer to be anonymous too. By contrast, with NIAR, we no longer need the network layer to provide anonymity protection.

Other applications. Besides these aforementioned applications, it is also known that a shuffler can lend to the design of light-weight multi-party computation (MPC) protocols [55].

Private Messaging NIAR can also be used to enable private messaging, which is the more traditional application of anonymous routing. We give a few scenarios to motivate the different security requirements.

In the first scenario, we may imagine that members of a secret society wish to send private messages or emails to one another without identified. To do so, pairs of members that wish to communicate regularly can join a NIAR group. In this scenario, each pair of communicating parties know each other's identities, and therefore we only need the basic security notion, i.e., without insider protection.

Another application is to build an anonymous mentor-mentee system, or an anonymous buddy or mutual-support system. For example, some scientists have relied on Slack to provide such functionalities [1], where members can anonymously post questions, and others can anonymously provide advice. Currently, the anonymity guarantee is provided solely by the Slack server. However, one can easily imagine scenarios where trusting a centralized party for anonymity is undesirable. In these cases, we can rely on NIAR to build an anonymous buddy system. Each pair of buddies can regularly engage in conversations to provide mutual support, and the untrusted router (e.g., Slack) cannot learn the communication pattern or the messages being exchanged. Like the earlier mentor-mentee scenario, the buddies themselves may not wish to reveal their identities to each other. Therefore, in this scenario, we would need the NIAR scheme to provide *full insider protection*.

1.5 Open Questions

Partly, our work makes a conceptual contribution since we are the first to define the NIAR and NIAS abstractions. Our work should be viewed as an initial exploration of these natural abstractions, inspired by a fundamental and long-standing online privacy problem. Many open questions arise given our new abstractions, and our work lays the groundwork for further exploring exciting future directions. We present a list of open questions in the the online full version [68].

1.6 Technical Highlight

Why existing approaches fail. A first strawman attempt is to rely on a Multi-Client Functional Encryption (MCFE) scheme for inner products, also known as Multi-Client Inner-Product Encryption (MCIPE) [2, 4, 37, 51]. In a Multi-Client Inner-Product Encryption scheme, each of the n clients obtains a secret encryption key during a setup phase. During every time step t, each client i uses its secret encryption key to encrypt a message $x_{i,t}$ — henceforth the i-th ciphertext is denoted $\mathsf{ct}_{i,t}$ for $i \in [n]$, and moreover, let $\mathbf{x}_t := (x_{1,t}, \ldots, x_{n,t})$. An authority with a master secret key can generate a functional key $\mathsf{sk}_{\mathbf{y}}$ for a vector $\mathbf{y}$ whose length is also n. Given the collection of ciphertexts $\{\mathsf{ct}_{i,t}\}_{i\in[n]}$ and the functional key $\mathsf{sk}_{\mathbf{y}}$, one can evaluate the function $\langle \mathbf{x}_t, \mathbf{y} \rangle$ of the encrypted plaintexts but nothing else is revealed.

Our idea is to express the permutation π as n selection vectors, and each is used to select what one receiver would receive from the vector of input messages. The router receives one functional key for each selection vector. A selection vector $\mathbf{y}$ has exactly one coordinate that is set to 1, whereas all other coordinates are set to 0. In this way, the inner product of $\mathbf{x}_t$ and $\mathbf{y}$ selects exactly one coordinate of $\mathbf{x}_t$. In our NIAR construction, the input messages $\mathbf{x}_t$ to the MCFE-for-selection scheme will be inner ciphertexts encrypted under keys shared between each pair of sender and receiver, such that the router cannot see to the plaintext message.

At first sight, an MCFE scheme for inner products may seem like a good match for our problem, but upon more careful examination, all known MCFE schemes, including those based on program obfuscation, fail in our context. To the best of our knowledge, *all existing MCFE schemes* (for evaluating any function, not just inner products) *are NOT function-hiding*. In our context, this means that the functional key $\mathsf{sk}_{\mathbf{y}}$ is allowed to reveal the selection vector $\mathbf{y}$. This unfortunately means that the token could leak the routing permutation π and thus violate anonymity. Not only so, in fact, it appears that *no prior work has attempted to define or construct function-hiding MCFE* [2,4,37,57,66], likely because we currently lack techniques to get function privacy for MCFE schemes, even allowing RO and program obfuscation [51]. The known techniques for upgrading Functional Encryption and Multi-Input Functional Encryption to have function privacy [5,22,58,59,63] do not apply to MCFE, because they are fundamentally *incompatible with the scenario where some clients can be corrupt*.

Finally, we point out that a related line of work called *Multi-Input* Inner-Product Encryption [5,6,28,51] also fails to solve our problem, because its security definition is too permissive: specifically, mix-and-matching ciphertexts from multiple time steps is allowed during evaluation, and this could be exploited by an adversary to break anonymity in our context.

Key insights and roadmap. We are the first to define function-hiding MCFE, and demonstrate a construction for a meaningful functionality, i.e., *selection*. Selection is a special case of inner product computation, and is structurally simpler than inner product. Leveraging this structural simplicity, we develop new construction and proof techniques that allow us to prove function-hiding security even when some of the clients can be corrupted. We use the resulting "function-hiding MCFE for selection" as a core building block to realize NIAR.

At a very high level, the structural simplicity of selection helps us in the following way. First, in a more general MCIPE scheme, even without function privacy, one must prevent mix-and-match attacks — in other words, the adversary should not be able to take clients' ciphertext from different time steps and combine them in the same inner-product evaluation. When it comes to the special case of selection, however, we can *defer* the handling of such mix-and-match attacks. Specifically, if we were not concerned about function privacy, then mix-and-match attacks turned out to be a non-issue in an MCFE-for-selection scheme. With this observation, we first construct a conceptually simple MCFE-for-selection scheme *without* function privacy. Essentially, the construction runs n independent instances of semantically secure public-key encryption (PKE), one for each client. The functional key for selecting one client's plaintext is simply the corresponding PKE's secret key.

Next, we perform a function-privacy upgrade — during this function-privacy upgrade, we do need to take care and prevent the aforementioned *mix-and-match* attacks. The function-privacy upgrade is technically much more involved, and we will give an informal overview in Section 3. What lends to the function-privacy upgrade is the fact that the underlying MCFE scheme (without function privacy) is essentially "decomposable" into n independent components. This is an important reason why we can accomplish the function privacy upgrade *even when some of the clients can be corrupted.* In comparison, prior MCFE schemes for general inner-products [2,4,37] need more structurally complicated techniques to prevent mix-and-match, even without function privacy. For this reason, our techniques in the current form are not capable of getting a *function-private* MCFE scheme for general inner-products — this remains a challenging open question.

Once we construct a function-hiding MCFE-for-selection scheme, we then use it to construct two NIAR schemes: one with receiver-insider protection, and one with full insider protection. The scheme with receiver-insider protection can be constructed without introducing additional assumptions — and this notion of security suffices for most applications including NIAS. As explained in Section 1.3, full insider protection seems much more challenging and a natural class of approaches fail. To get a paranoid construction with full insider protection, we additionally rely on sub-exponentially secure indistinguishability obfuscation and sub-exponentially one-way functions.

2 New Definitions: Non-Interactive Anonymous Router

We now define the syntax and security requirements of NIAR. Since our approach relies on a single untrusted router and is non-interactive, both the syntax and security definitions are incomparable to the formal definitions of anonymous routing in prior works, all of which involve multiple routers and interactive protocols [13,29,41].

2.1 Syntax

Suppose that there are n senders and n receivers, and each sender wants to talk to a distinct receiver. They would like to route their messages anonymously to hide who is talking to who. The routing is performed by a single router non-interactively.

Let $\mathsf{Perm}([n])$ denote the set of all permutations on the set $[n]$. Let $\pi \in \mathsf{Perm}([n])$ be a permutation that represents the mapping between the sender and the receivers. For example, $\pi(1) = 3$ means that sender 1 wants to talk to receiver 3.

A Non-Interactive Anonymous Router (NIAR) is a cryptographic scheme consisting of the following, possibly randomized algorithms:

- $(\{\mathsf{ek}_i\}_{i\in[n]}, \{\mathsf{rk}_i\}_{i\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}(1^\kappa, n, \pi)$: the trusted **Setup** algorithm takes the security parameter 1^κ, the number of senders/receivers n, and a permuation $\pi \in \mathsf{Perm}([n])$ that represents the mapping between the senders and the receivers. The **Setup** algorithm outputs a sender key for each sender denoted $\{\mathsf{ek}_i\}_{i\in[n]}$, a receiver key for each receiver denoted $\{\mathsf{rk}_i\}_{i\in[n]}$, and a token for the router denoted tk.

