Non-Interactive Anonymous Router with Quasi-Linear Router Computation[*]

Rex Fernando   Elaine Shi   Pratik Soni   Nikhil Vanjani

Carnegie Mellon University

Abstract

Anonymous routing is an important cryptographic primitive that allows users to communicate privately on the Internet, without revealing their message contents or their contacts. Until the very recent work of Shi and Wu (Eurocrypt'21), all classical anonymous routing schemes are *interactive* protocols, and their security rely on a *threshold* number of the routers being honest. The recent work of Shi and Wu suggested a new abstraction called Non-Interactive Anonymous Router (NIAR), and showed how to achieve anonymous routing *non-interactively* for the first time. In particular, a *single untrusted router* receives a token which allows it to obliviously apply a permutation to a set of encrypted messages from the senders. While Shi and Wu's scheme is efficient in other dimensions, one unsatisfying aspect of their construction is that the router takes time *quadratic* in the number of senders to obliviously route their messages.

In this work, we show how to construct a non-interactive anonymous router scheme with *sub-quadratic* router computation, assuming the existence of subexponential indistinguishability obfuscation and one-way permutation. To achieve this, we devise new techniques for reasoning about a *network of obfuscated programs*.

[*]A concurrent and independent work by Bunn, Kushilevitz, and Ostrovsky [BKO22] proposes a NIAR scheme with quasi-linear router computation from DDH, QR, or LWE. We stress that their security definition is of a *fundamentally different nature* from ours and that of Shi and Wu [SW21], and in particular, their notion is insufficient for constructing a Non-Interactive Anonymous Shuffler which is one of the main implications of our result. Thus their result is incomparable and of a fundamentally different nature to ours. See the end of Section 1 for more detailed discussions.

1 Introduction

Anonymous communication systems allow users to communicate without revealing their identities and messages. The earliest design of an anonymous communication system goes back to Chaum [Cha81] who proposed the design of an encrypted email service that additionally hides the identities of the sender and the receiver. Since then, numerous approaches have been proposed to build anonymous routing schemes [Cha81, Abe99, BG12, Cha88, CGF10, DMS04, GRS99, CBM15, ZZZR05, vdHLZZ15, TGL+17, SBS02] – a key component of anonymous communication systems. These include mix-nets [Cha81, Abe99, BG12], the Dining Cryptographers' nets [Cha88, CGF10, APY20], onion routing [DMS04, GRS99, DS18, CL05], multi-party-computation-based approaches [AKTZ17, HEK12, SA19], multi-server PIR-write [CBM15, OS97, GIKM00], as well as variants [ZZZR05, vdHLZZ15, TGL+17].

However, all of these routing schemes are *interactive*, where many servers or routers engage in an interactive protocol to achieve routing. The security relies on *threshold* type assumptions, e.g., majority or at least one of the routers must be honest. This is unsatisfactory since the threshold-based trust model increases the barrier of adoption, the interactivity leads to higher network latency, and finally, the schemes provide no guarantees when *all* routers may be malicious, or worse yet, colluding with a subset of the receivers and senders.

The recent work of Shi and Wu [SW21] was the first to study the feasibility of *non-interactive* anonymous routing (NIAR) with a *single, untrusted* router which can additionally collude with a subset of senders and receivers. The setting is as follows: there are n senders and n receivers, and each sender u wants to talk to a unique receiver $v = \pi(u)$ given by the routing permutation π. The NIAR scheme has a trusted setup that given a routing permutation π outputs encryption keys for senders, decryption keys for receivers, and a routing token for the router that secretly encrypts the routing permutation. In each time step, each sender uses its encryption key to encrypt a message. The router upon collecting all the n ciphertexts applies the routing token to permute them and convert them into n transformed ciphertexts, and delivers each receiver a single transformed ciphertext. Each receiver learns their message by decrypting the received ciphertext with their key. The computation of the permuted ciphertexts can be viewed as the router *obliviously* applying the routing permutation π, without learning π.

NIAR was shown to have numerous applications in [SW21] including realizing a non-interactive anonymous shuffle (NIAS) where n senders send encryptions of their private messages to an entity called *shuffler* who, upon decryption, learns a permutation of the senders' messages, without learning the mapping between each message and the corresponding sender. A NIAS scheme can be used to instantiate the *shuffle model* adopted in a line of work on distributed, differentially private mechanisms [CSU+19, BBGN19b, GPV19, EFM+19, BEM+17, BBGN19a]. We can realize such a NIAS construction from NIAR by having the shuffler act on behalf of the NIAR router and all n receivers, as long as the underlying NIAR scheme provides meaningful security even when all the receivers collude with the router – termed as *receiver-insider* security by Shi and Wu [SW21].