- $\mathsf{ct}_{i,t} \leftarrow \mathbf{Enc}(\mathsf{ek}_i, x_{i,t}, t)$: sender i uses its sender key ek_i to encrypt the message $x_{i,t}$ where $t \in \mathbb{N}$ denotes the current time step. The **Enc** algorithm produces a ciphertext $\mathsf{ct}_{i,t}$.
- $(\mathsf{ct}'_{1,t}, \mathsf{ct}'_{2,t}, \ldots, \mathsf{ct}'_{n,t}) \leftarrow \mathbf{Rte}(\mathsf{tk}, \mathsf{ct}_{1,t}, \mathsf{ct}_{2,t}, \ldots, \mathsf{ct}_{n,t})$: the routing algorithm **Rte** takes its token tk (which encodes some permutation π), and n ciphertexts received from the n senders denoted $\mathsf{ct}_{1,t}, \mathsf{ct}_{2,t}, \ldots, \mathsf{ct}_{n,t}$, and produces *transformed ciphertexts* $\mathsf{ct}'_{1,t}, \mathsf{ct}'_{2,t}, \ldots, \mathsf{ct}'_{n,t}$ where $\mathsf{ct}'_{i,t}$ is destined for the receiver $i \in [n]$.
- $x \leftarrow \mathbf{Dec}(\mathsf{rk}_i, \mathsf{ct}'_{i,t})$: the decryption algorithm **Dec** takes a receiver key rk_i, a transformed ciphertext $\mathsf{ct}'_{i,t}$, and outputs a decrypted message x.

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

Correctness. Without loss of generality, we may assume that each plaintext message is a single bit — if the plaintext contains multiple bits, we can always split it bit by bit and encrypt it over multiple time steps. Correctness requires that with probability 1, the following holds for any $\kappa \in \mathbb{N}$, any $(x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ and any $t \in \mathbb{N}$: let $(\{\mathsf{ek}_i\}_{i\in[n]}, \{\mathsf{rk}_i\}_{i\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}(1^\kappa, n, \pi)$, let $\mathsf{ct}_{i,t} \leftarrow \mathbf{Enc}(\mathsf{ek}_i, x_i, t)$ for $i \in [n]$, let $(\mathsf{ct}'_{1,t}, \mathsf{ct}'_{2,t}, \ldots, \mathsf{ct}'_{n,t}) \leftarrow \mathbf{Rte}(\mathsf{tk}, \mathsf{ct}_{1,t}, \mathsf{ct}_{2,t}, \ldots, \mathsf{ct}_{n,t})$, and let $x'_i \leftarrow \mathbf{Dec}(\mathsf{rk}_i, \mathsf{ct}'_{i,t})$ for $i \in [n]$; it must be that

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

Communication compactness. We require our NIAR scheme to have *compact communication*, that is, the total communication cost per time step should be upper bounded by $\mathsf{poly}(\kappa) \cdot O(n)$. Furthermore, we would like that the token tk, and every sender and receiver's secret key ek_i and rk_i respectively, are all upper bounded by a fixed polynomial in n.

2.2 Simulation-Based Security

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

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

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

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

- $n, \pi, \mathcal{K}_S, \mathcal{K}_R \leftarrow \mathcal{A}(1^\kappa)$
- $(\{\mathsf{ek}_i\}_{i\in[n]}, \{\mathsf{rk}_i\}_{i\in[n]}, \mathsf{tk}) \leftarrow \mathsf{Sim}(1^\kappa, n, \mathcal{K}_S, \mathcal{K}_R, \mathsf{Leak}(\pi, \mathcal{K}_S, \mathcal{K}_R))$
- For $t = 1, 2, \ldots$:
  - if $t = 1$ then $\{x_{i,t}\}_{i\in\mathcal{H}_S} \leftarrow \mathcal{A}(\mathsf{tk}, \{\mathsf{ek}_i\}_{i\in\mathcal{K}_S}, \{\mathsf{rk}_i\}_{i\in\mathcal{K}_R})$; else $\{x_{i,t}\}_{i\in\mathcal{H}_S} \leftarrow \mathcal{A}(\{\mathsf{ct}_{i,t-1}\}_{i\in\mathcal{H}_S})$;
  - $\{\mathsf{ct}_{i,t}\}_{i\in\mathcal{H}_S} \leftarrow \mathsf{Sim}\left(\{\forall i \in \mathcal{K}_R \cap \pi(\mathcal{H}_S) : (i, x_{j,t}) \text{ for } j = \pi^{-1}(i)\}\right)$. In other words, the simulator Sim is allowed to see for each corrupt receiver talking to an honest sender, what message it receives.

Defining the insider information $\mathsf{Leak}(\pi, \mathcal{K}_S, \mathcal{K}_R)$ known to corrupt players. We require that *when no sender or receiver is corrupt, the adversary should not learn anything about the routing permutation* π. When some senders and receivers are corrupt, the adversary *may learn the insider information about* π *known to the corrupt players, but nothing else.* We use the function $\mathsf{Leak}(\pi, \mathcal{K}_S, \mathcal{K}_R)$ to describe the insider information known to corrupt senders and receivers about the routing permutation π. We define three natural notions of insider information:

1. *Every player knows who it is talking to.* The first natural notion is to assume that each sender or receiver knows whom the player itself is talking to, but it is not aware who others are talking to. By corrupting some senders and receivers, the adversary should not learn more about the routing permutation π beyond what the corrupt senders and receivers know. In other words, the part of the permutation π containing "corrupt $\rightarrow *$" and "$* \rightarrow$ corrupt" is leaked. More formally, we can define leakage as below:
$$\mathsf{Leak}^{\mathrm{SR}}(\pi, \mathcal{K}_S, \mathcal{K}_R) := \{\forall i \in \mathcal{K}_S : (i, \pi(i))\} \cup \{\forall i \in \mathcal{K}_R : (\pi^{-1}(i), i)\}$$
2. *Every sender knows who it is talking to.* Another natural notion is when a sender knows which receiver it is talking to, but a receiver may not know who it is receiving from. By corrupting a subset of the senders and receivers, the adversary should not learn more than what those corrupt players know. In other words, the "corrupt $\rightarrow *$" part of the permutation π is leaked. More formally, we can formally define leakage as below:
$$\mathsf{Leak}^{\mathrm{S}}(\pi, \mathcal{K}_S, \mathcal{K}_R) := \{\forall i \in \mathcal{K}_S : (i, \pi(i))\}$$
3. *Inherent leakage.* The least possible leakage is when only the "corrupt $\rightarrow$ corrupt" part of the permutation π is leaked. Note that this leakage is inherent because a corrupt sender can always encrypt multiple random messages in

the same time slot, and observe whether any corrupt receiver received this message. In the minimum, inherent leakage scenario, we require that only this is leaked about the permutation π and nothing else. More formally, we can formally define leakage as below:

$$\mathsf{Leak}^{\min}(\pi, \mathcal{K}_S, \mathcal{K}_R) := \{\forall i \in \mathcal{K}_S \cap \pi^{-1}(\mathcal{K}_R) : (i, \pi(i))\}$$

Remark 2. Note that even in the minimum, inherent leakage scenario, knowing the leaked information $\mathsf{Leak}^{\min}(\pi, \mathcal{K}_S, \mathcal{K}_R) := \{\forall i \in \mathcal{K}_S \cap \pi^{-1}(\mathcal{K}_R) : (i, \pi(i))\}$ as well as $\mathcal{K}_R$, one can efficiently compute the set $\mathcal{K}_R \cap \pi(\mathcal{H}_S)$. Therefore, during the encryption phase, by learning $\{\forall i \in \mathcal{K}_R \cap \pi(\mathcal{H}_S) : (i, x_j) \text{ for } j = \pi^{-1}(i)\}$, i.e., the set of leaked messages received by corrupt receivers from honest senders, the simulator Sim does not learn anything extra about the routing permutation π beyond what it already learned earlier in the experiment, that is, $\mathsf{Leak}^{\min}(\pi, \mathcal{K}_S, \mathcal{K}_R)$.

Definition 1 (NIAR simulation security). *We define simulation security of a NIAR scheme as below depending on which leakage function is used in the* $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}$ *experiment:*

1. *We say that a Non-Interactive Anonymous Routing (NIAR) scheme is SIM-secure iff the following holds when using* $\mathsf{Leak} := \mathsf{Leak}^{\mathrm{SR}}$ *in the* $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}$ *experiment: there exists a p.p.t. simulator* Sim *such that for any non-uniform p.p.t. adversary* $\mathcal{A}$*,* $\mathcal{A}$*'s view in* $\mathsf{Real}^{\mathcal{A}}(1^\kappa)$ *and* $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}(1^\kappa)$ *are computationally indistinguishable.*
2. *We say that a NIAR scheme is SIM-secure with receiver-insider protection, iff the above holds when using* $\mathsf{Leak} := \mathsf{Leak}^{\mathrm{S}}$ *in the* $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}$ *experiment.*
3. *We say that a NIAR scheme is SIM-secure with full insider protection, iff the above holds when using* $\mathsf{Leak} := \mathsf{Leak}^{\min}$ *in the* $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}$ *experiment.*

2.3 Equivalence to Indistinguishability-Based Security

In our online full version [68], we define an alternative, indistinguishability-based security notion, and prove that it is equivalent to the simulation-based notion.

3 Informal Overview of Our Construction

We now give an informal overview of our constructions.

3.1 Notations and Building Block

We will concretely instantiate a scheme using a cyclic group $\mathbb{G}$ of prime order q. Therefore, we introduce some notations for group elements and group operations.

Group notation and implicit notation for group exponentiation. Throughout the paper, we use the notation $[\![x]\!]$ to denote a group element $g^x \in \mathbb{G}$ where $g \in \mathbb{G}$ is the generator of an appropriate cyclic group of prime order q where $x \in \mathbb{Z}_q$. Similarly, $[\![\mathbf{x}]\!]$ denotes a vector of group elements where $\mathbf{x} \in \mathbb{Z}_q^{|\mathbf{x}|}$ is the exponent vector. If we know $[\![x]\!] \in \mathbb{G}$ and $y \in \mathbb{Z}_p$, we can compute $[\![xy]\!] \in \mathbb{G}$. Therefore, whenever an algorithm needs to compute $[\![xy]\!]$, it only needs to know one of the exponents x or y. The same implicit notation is used for vectors too.

Correlated pseudorandom functions. We will need a building block which we call a correlated pseudorandom function, denoted CPRF. A CPRF scheme has the following possibly randomized algorithms:

– $(K_1, K_2, \ldots, K_n) \leftarrow \mathbf{Gen}(1^\kappa, n, q)$: takes a security parameter 1^κ and the number of users n, some prime q, and outputs the user secret key K_i for each $i \in [n]$.
– $v \leftarrow \mathbf{Eval}(K_i, x)$: given a user secret key K_i and an input $x \in \{0,1\}^\kappa$, output an evaluation result $v \in \mathbb{Z}_q$.

For correctness, we require that the following always holds if $\{K_i\}_{i\in[n]}$ is in the support of $\mathbf{Gen}(1^\kappa, n, q)$:

$$\sum_{i\in[n]} \mathsf{CPRF}.\mathbf{Eval}(K_i, x) = 0 \mod q \tag{1}$$

For security, we require that even when a subset of the keys $\mathcal{K} \subset [n]$ can be corrupted by the adversary, it must be that for every fresh x, all honest evaluations $\{\mathsf{CPRF}.\mathbf{Eval}(K_i, x)_{i\notin\mathcal{K}}\}$ are computationally indistinguishable from random terms subject to the constraint $\sum_{i\notin\mathcal{K}} \mathsf{CPRF}.\mathbf{Eval}(K_i, x) = -\sum_{i\in\mathcal{K}} \mathsf{CPRF}.\mathbf{Eval}(K_i, x)$ mod q — note that the adversary can compute the right-hand-side of the equation.

Intuitively, such a correlated PRF guarantees that even when some players' keys can be corrupt, honest players' evaluations for any fresh input x must appear random, except that they are subject to the constraint in Equation 1. A couple earlier works [2, 24] showed how to construct such a CPRF from ordinary PRFs. We will present more formal definitions and construction in the the online full version [68].

3.2 A Simple, Function-Revealing MCFE Scheme for Selection

Multi-client functional encryption for summation was first suggested by Shi et al. [66] (coined "private stream aggregation" in their paper). Later, Goldwasser et al. [51] defined multi-client functional encryption for general functions, and constructed a scheme assuming indistinguishable obfuscation, random oracles, and other assumptions. Subsequently, a line of work focused on constructing MCFE schemes for inner-products.

We consider MCFE for "selection", which can be viewed as a special case of inner-product computation. An MCFE-for-selection scheme has four possibly randomized algorithms (**Setup**, **KGen**, **Enc**, **Dec**) — in our definition below, we allow each client to encrypt a vector $\mathbf{x}_{i,t} \in \{0,1\}^m$ of length m, and the selection vector $\mathbf{y} \in \{0,1\}^{mn}$ selects exactly one coordinate from one client's plaintext vector[4]:

- The $\mathbf{Setup}(1^\kappa, m, n)$ algorithm[5] outputs a secret key for each of the n clients where the i-th client's key is denoted ek_i, and a master public- and secret-key pair $(\mathsf{mpk}, \mathsf{msk})$.
- The $\mathbf{KGen}(\mathsf{mpk}, \mathsf{msk}, \mathbf{y})$ algorithm takes the master public-key mpk and the master secret-key msk, and outputs a functional key $\mathsf{sk}_\mathbf{y}$ for the selection vector $\mathbf{y} \in \{0,1\}^{mn}$. It is promised that the input $\mathbf{y}$ has only one coordinate set to 1, and the rest are set to 0.
- The $\mathbf{Enc}(\mathsf{mpk}, \mathsf{ek}_i, \mathbf{x}_{i,t}, t)$ algorithm lets client $i \in [n]$ use its secret key ek_i to encrypt a plaintext $\mathbf{x}_{i,t} \in \{0,1\}^m$ for the time step t.
- Finally, given the n ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$ collected from all clients pertaining to the same time step, as well as the functional key $\mathsf{sk}_\mathbf{y}$, one can call $\mathbf{Dec}(\mathsf{mpk}, \mathsf{sk}_\mathbf{y}, \{\mathsf{ct}_i\}_{i\in[n]})$ to evaluate the selection outcome $\langle \mathbf{x}, \mathbf{y} \rangle$ where $\mathbf{x}$ denotes the concatenation of the plaintexts encrypted under $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$.

If we did not care about function privacy, it turns out that we can construct a very simple MCFE-for-selection scheme as follows. Basically, for each of the n clients, there is a separate symmetric-key encryption instance. During **Setup**, client i obtains the secret keys $\mathsf{sk}_{i,1}, \ldots, \mathsf{sk}_{i,m}$ corresponding to m independent encryption instances. For client i to encrypt a message of m bits during some time step t, it simply encrypts each bit $j \in [m]$ using $\mathsf{sk}_{i,j}$, and output the union of the ciphertexts. To generate a functional key for selection vector $\mathbf{y}$ that selects the j-th coordinate of the client i's message, simply output $(\mathbf{y}, \mathsf{sk}_{i,j})$, and decryption can be completed, i.e., using $\mathsf{sk}_{i,j}$ to decrypt the coordinate in the ciphertext that is being selected.

3.3 Preparing the MCFE Scheme for Function Privacy Upgrade

The next challenge is how to upgrade the above MCFE-for-selection scheme to have function privacy. Function privacy in inner-product functional encryption (FE) was first studied by Shen, Shi, and Waters [63], who considered single-input FE and a weaker notion of function privacy than what we will need. Subsequent works have generalized and improved the techniques of Shen, Shi, and Waters [63] to achieve stronger notions of function privacy [58], and have extended the techniques to a multi-input FE context [5].

[4] Our scheme can support the case where each coordinate of the plaintext vector $\mathbf{x}_{i,t}$ comes from a polynomially sized space, but we simply assume each coordinate is a bit for simplicity.

[5] In our subsequent formal sections, for notational reasons needed to make our presentation formal, we shall separate the **Setup** algorithm into a parameter generation algorithm **Gen** and a **Setup** algorithm, respectively.

Our function privacy upgrade techniques are inspired by these earlier works [5, 58, 63], but we need non-trivial new techniques to make it work in our context. Specifically, previous function privacy techniques assume the encryptor to be trusted, and thus they are not directly applicable to the MCFE setting in which some of the clients may be corrupted, and their secret keys become known to the adversary.

To enable the function-private upgrade, let us first understand where the above MCFE-for-selection scheme in Section 3.2 leaks information about the selection vector $\mathbf{y}$. First, the scheme blantantly embeds the selection vector $\mathbf{y}$ in cleartext in the functional key $\mathsf{sk}_{\mathbf{y}}$. Second, the decryption process itself also reveals $\mathbf{y}$ because decryption works on only the coordinate being selected. To fix the above problems, we would like to first modify the idea in Section 3.2 to satisfy the following two requirements:

1. We change the decryption process such that decryption involves all coordinates, and not just the coordinate being selected.
2. Further, we want to randomize the partial decryption outcome corresponding to every client such that from the partial decryptions alone, one cannot tell which coordinate is being selected.