Shi and Wu [SW21] give a NIAR scheme that satisfies receiver-insider security assuming the hardness of the decisional linear problem. Their scheme not only supports an *unbounded* number of time steps, but also has good efficiency features: each sender only needs to send $O_\lambda(1)$ bits per time step to encrypt a bit,[1] moreover, the sender and receiver keys are $O_\lambda(1)$ and the public parameters are $O_\lambda(n)$ in size. What is undesirable is their token size and router computation, both of which are *quadratic* in the number of users n, that is, $O_\lambda(n^2)$. We also stress that the quadratic router computation drawback pertains not only to the work of Shi and

[1]Throughout the paper, we use $O_\lambda(\cdot)$ to hide $\mathsf{poly}(\lambda)$ multiplicative factors where λ denotes the security parameter.

Wu [SW21]. As Gordon et al. [GKLX22] pointed out, even in classical, *interactive* anonymous routing constructions [Cha88, Cha81, SA19, HEK12], the total router computation is typically $\Omega(nm)$ where n and m denote the number of clients and routers, respectively — therefore, in a peer-to-peer environment where the clients also act as routers, the total computation would be quadratic in n.

The status quo gives rise to the following natural question:

> *Can we have a non-interactive anonymous router scheme with* subquadratic *router computation?*

The recent work of Bünz, Hu, Matsuo and Shi [BHMS21] made a notable attempt at answering the question. They could not fully achieve the above goal, but did suggest a scheme with $O(\lambda^{\frac{1}{\gamma}} \cdot n^{1+\gamma})$ router computation for any $\gamma \in (0, 1)$. Their scheme has two significant drawbacks. First, their subquadratic router computation comes at the price of relaxing the security definition to (ϵ, δ)-*differential privacy* [DMNS06]. In other words, their scheme ensures that the adversary's views are indistinguishable only for two *neighboring* routing permutations (whereas full security requires indistinguishability for any two routing permutations). Not only is differential privacy a significantly weaker security notion, it can also leads to additional complications in terms of managing the privacy budget. Second, their $\mathsf{poly}(\lambda)$ dependency is not a fixed one — to improve the dependence on the parameter n, we want to choose an arbitrarily small γ, however, this would significantly blow up the polynomial dependence on the security parameter λ.

1.1 Our Results

We revisit the question of constructing an anonymous router scheme with subquadratic router computation. As mentioned, this question has not been satisfactorily answered even in the classical *interative* setting [Cha88, Cha81, SA19, HEK12, GKLX22]; however, we focus on the *non-interactive* setting in our work.

We address the above question in the affirmative. Specifically, we propose a NIAR scheme that achieves *quasilinear* in n router computation. Our NIAR scheme achieves a selective-security relaxation of Shi and Wu's receiver-insider security definition. In terms of assumptions, we need the subexponential security of indistinguishability obfuscator [GGH$^+$13, JLS21, GP20, WW20, BDGM20] and one-way permutations.

Theorem 1.1 (Informal)**.** *Let λ be a security parameter. Let $n = n(\lambda)$ be the number of senders/receivers. Then, assuming the existence of subexponentially-secure indistinguishability obfuscator and one-way permutations, there exists a NIAR scheme that satisfies selectively receiver-insider security. Further, the asymptotical performance bounds are as follows:*

1. *the token size and router computation per time step is $\widetilde{O}_\lambda(n)$ where $\widetilde{O}_\lambda(\cdot)$ hides $\mathsf{poly}(\lambda, \log n)$ factors for some fixed $\mathsf{poly}(\cdot)$;*
2. *the per-sender communication and encryption time per bit of the message is $\widetilde{O}_\lambda(1)$;*
3. *each sender key is of length $\widetilde{O}_\lambda(1)$, each receiver key is of length $O_\lambda(1)$.*

Our work should be viewed as an initial theoretical exploaration of the feasibility of anonymous router with subquadratic router computation. Our feasibility results naturally raise several open questions for future work. Can we achieve subquadratic computation with polynomial-strength security assumptions? Can we achieve subquadratic computation with standard assumptions? Last but not the least, can we construct a scheme with good concrete performance?