We can instantiate an MCFE-for-selection scheme satisfying the above requirements in a cyclic group $\mathbb{G}$ of prime order q. The resulting scheme is still *function-revealing* — at this point, we have merely "prepared" the scheme for the function privacy upgrade described later in Section 3.4. We describe this scheme below where we use $\mathsf{CPRF}(K_i, t)$ as an abbreviation for $\mathsf{CPRF}.\mathbf{Eval}(K_i, t)$:

MCFE: function-revealing MCFE for selection, w/ randomized partial decryptions

$\mathsf{mpk} = [\![w]\!]$, $\mathsf{msk} = \{\mathbf{S}_i, a_i\}_{i\in[n]}$, $\mathsf{ek}_i = (K_i, a_i)$ where each $\mathbf{S}_i \in \mathbb{Z}_q^{m\times 2}$

Ciphertext for t where each $\mathbf{x}_{i,t} \in \{0,1\}^m$

$$\forall i \in [n]: \quad \left([\![\mathbf{x}_{i,t} + \mathbf{S}_i\mathbf{r}_i]\!], \; [\![\mathbf{r}_i]\!], \; [\![\mathsf{CPRF}(K_i,t) + a_i w \mu_i]\!], \; [\![w\mu_i]\!] \right)$$

where $\mathbf{r}_i$ and μ_i are chosen at random

Functional key for $\mathbf{y} := (\mathbf{y}_1, \ldots, \mathbf{y}_n)$ where each $\mathbf{y}_i \in \{0,1\}^m$

$$\forall i \in [n]: \quad \left(\mathbf{y}_i, \quad -\mathbf{S}_i^\top \mathbf{y}_i, \quad \rho, \quad -\rho a_i \right)$$

where ρ is chosen at random

Henceforth, we will name $[\![\mathbf{c}_{i,1}]\!] := [\![\mathbf{x}_{i,t} + \mathbf{S}_i\mathbf{r}_i]\!]$, $[\![\mathbf{c}_{i,2}]\!] := [\![r_i]\!]$, and $[\![\widetilde{\mathbf{c}}]\!] := [\![\mathsf{CPRF}(K_i,t) + a_i w\mu_i, w\mu_i]\!]$. Additionally, let $\mathbf{k}_{i,1} := \mathbf{y}_i$, $\mathbf{k}_{i,2} := \mathbf{S}_i^\top \mathbf{y}_i$, and $\widetilde{\mathbf{k}}_i := (\rho, -\rho a_i)$.

For the above scheme to be a correct function-revealing MCFE-for-selection, we only need the first two terms of the ciphertext and functional keys, i.e., $([\![\mathbf{c}_{i,1}]\!], [\![\mathbf{c}_{i,2}]\!])$ and $(\mathbf{k}_{i,1}, \mathbf{k}_{i,2})$. Essentially, these terms can be viewed as a concrete instantiation of the ideas mentioned in Section 3.2: the j-th row of $\mathbf{S}_i$ is used to encrypt the j-th coordinate of $x_{i,t}$; further, to compute a functional key for $\mathbf{y}$ which is selecting the j-th coordinate of the i-th client's message, simply output $\mathbf{y}$ and the j-th row of $\mathbf{S}_i$ (which is equal to $\mathbf{S}_i^\top \mathbf{y}_i$). Security of the encryption follows from the Decisional Linear assumption. The extra terms in the ciphertexts and functional keys, denoted $\widetilde{\mathbf{c}}_i$ and $\widetilde{\mathbf{k}}_i$ are randomizing terms added to satisfy the aforementioned randomized partial decryption requirement as we explain below.

We now explain how decryption works. Given a ciphertext vector for n all clients $[\![\mathbf{c}]\!] := \big(([\![\mathbf{c}_{1,1}]\!], [\![\mathbf{c}_{1,2}]\!], [\![\widetilde{\mathbf{c}}_1]\!]), \ldots, ([\![\mathbf{c}_{n,1}]\!], [\![\mathbf{c}_{n,2}]\!], [\![\widetilde{\mathbf{c}}_n]\!])\big)$, and a key vector $\mathbf{k} := \big((\mathbf{k}_{1,1}, \mathbf{k}_{1,2}, \widetilde{\mathbf{k}}_1), \ldots, (\mathbf{k}_{n,1}, \mathbf{k}_{n,2}, \widetilde{\mathbf{k}}_n)\big)$, decryption computes the "inner-product-in-the-exponent" of the ciphertext vector and the token vector, i.e.,

$$[\![\langle \mathbf{c}, \mathbf{k} \rangle]\!] = \prod_{i \in [n]} \Big([\![\langle \mathbf{c}_{i,1}, \mathbf{k}_{i,1} \rangle]\!] \cdot [\![\langle \mathbf{c}_{i,2}, \mathbf{k}_{i,2} \rangle]\!] \cdot [\![\langle \widetilde{\mathbf{c}}_i, \widetilde{\mathbf{k}}_i \rangle]\!] \Big).$$

Finally, we output the discrete logarithm of the above expression as the decrypted message[6].

The decryption can alternatively be viewed as computing the partial decryption of each client and then multiplying the partial decryptions together. Henceforth, let $\mathsf{MCFE}.\mathbf{Dec}^i$ denote the function that computes the partial decryption corresponding to client i, and let $p_{i,t}$ denote the i-th partial decryption:

$$p_{i,t} := \mathsf{MCFE}.\mathbf{Dec}^i(\mathsf{sk}_i, \mathsf{ct}_{i,t}) = \Big([\![\langle \mathbf{c}_{i,1}, \mathbf{k}_{i,1} \rangle]\!] \cdot [\![\langle \mathbf{c}_{i,2}, \mathbf{c}_{i,2} \rangle]\!] \cdot [\![\langle \widetilde{\mathbf{c}}_i, \widetilde{\mathbf{k}}_i \rangle]\!] \Big),$$

and then multiplying all the randomized partially decrypted results. Note that the partial decryption function $\mathsf{MCFE}.\mathbf{Dec}^i(\mathsf{sk}_i, \mathsf{ct}_{i,t})$ also evaluates an inner-product in the exponent. One can verify the following: let $\mathbf{x}_{i,t} := (x_{i,1,t}, \ldots, x_{i,m,t})$ be the plaintext message encrypted under $\mathsf{ct}_{i,t}$, we have that

$$p_{i,t} = \begin{cases} [\![\mathsf{CPRF}(K_i, t) \cdot \rho]\!] & \text{if client } i\text{'s vector is not being selected} \\ [\![x_{i,j,t} + \mathsf{CPRF}(K_i, t) \cdot \rho]\!] & \text{if the } j\text{-th coordinate of the } i\text{-th client is being selected} \end{cases}$$

Thus, the above decryption indeed involves all coordinates, and moreover, the partial decryption results $\{p_{i,t}\}_{i \in [n]}$ are randomized due to the use of the CPRF.

Remark 3 (Technical condition needed for the function privacy upgrade). Informally speaking, we want the following (necessary but not sufficient) condition to hold for our function privacy upgrade to work. Let $\mathcal{H} \subseteq [n]$ be the set of honest clients. Assume that the Decisional Linear assumption holds. We want that even after having seen the public key, honest ciphertexts in all time steps other than t, honest ciphertexts in time step t, i.e., $\{\mathsf{ct}_{i,t}\}_{i \in \mathcal{H}}$, as well as $[\![\rho]\!]$ for a fresh random $\rho \in \mathbb{Z}_q$, the terms $\{[\![\mathsf{CPRF}(K_i, t) \cdot \rho]\!]\}_{i \in \mathcal{H}}$ must be computationally indistinguishable from random, except that their product is equal to some fixed term known to the adversary. This condition is needed in the proof of a key lemma in the function privacy upgrade proof (see our the online full version [68]).

[6] Note that because decryption involves computing a discrete logarithm, we require the plaintext space to be small.

3.4 Function Privacy Upgrade

Since we do not want the functional key to leak the selection vector $\mathbf{y}$, we want to encrypt the functional key $\mathsf{sk}_{\mathbf{y}}$; but how can we use the encrypted $\mathsf{sk}_{\mathbf{y}}$ for correct decryption? Inspired by earlier works [5, 58], our idea is to adopt n instances (single-input) functional encryption henceforth denoted FE, such that the i-th client obtains the master secret key of the i-th instance, henceforth denoted msk_i. During **KGen**, we encrypt the the i-th coordinate of $\mathsf{sk}_{\mathbf{y}}$ using the i-th FE, and let the result be $\overline{\mathsf{sk}}_i$. To encrypt its message $\mathbf{x}_{i,t}$, the i-th client first encrypts $\mathbf{x}_{i,t}$ using the MCFE-for-selection scheme and obtains the ciphertext $\mathsf{ct}_{i,t}$; then it calls $\overline{\mathsf{ct}}_{i,t} := \mathsf{FE}.\mathbf{KGen}(\mathsf{msk}_i, f^{\mathsf{ct}_{i,t}})$ to transform $\mathsf{ct}_{i,t}$ into an FE token for the function $f^{\mathsf{ct}_{i,t}}(\star) := \mathsf{MCFE}.\mathbf{Dec}^i(\star, \mathsf{ct}_{i,t})$. Recall that $\mathsf{MCFE}.\mathbf{Dec}^i$ computes the MCFE's partial decryption for the i-th coordinate. In this way, an evaluator can invoke $\mathsf{FE}.\mathbf{Dec}$ on the pair $\overline{\mathsf{ct}}_{i,t}$ and $\overline{\mathsf{sk}}_i$ to obtain the i-th partial decryption.

To make this idea work, in fact, we do not even need FE for general circuits. Recall that in our MCFE-for-selection scheme above, each partial decryption function $\mathsf{MCFE}.\mathbf{Dec}^i$ computes an inner-product in the exponent. We therefore only need an FE scheme capable of computing an inner-product in the exponent. Several earlier works [3, 12, 72] showed how to construct inner-product function encryption based on the DDH assumption. By slightly modifying these constructions, one can construct an FE scheme for evaluating "inner-product-in-the-exponent" as long as the Decisional Linear assumption holds in certain bilinear groups. For completeness, we shall present this special FE scheme for computing "inner-product-in-the-exponent" in the the online full version [68].

From weak to full function privacy. Although intuitively, the above idea seems like it should work, it turns out for technical reasons, we can only prove that it satisfies a weak form of function privacy henceforth called *weak function hiding*. We defer its detailed technical definition to the the online full version [68]. Fortunately, we can borrow a two-slot trick from various prior works on Functional Encryption [5, 22, 63] and Indistinguishability Obfuscation [58, 59], and upgrade a weakly function-hiding MCFE-for-selection scheme to a fully function-hiding one. At a very high level, to achieve this, instead of having each client $i \in [n]$ encrypt its plaintext $\mathbf{x}_i \in \{0,1\}^m$, we have each client i encrypt the expanded vector $(\mathbf{x}_i, \mathbf{0})$ instead where $\mathbf{0}$ is also of length m. Similarly, the selection vector's length will need to be doubled accordingly too, i.e., to compute a functional key for $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$ where each $\mathbf{y}_i \in \{0,1\}^m$, we instead compute a functional key for the expanded vector $((\mathbf{y}_1, \mathbf{0}), \ldots, (\mathbf{y}_n, \mathbf{0}))$.

By expanding the plaintext and selection vectors, we gain some spare slots which can serve as "wiggle room" during our security proofs. This way, in our security proofs, we can make incremental modifications in every step of the hybrid sequence and make progress with the proof.

Our exposition above is geared towards understandability and is sometimes informal. The actual details and proofs are somewhat more involved and we refer the reader to the the online full version [68] for a formal exposition.

Summarizing the above, we can construct an MCFE-for-selection scheme with (full) function privacy, henceforth denoted $\mathsf{MCFE}^{\mathsf{ffh}}$, presented more formally below. In the description below, MCFE is the aforementioned function-revealing MCFE for selection, augmented to have randomized partial decryptions; FE is a single-input functional encryption scheme for computing inner-products in exponents, formally defined in the the online full version [68].

$\mathsf{MCFE}^{\mathrm{ffh}}$: **function-hiding MCFE for selection**

- $\mathbf{Gen}(1^\kappa)$: Sample a suitable prime q, and generate a suitable bilinear group of order q, with the pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let $H : \{0,1\}^* \to \mathbb{G}_1$ be a random oracle. The public parameter pp contains the prime q, and the description of the bilinear group; the parameters pp' contains a description of $\mathbb{G}_1$, its order q, and a description of H.
- $\mathbf{Setup}(\mathsf{pp}, m, n)$: Call $(\mathsf{mpk}', \mathsf{msk}', \{\mathsf{ek}'_i\}_{i\in[n]}) \leftarrow \mathsf{MCFE}.\mathbf{Setup}(\mathsf{pp}', 2m, n)$. For $i \in [n]$, call $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{FE}.\mathbf{Setup}(\mathsf{pp}, 2m+2)$. Output:

$$\mathsf{mpk} := (\mathsf{pp}, \mathsf{mpk}', \{\mathsf{mpk}_i\}_{i\in[n]}), \quad \mathsf{msk} := (\mathsf{msk}', \{\mathsf{msk}_i, \mathsf{ek}_i\}_{i\in[n]}),$$
$$\forall i \in [n] : \mathsf{ek}_i := (\mathsf{msk}_i, \mathsf{ek}'_i)$$

- $\mathbf{Enc}(\mathsf{mpk}, \mathsf{ek}_i, \mathbf{x}, t)$:
  1. Let $\mathsf{ct} := \mathsf{MCFE}.\mathbf{Enc}(\mathsf{mpk}', \mathsf{ek}'_i, (\mathbf{x}, \mathbf{0}), t) \in \mathbb{G}_1^{2m+2}$.
  2. Let $\overline{\mathsf{ct}} := \mathsf{FE}.\mathbf{KGen}(\mathsf{msk}_i, \mathsf{ct})$.
  3. Output $\mathsf{CT} := (\mathsf{ct}, \overline{\mathsf{ct}})$.
- $\mathbf{KGen}(\mathsf{mpk}, \mathsf{msk}, \mathbf{y})$:
  1. Parse $\mathbf{y} := (\mathbf{y}_1, \ldots, \mathbf{y}_n)$ where each $\mathbf{y}_i \in \{0,1\}^m$.
  2. Let $\widetilde{\mathbf{y}} = ((\mathbf{y}_1, \mathbf{0}), \ldots, (\mathbf{y}_n, \mathbf{0})) \in \{0,1\}^{2mn}$.
  3. Call $(\mathbf{k}_1, \ldots, \mathbf{k}_n) := \mathsf{MCFE}.\mathbf{KGen}(\mathsf{mpk}', \mathsf{msk}', \widetilde{\mathbf{y}})$ where each $\mathbf{k}_i \in \mathbb{Z}_q^{2m+2}$ for $i \in [n]$.
  4. For $i \in [n]$, call $\overline{\mathbf{k}}_i := \mathsf{FE}.\mathbf{Enc}(\mathsf{mpk}_i, \mathbf{k}_i)$.
  5. Output $\mathsf{sk}_{\mathbf{y}} := (\overline{\mathbf{k}}_1, \ldots, \overline{\mathbf{k}}_n)$.
- $\mathbf{Dec}(\mathsf{mpk}, \mathsf{sk}_{\mathbf{y}}, \{\mathsf{CT}_i\}_{i\in[n]})$: Parse each $\mathsf{CT}_i := (\mathsf{ct}_i, \overline{\mathsf{ct}}_i)$. Parse $\mathsf{sk}_{\mathbf{y}} := (\overline{\mathbf{k}}_1, \ldots, \overline{\mathbf{k}}_n)$. For $i \in [n]$, call $v_i := \mathsf{FE}.\mathbf{Dec}(\overline{\mathsf{ct}}_i, \mathsf{ct}_i, \overline{\mathbf{k}}_i)$. Output $\log(\prod_{i=1}^n v_i)$.

Our $\mathsf{MCFE}^{\mathrm{ffh}}$ scheme will be at the core of both our NIAR schemes, the one with receiver-insider protection, and the one with full insider protection.

Proof roadmap for $\mathsf{MCFE}^{\mathrm{ffh}}$. To prove our $\mathsf{MCFE}^{\mathrm{ffh}}$ scheme secure, a critical stepping stone is to prove that it satisfies a weak notion of function privacy — afterwards we can rely on known techniques [5, 22, 58, 59] to prove full function privacy. Roughly speaking, we say that an MCFE scheme for selection satisfies weak function privacy iff no p.p.t. *admissible* adversary $\mathcal{A}$ can distinguish two worlds indexed by $b \in \{0,1\}$. In world b:

- the adversary $\mathcal{A}$ first specifies a set of corrupt clients, and obtains the public parameters as well as secret keys for corrupt clients;
- the adversary $\mathcal{A}$ now submits multiple **KGen** queries, each time specifying $\mathbf{y}^{(0)}$ and $\mathbf{y}^{(1)}$; and the challenger computes and returns tokens for $\mathbf{y}^{(b)}$;

– then $\mathcal{A}$ makes **Enc** queries for each time step t by specifying $\{\mathbf{x}_{i,t}^{(0)}\}_{i\in\mathcal{H}}$ and $\{\mathbf{x}_{i,t}^{(1)}\}_{i\in\mathcal{H}}$ where $\mathcal{H} \subseteq [n]$ denotes the set of honest clients; and the challenger computes and returns encryptions for $\{\mathbf{x}_{i,t}^{(b)}\}_{i\in\mathcal{H}}$.