Technical highlight. For achieving quasilinear router computation, we employ a novel approach where we obfuscate polylogarithmically-sized gates in a *layered routing network* of size $\tilde{O}(n)$, resulting in a *network of iO obfuscated circuits*. Such an approach is reminiscent of the work of Canetti et al. [CLTV15] who build levelled fully-homomorphic encryption scheme from iO. However, as we elaborate later, our setting is significantly more challenging and requires novel techniques. In our setting, there are multiple encrypters some of whom may be malicious, whereas in the setting of Canetti et al. [CLTV15], there is a single encrypter who is assumed to be honest. In our construction, it is important for each obfuscated gate to authenticate the outputs of an obfuscated circuit in a previous layer. The notion of iO guarantees that obfuscations of two functionally equivalent programs/circuits are indistinguishable. Despite the weak guarantee, iO has been shown to have numerous applications. However, as also evident from prior works, computationally secure primitives are generally incompatible with the functional equivalence requirements of iO. Therefore, in our case we need to develop new iO-compatible techniques for authentication, and a key stepping stone is the construction of a Somewhere Statistically Unforgeable (SSU) signature scheme.

Somewhere Statistically Unforgeable (SSU) signatures. We introduce a new signature scheme called *somewhere statistically unforgeable* signatures for authentication. Informally, an SSU signature scheme is a signature scheme for which one can compute a pair of *punctured* signing and *punctured* verification key w.r.t. a set of points X such that *no* valid signature even exists for points out of the set X. Further, we require that the distributions of the real verification key and punctured verification key be indistinguishable. This primitive can be seen to be in spirit with prior work on somewhere statistically secure primitives [HW15, OPWW15]. We construct such a signature scheme from the subexponential hardness of iO and OWPs.

Theorem 1.2. *Assuming the existence of a subexponentially secure indistinguishability obfuscator and one-way permutations, there exists a somewhere statistically unforgeable signature scheme for sets* $X_{t^*,x^*} = \{(t,x) : t \neq t^* \lor (t = t^* \land x \neq x^*)\}$.

Comparison with concurrent work. A concurrent work by Bunn, Kushilevitz, and Ostrovsky [BKO22] proposes an alternate non-interactive router scheme (which they also refer to as NIAR), which achieves quasi-linear router computation from DDH, QR, or LWE. We stress that their security definition is of a *fundamentally different nature* than ours and that of the work of Shi and Wu [SW21], which introduced the study of NIAR and gave the first definition of security for such a primitive. Namely, the work of [BKO22] achieves *sender-insider security*, that is, they allow each receiver to know its corresponding sender, and corrupt senders (even when colluding with the router) should not learn which honest receivers they are sending to. In contrast, we achieve *receiver-insider security*, which is the primary security notion given in Shi and Wu [SW21]. In receiver-insider security, we allow each sender to know the receiver it is sending to; however, corrupt receivers (even when colluding with the router) should not find out which honest senders they are receiving from. Crucially, receiver-insider security is necessary for the *Non-Interactive Anonymous Shuffler* (NIAS) application, which is relevant to a line of work distributed differentially private mechanisms in the *shuffle model* [CSU+19, BBGN19b, GPV19, EFM+19, BEM+17, BBGN19a]. By contrast, sender-insider security is not sufficient for realizing NIAS. Intuitively, receiver-insider security seems more challenging to obtain than sender-insider security. To see this, observe that if router computation overhead is not of concern, then a NIAR with *sender*-insider security is directly implied by Private Information Retrieval (PIR). By contrast, PIR does not directly lead to NIAR with *receiver*-insider security. So far, the only known way to achieve receiver-insider security is the work by Shi and Wu [SW21], and they get it through functional encryption techniques. In this sense, the result as well as the techniques of Bunn et al. [BKO22] are of a very different nature from ours.

2 Technical Roadmap

In this section, we give an overview of our techniques starting with defining the notion of a non-interactive anonymous router (NIAR).

2.1 Background: Non-Interactive Anonymous Router (NIAR)

We review the original definition of Non-Interactive Anonymous Router (NIAR) by Shi and Wu [SW21], using some of their notation and descriptions verbatim. Imagine that there are n senders and n receivers, and each sender wants to talk to a distinct receiver. Henceforth we use π to denote the routing permutation. For example, $\pi(2) = 3$ means that sender 2 wants to talk to receiver 3.