Moreover, an *admissible* adversary $\mathcal{A}$ must respect the following constraints:

1. for $i \in [n]\backslash\mathcal{H}$, $\mathbf{y}_i^{(0)} = \mathbf{y}_i^{(1)}$.
2. for any $\{\mathbf{x}_{i,t}^{(0)}, \mathbf{x}_{i,t}^{(1)}\}_{i\in\mathcal{H}}$ submitted in an **Enc** query,

$$\left\langle (\mathbf{x}_{i,t}^{(0)})_{i\in\mathcal{H}}, (\mathbf{y}_i^{(0)})_{i\in\mathcal{H}} \right\rangle = \left\langle (\mathbf{x}_{i,t}^{(1)})_{i\in\mathcal{H}}, (\mathbf{y}_i^{(0)})_{i\in\mathcal{H}} \right\rangle = \left\langle (\mathbf{x}_{i,t}^{(1)})_{i\in\mathcal{H}}, (\mathbf{y}_i^{(1)})_{i\in\mathcal{H}} \right\rangle$$

In our proof, we start from world 0, and through a sequence of hybrids, we switch to world 1; and every adjacent pair of hybrids are computationally indistinguishable. First, we use the function-revealing privacy of the underlying MCFE scheme to switch the encrypted vectors from $\{\mathbf{x}_{i,t}^{(0)}\}_{i\in\mathcal{H}}$ to $\{\mathbf{x}_{i,t}^{(1)}\}_{i\in\mathcal{H}}$ — this step is possible due to the aforementioned admissibility rule $\mathcal{A}$ must respect. Next, we want to switch to using $\mathbf{y}^{(1)}$ in each **KGen** query. To accomplish this, we rely on a hybrid sequence over the multiple **KGen** queries one by one. Essentially, in the ℓ-th hybrid, the first ℓ **KGen** queries are answered with $\mathbf{y}^{(1)}$, and the rest of the **KGen** queries are answered using $\mathbf{y}^{(0)}$. It suffices to argue that the $(\ell-1)$-th hybrid and the ℓ-th hybrid are computationally indistinguishable, and this turns out to be the most subtle step in our proof. To achieve this, let us consider the following modification of the $(\ell-1)$-th hybrid. Henceforth the ℓ-th **KGen** query is also called the *challenge* **KGen** query, and the two vectors submitted during this query are denoted $\mathbf{y}_*^{(0)}$ and $\mathbf{y}_*^{(1)}$ respectively:

1. During the ℓ-th **KGen** quer, for computing components of the key corresponding to honest players, the challenger switches the FE.**Enc** inside the challenge **KGen** query to a simulated encryption which does not use the underlying MCFE's functional key as input. Corrupt players' key components are still computed honestly.
   Correspondingly, in every time step, the challenger answers **Enc** queries by calling the a simulated FE.**KGen** for every honest client i's ciphertext component: the i-th simulated FE.**KGen** embeds the i-th partial decryption when paired with the challenge key for $\mathbf{y}_*^{(0)}$ of the underlying MCFE scheme. This step relies on the 1-SEL-SIM security of the single-input FE scheme (defined in the the online full version [68]).
2. At this moment, due to the randomizing terms, and the aforementioned admissibility rule, we argue that during each **Enc** query, instead of encoding in the simulated FE.**KGen** the partial decryptions when paired with the challenge key for $\mathbf{y}_*^{(0)}$ of the underlying MCFE scheme, we could use $\mathbf{y}_*^{(1)}$ instead. This step is more involved and requires the technical condition in Remark 3.

From this point onwards, we can use a symmetric argument to switch all the way to the aforementioned ℓ-th hybrid, in which the first ℓ **KGen** queries are answered with $\mathbf{y}^{(1)}$, and the remaining answered with $\mathbf{y}^{(0)}$. We defer the detailed proof to the subsequent formal sections.

3.5 Constructing NIAR with Receiver-Insider Protection

Construction. With a function-hiding MCFE-for-selection scheme henceforth denoted $\mathsf{MCFE}^{\mathrm{ffh}}$, we can construct a NIAR scheme in a natural fashion informally described below:

– **Setup**: The idea is to use the $\mathsf{MCFE}^{\mathrm{ffh}}$ scheme to generate functional keys for n selection vectors, denoted $\mathsf{tk}_1, \ldots, \mathsf{tk}_n$, where tk_i is for selecting the message received by receiver $i \in [n]$. The colletion $\{\mathsf{tk}_i\}_{i\in[n]}$ is given to the router as the token. The $\mathsf{MCFE}^{\mathrm{ffh}}$ also generates n secret encryption keys denoted $\{\mathsf{ek}_i\}_{i\in[n]}$, one for each sender. Finally, the setup procedure generates n symmetric encryption keys, one for each sender-receiver pair.
– **Enc**: During each time step t, to encrypt a message $x_{i,t}$, the i-th sender first encrypts $x_{i,t}$ with its symmetric key shared with its receiver — let $c_{i,t}$ denote the resulting ciphertext. Now, call $\mathsf{ct}_{i,t} := \mathsf{MCFE}^{\mathrm{ffh}}.\mathbf{Enc}(\mathsf{mpk}, \mathsf{ek}_i, c_{i,t})$ to further encrypt $c_{i,t}$ and obtain a final ciphertext $\mathsf{ct}_{i,t}$. Here, we abuse notation slightly and use $\mathsf{MCFE}^{\mathrm{ffh}}.\mathbf{Enc}(\mathsf{mpk}, \mathsf{ek}_i, c_{i,t})$ to mean encrypting $c_{i,t}$ bit by bit with the $\mathsf{MCFE}^{\mathrm{ffh}}$ scheme.
– **Rte**: Using the n functional keys $\{\mathsf{tk}_i\}_{i\in[n]}$, a router can call $\mathsf{MCFE}^{\mathrm{ffh}}.\mathbf{Dec}$ to obtain the n inner ciphertexts encrypted under the symmetric keys, and send the corresponding inner ciphertext to each receiver.
– **Dec**: Finally, each receiver uses its symmetric key to decrypt the final outcome.

Proof roadmap. In the the online full version [68], we shall prove that as long as $\mathsf{MCFE}^{\mathrm{ffh}}$ satisfies function-hiding security and the symmetric-key encryption scheme employed is secure, then, the above NIAR construction satisfies *receiver-insider protection.* To prove this, we use the indistinguishability security notion for NIAR, which is shown to be equivalent to the simulation-based notion. Rouhgly speaking, the indistinguishability game for NIAR, denoted $\mathsf{NIAR\text{-}Expt}^b$ is indexed by a bit $b \in \{0, 1\}$: imagine the adversary $\mathcal{A}$ chooses two permutations $\pi^{(0)}$ and $\pi^{(1)}$, and specifies two sets of messages $\{x_{i,t}^{(0)}\}_{i\in\mathcal{H}_S}$ and $\{x_{i,t}^{(1)}\}_{i\in\mathcal{H}_S}$ to query in each time step t. The challenger gives $\mathcal{A}$ a token for $\pi^{(b)}$, and ciphertexts for $\{x_{i,t}^{(b)}\}_{i\in\mathcal{H}_S}$ in each time step t. An *admissible* adversary must choose the permutations and messages such that the leakage in the two worlds are the same, where the leakage contains the corrupt-to-$*$ part of the permutation and the messages received by corrupt receivers in every time step. We want to prove that any efficient, admissible $\mathcal{A}$ cannot distinguish whether it is playing $\mathsf{NIAR\text{-}Expt}^0$ or $\mathsf{NIAR\text{-}Expt}^1$.

To prove this, we first modify $\mathsf{NIAR\text{-}Expt}^b$ slightly to obtain a hybrid Hyb^b for $b \in \{0, 1\}$: in Hyb^b, we replace the inner symmetric-key encryption from honest senders to honest receivers with simulated ciphertexts. We can easily show that Hyb^b is computationally indistinguishable from $\mathsf{NIAR\text{-}Expt}^b$ by reducing to the security of the symmetric encryption scheme.

To complete the proof, the more challenging step is to show that Hyb^0 is computationally indistinguishable from Hyb^1 for any efficient, admissible adversary $\mathcal{A}$. Here, we want to leverage $\mathcal{A}$ to create an efficient reduction $\mathcal{B}$ that breaks the function-hiding security of the underlying $\mathsf{MCFE}^{\mathrm{ffh}}$ scheme. The subtlety is to make sure that $\mathcal{B}$ indeed respects the $\mathsf{MCFE}^{\mathrm{ffh}}$'s admissibility rules. In our formal proofs, we fix the randomness ψ consumed by the SE instances corresponding to each receiver in the set $\pi^{(0)}(\mathcal{H}_S) = \pi^{(1)}(\mathcal{H}_S)$, and prove that the two experiments are indistinguishability for every choice of fixed ψ. We then define the reduction $\mathcal{B}$ in a natural manner, and make a careful argument that if $\mathcal{A}$ satisfies the NIAR game's admissibility rule (for the receiver-insider protection notion), then $\mathcal{B}$ will indeed respect the admissibility rules of the underlying $\mathsf{MCFE}^{\mathrm{ffh}}$.

We defer the formal description and proofs to the the online full version [68].

3.6 Achieving Full Insider Protection

To upgrade our NIAR scheme to have full insider protection turns out to be more involved. As explained earlier in Section 1.3, for such a scheme to work, *all* the transformed ciphertexts output by **Rte** must change when a single sender's input ciphertext changes.

Construction (sketch). To accomplish this, we leverage a indistinguishability obfuscator for probabilistic circuits (piO) whose existence is implied by sub-exponentially secure indistinguishability obfuscation and sub-exponentially secure one-way functions [30].

– **Setup**: during the trusted setup, each receiver i receives the secret key of a PKE scheme (with special properties mentioned later); and each sender receives the encryption key generated by an $\mathsf{MCFE}^{\mathrm{ffh}}$ scheme.
  The router's token tk is a piO which encodes the $\mathsf{MCFE}^{\mathrm{ffh}}$ scheme's functional keys for all n selection vectors. Inside the piO, the following probabilistic program is evaluated:
  1. first, use the $\mathsf{MCFE}^{\mathrm{ffh}}$ functional keys to decrypt the messages that each receiver should receive;
  2. next, encrypt the messages under each receiver's respective public keys, and output the encrypted ciphertexts — note that the encryption scheme is randomized.

– **Enc**: in every time step, senders encrypt their messages using $\mathsf{MCFE}^{\mathrm{ffh}}$.
– **Rte**: in each time step, the router applies its token tk, which is an obfuscated program, to the n ciphertexts collected from senders. The outcome will be n transformed ciphertexts.
– **Dec**: When a receiver receives a transformed ciphertext, it simply uses its secret key to decrypt it.

Observe that in this construction, indeed, if a single sender's input ciphertext changes, *all* transformed ciphertexts output by the **Rte** procedure will change.

Proof roadmap and subtleties. We encounter some more subtleties when we attempt to prove the above construction secure. First, it turns out that for technical reasons, to prove the above scheme secure, we need the public-key encryption (PKE) scheme used by the piO to reencrypt output messages to satisfy a special property: the PKE must be a special trapdoor mode in which encryptions of 0 and 1 are identically distributed. Obviously, the trapdoor mode loses information and cannot support correct decryption. In fact, in the real world, we will never use the trapdoor mode — it is used only inside our security proofs. We henceforth call a PKE scheme with this special property a perfectly hiding trapdoor encryption (tPKE). Such a tPKE scheme can be constructed assuming DDH [30].

Informally, our proof strategy is the following: First, we modify the real-world experiment (in which $\pi^{(0)}$ and $x_{i,t}^{(0)}$ are used), and switch the tPKE instances corresponding to honest receivers' to use a trapdoor setup. This step can be reduced to the tPKE's security, since the adversary does not have the tPKE instances' secret keys corresponding to honest receivers. Next, we modify the obfuscated program to *no longer use the functional keys corresponding to the honest receivers*; instead, the obfuscated program will simply output encryptions of 0 under the trapdoor public keys for honest receivers. For corrupt receivers, the obfuscated program still behaves like the real world: use the $\mathsf{MCFE}^{\mathsf{ffh}}$ scheme's **Dec** procedure to decrypt the messages they ought to receive, and output encryptions of these messages under each corrupt receiver's public keys, respectively. This step relies on the security of the piO and the fact that the modified program is "distributionally equivalent" to the original program. At this moment, the obfuscated program *no longer uses the functional keys for honest receivers*, and only at this point can we rely on the $\mathsf{MCFE}^{\mathsf{ffh}}$'s security and switch from using $\pi^{(0)}$ in the setup and encrypting $x_{i,t}^{(0)}$ to using $\pi^{(1)}$ in the setup and encrypting $x_{i,t}^{(1)}$. The remaining hybrids are symmetric to the above, such that eventually we arrive at an experiment that is the same as the real-world experiment in which $\pi^{(1)}$ and $x_{i,t}^{(1)}$ are used by the challenger.

Notice that in our construction, we use the piO to obfuscate the $\mathsf{MCFE}^{\mathsf{ffh}}$ scheme's **Dec** procedure using all n functional keys. One natural question is why we did not directly use the piO to obfuscate a program that calls the **Rte** procedure of our earlier NIAR scheme (with receiver-insider protection) and then encrypts the n outcomes using n instances of tPKE. It turns out that our proof strategy would not have worked for the latter, exactly because in our proofs, we needed an intermediate hybrid to completely stop using functional keys for honest receivers — intuitively, this is how we can prove the privacy of messages received by honest receivers. This explains why in our construction and proofs, we need to open up the NIAR scheme and directly manipulate the functional keys of the underlying $\mathsf{MCFE}^{\mathsf{ffh}}$.

3.7 Achieving Fault Tolerance

So far in our constructions, unless all senders send their encryption during a certain time step, the router would fail to perform the **Rte** operation. Such a scheme relies all senders to be online all the time, and thus is not fault-tolerant.

We modify our earlier NIAR abstraction to one that is fault-tolerant. The idea is to let **Enc** and **Rte** take an extra parameter $\mathcal{O} \subseteq [n]$ which denotes the set of senders that remain online. Additionally, **Rte** now takes in only ciphertexts from those in $\mathcal{O}$. In fact, our fault-tolerant NIAR abstraction can be viewed as a generalization of the non-fault-tolerant version.

To achieve fault tolerance, we observe that the underlying CPRF construction we use has a nice fault-tolerance property. In fact, we can modify the CPRF's evaluation function to take in $\mathcal{O}$, such that the following is satisfied:

$$\forall t \in \mathbb{N}: \quad \sum_{i \in \mathcal{O}} \mathsf{CPRF}.\mathbf{Eval}(K_i, t, \mathcal{O}) = 0$$

This way, for every receiver whose corresponding sender is in the online set $\mathcal{O}$, the router can correctly perform the $\mathsf{MCFE}^{\mathrm{ffh}}$'s decryption procedure using only ciphertexts from those in $\mathcal{O}$. Note that the recent elegant work of Bonawitz et al. [24] also made a similar observation of the fault-tolerance of the CPRF, and leveraged it to enable fault-tolerant, privacy-preserving federated learning — this is not explicitly stated in their paper but implicit in their constructions.

If a receiver i's corresponding sender is no longer online, however, then the $\mathsf{MCFE}^{\mathrm{ffh}}$'s decryption procedure will output an inner ciphertext of $\mathbf{0}$ for receiver i. Since receiver i cannot decrypt the $\mathbf{0}$ ciphertext using its symmetric key, it will simply output $\perp$ — this is inevitable since the corresponding sender is no longer around. However, the router can also observe that receiver i received an inner-ciphertext $\mathbf{0}$. In this way, if the adversary is able to drop the senders one by one and check which receiver starts to receive an inner ciphertext of $\mathbf{0}$, it can learn the receivers paired up with crashed senders. In our subsequent formal sections, we shall prove that in this fault-tolerant NIAR scheme, indeed the adversary can learn only the (corrupt+crashed)-to-$*$ part of the permutation π, as well as the messages received by corrupt receivers every time step, and nothing else.

Finally, using techniques similar to those sketched in Section 3.6, we can upgrade the security of the above fault-tolerant scheme to full insider protection, i.e., only the (corrupt + crashed)-to-corrupt part of the permutation is leaked as well as the messages received by corrupt receivers, but nothing else. As mentioned earlier, this leakage is inherent and unavoidable for any fault-tolerant NIAR scheme, since the adversary can always make the senders crash one by one and check which corrupt receiver now starts to receive $\perp$.

Of course, the above description is a gross simplification omitting various subtleties both in definitions and constructions. We refer the reader to the the online full version [68] for the detailed definitions, constructions, and proofs.

Deferred contents. Due to lack of space, the formal definitions, constructions and proofs can be found in our online full version [68].

Acknowledgments. This work is in part supported by a Packard Fellowship, a DARPA SIEVE grant under a sub-contract from SRI, and NSF grants under award numbers 2001026 and 1601879.

References

1. Computer science research and practice on slack.
2. M. Abdalla, F. Benhamouda, and R. Gay. From single-input to multi-client inner-product functional encryption. In *Asiacrypt*, 2019.
3. M. Abdalla, F. Bourse, A. D. Caro, and D. Pointcheval. Better security for functional encryption for inner product evaluations. Cryptology ePrint 2016/011, 2016.
4. M. Abdalla, F. Bourse, H. Marival, D. Pointcheval, A. Soleimanian, and H. Waldner. Multi-client inner-product functional encryption in the random-oracle model. Cryptology ePrint Archive, Report 2020/788, 2020.
5. M. Abdalla, D. Catalano, D. Fiore, R. Gay, and B. Ursu. Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In *CRYPTO*, 2018.
6. M. Abdalla, R. Gay, M. Raykova, and H. Wee. Multi-input inner-product functional encryption from pairings. In *EUROCRYPT*, 2017.
7. M. Abdalla, J. Gong, and H. Wee. Functional encryption for attribute-weighted sums from k-lin. In *CRYPTO*, volume 12170, pages 685–716, 2020.
8. M. Abe. Mix-networks on permutation networks. In *ASIACRYPT*, 1999.
9. I. Abraham, B. Pinkas, and A. Yanai. Blinder: Mpc based scalable and robust anonymous committed broadcast. In *ACM CCS*, 2020.
10. B. Adida. Helios: Web-based open-audit voting. In *Usenix Security*, 2008.
11. B. Adida and D. Wikström. How to shuffle in public. In *TCC*, 2007.
12. S. Agrawal, B. Libert, and D. Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In *CRYPTO*, 2016.
13. N. Alexopoulos, A. Kiayias, R. Talviste, and T. Zacharias. Mcmix: Anonymous messaging via secure multiparty computation. In *Usenix Security*, 2017.
14. F. Baldimtsi and A. Lysyanskaya. Anonymous credentials light. In *CCS*, 2013.
15. B. Balle, J. Bell, A. Gascón, and K. Nissim. Differentially private summation with multi-message shuffling. *CoRR*, abs/1906.09116, 2019.
16. B. Balle, J. Bell, A. Gascón, and K. Nissim. The privacy blanket of the shuffle model. In *CRYPTO*, 2019.
17. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.
18. S. Bayer and J. Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Eurocrypt*, volume 7237, pages 263–280, 2012.
19. A. Beimel, K. Nissim, and E. Omri. Distributed private data analysis: Simultaneously solving how and what. In *CRYPTO*, page 451–468, 2008.
20. M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. P-signatures and noninteractive anonymous credentials. In *TCC*, 2008.
21. E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE S & P*, 2014.
22. A. Bishop, A. Jain, and L. Kowalczyk. Function-hiding inner product encryption. In *Asiacrypt*, 2015.
23. A. Bittau, U. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *SOSP*, 2017.
24. K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *CCS*, 2017.

25. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC*, 2011.
26. D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *Theoretical Cryptography Conference (TCC)*, 2007.
27. Z. Brakerski, N. Dottling, S. Garg, and G. Malavolta. Factoring and pairings are not necessary for io: Circular-secure lwe suffices. Cryptology ePrint 2020/1024, 2020.
28. Z. Brakerski, I. Komargodski, and G. Segev. Multi-input functional encryption in the private-key setting: Stronger security from weaker assumptions. *J. Cryptology*, 31(2):434–520, 2018.
29. J. Camenisch and A. Lysyanskaya. A formal treatment of onion routing. In *CRYPTO*, 2005.
30. R. Canetti, H. Lin, S. Tessaro, and V. Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In *Theory of Cryptography*, pages 468–497, 2015.
31. T.-H. H. Chan, E. Shi, and D. Song. Optimal lower bound for differentially private multi-party aggregation. In *European Symposium on Algorithms (ESA)*, 2012.
32. D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R. L. Rivest, and A. T. Sherman, editors, *CRYPTO*, 1982.
33. D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *CRYPTO*, 1990.
34. D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, Feb. 1981.
35. D. L. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, Mar. 1988.
36. A. Cheu, A. Smith, J. Ullman, D. Zeber, and M. Zhilyaev. Distributed differential privacy via shuffling. In *Eurocrypt*, 2019.
37. J. Chotard, E. D. Sans, R. Gay, D. H. Phan, and D. Pointcheval. Decentralized multi-client functional encryption for inner product. In *Asiacrypt*, 2018.
38. H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An anonymous messaging system handling millions of users. In *S & P*, 2015.
39. H. Corrigan-Gibbs and B. Ford. Dissent: Accountable anonymous group messaging. In *CCS*, page 340–350, 2010.
40. G. Danezis and C. Diaz. A survey of anonymous communication channels. Technical Report MSR-TR-2008-35, Microsoft Research, 2008.
41. J. P. Degabriele and M. Stam. Untagging tor: A formal treatment of onion encryption. In *EUROCRYPT*, 2018.
42. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, 2004.
43. C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.
44. M. Edman and B. Yener. On anonymity in an electronic society: A survey of anonymous communication systems. *ACM Comput. Surv.*, 42(1), Dec. 2009.
45. Ú. Erlingsson, V. Feldman, I. Mironov, A. Raghunathan, K. Talwar, and A. Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In *SODA*, 2019.
46. U. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *CCS*, 2014.
47. R. Gay and R. Pass. Indistinguishability obfuscation from circular security. Cryptology ePrint Archive, Report 2020/1010, 2020.
48. Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *J. Comput. Syst. Sci.*, 60(3), 2000.

49. B. Ghazi, R. Pagh, and A. Velingker. Scalable and differentially private distributed aggregation in the shuffled model. *CoRR*, abs/1906.08320, 2019.
50. D. Goldschlag, M. Reed, and P. Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42:39–41, 1999.
51. S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F. Liu, A. Sahai, E. Shi, and H. Zhou. Multi-input functional encryption. In *Eurocrypt*, 2014.
52. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Predicate encryption for circuits from LWE. In *CRYPTO*, 2015.
53. E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *NDSS*, 2017.
54. S. Hohenberger, S. A. Myers, R. Pass, and A. Shelat. ANONIZE: A large-scale anonymous survey system. In *IEEE S & P*, 2014.
55. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography from anonymity. In *FOCS*, 2006.
56. A. Jain, H. Lin, and A. Sahai. Indistinguishability obfuscation from well-founded assumptions. Cryptology ePrint Archive, Report 2020/1003, 2020.
57. B. Libert and R. Titiu. Multi-client functional encryption for linear functions in the standard model from lwe. 2019.
58. H. Lin. Indistinguishability obfuscation from sxdh on 5-linear maps and locality-5 prgs. In *CRYPTO*, 2017.
59. H. Lin and V. Vaikuntanathan. Indistinguishability obfuscation from DDH-like assumptions on constant-degree graded encodings. In *FOCS*, pages 11–20, 2016.
60. R. Ostrovsky and V. Shoup. Private information storage (extended abstract). In *STOC*, pages 294–303, 1997.
61. A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, 2005.
62. K. Sako and J. Kilian. Receipt-free mix-type voting scheme: A practical solution to the implementation of a voting booth. In *EUROCRYPT*, 1995.
63. E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In *TCC*, 2009.
64. R. Sherwood, B. Bhattacharjee, and A. Srinivasan. P5: A protocol for scalable anonymous communication. In *IEEE S & P*, 2002.
65. E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *S & P*, 2007.
66. E. Shi, T.-H. H. Chan, E. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS*, 2011.
67. E. Shi, T.-H. H. Chan, E. Rieffel, and D. Song. Distributed private data analysis: Lower bounds and practical constructions. *ACM Trans. Algorithms*, 13(4), 2017.
68. E. Shi and K. Wu. Non-interactive anonymous router. 2021.
69. F. Shirazi, M. Simeonovski, M. R. Asghar, M. Backes, and C. Diaz. A survey on routing in anonymous communication protocols. 2018.
70. N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich. Stadium: A distributed metadata-private messaging system. In *SOSP*, 2017.
71. J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *SOSP*, 2015.
72. H. Wee. New techniques for attribute-hiding in prime-order bilinear groups. Manuscript, 2016.
73. H. Wee and D. Wichs. Candidate obfuscation via oblivious LWE sampling. Cryptology ePrint Archive, Report 2020/1042, 2020.
74. L. Zhuang, F. Zhou, B. Y. Zhao, and A. Rowstron. Cashmere: Resilient anonymous routing. In *NSDI*, 2005.