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

- $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, n, \pi)$: the trusted **Setup** algorithm takes the security parameter 1^λ, the length of the time step tlen, the length of the messages len, the number of senders/receivers n, and a permuation π. The algorithm outputs a sender key for each sender denoted $\{\mathsf{ek}_u\}_{u\in[n]}$, a receiver key for each receiver denoted $\{\mathsf{rk}_u\}_{u\in[n]}$, and a token for the router denoted tk.

- $\mathsf{CT}_{u,t} \leftarrow \mathbf{Enc}(\mathsf{ek}_u, x_{u,t}, t)$: sender u uses its sender key ek_u to encrypt the message $x_{u,t} \in \{0,1\}^{\mathsf{len}}$ where $t \in \{0,1\}^{\mathsf{tlen}}$ denotes the current time step. The **Enc** algorithm produces a ciphertext $\mathsf{CT}_{u,t}$.

- $(\mathsf{CT}'_{1,t}, \mathsf{CT}'_{2,t}, \ldots, \mathsf{CT}'_{n,t}) \leftarrow \mathbf{Rte}(\mathsf{tk}, \mathsf{CT}_{1,t}, \mathsf{CT}_{2,t}, \ldots, \mathsf{CT}_{n,t})$: 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 $\mathsf{CT}'_{u,t}$ is destined for the receiver $u \in [n]$.

- $x \leftarrow \mathbf{Dec}(\mathsf{rk}_u, \mathsf{CT}'_{u,t}, t)$: the decryption algorithm **Dec** takes a receiver key rk_u, a transformed ciphertext $\mathsf{CT}'_{u,t}$, a time step t, and outputs a decrypted message x.

Correctness is defined in the most natural manner, and we defer the formal definition to Section 3.2. We focus on defining security.

Full security. In this paper, we would like to achieve "security with receiver-insider protection" which is the main security notion suggested by Shi and Wu for NIAR [SW21]. Henceforth when we say *full security*, we mean *security with receiver-insider protection*. Roughly speaking, receiver-insider protection requires that a corrupt receiver colluding with the router cannot learn which honest sender it is receiving messages from. On the other hand, corrupt senders are allowed to learn who their destinations are. Shi and Wu [SW21] argued that receiver-insider protection is sufficient for most conceivable applications. In particular, to instantiate a Non-Interactive Anonymous Shuffler (NIAS) scheme which leads to applications in shuffle-model distributed differential privacy [SW21], security with receiver-insider protection is sufficient.

Henceforth, we use the notation $\mathcal{K}_R$ and $\mathcal{K}_S$ to denote the set of corrupt receivers and senders, respectively; we use $\mathcal{H}_S$ and $\mathcal{H}_R$ to denote the set of honest senders and honest receivers, respectively. Consider the following experiment $\mathsf{NIAR\text{-}Expt}^{b,\mathcal{A}}(1^\lambda, \mathsf{tlen}, \mathsf{len})$ parametrized by a bit $b \in \{0,1\}$:

- $n, \mathcal{K}_S, \mathcal{K}_R, \pi^{(0)}, \pi^{(1)} \leftarrow \mathcal{A}(1^\lambda, \mathsf{tlen}, \mathsf{len})$
- $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, n, \pi^{(b)})$
- For $t = 1, 2, \ldots$:
  - if $t = 1$ then $\{x^{(0)}_{u,t}\}_{u\in\mathcal{H}_S}, \{x^{(1)}_{u,t}\}_{u\in\mathcal{H}_S} \leftarrow \mathcal{A}(\mathsf{tk}, \{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{rk}_u\}_{u\in\mathcal{K}_R})$;
    else $\{x^{(0)}_{u,t}\}_{i\in\mathcal{H}_S}, \{x^{(1)}_{u,t}\}_{u\in\mathcal{H}_S} \leftarrow \mathcal{A}(\{\mathsf{CT}_{u,t-1}\}_{u\in\mathcal{H}_S})$;
  - for $u \in \mathcal{H}_S$, $\mathsf{CT}_{u,t} \leftarrow \mathbf{Enc}(\mathsf{ek}_u, x^{(b)}_{u,t}, t)$

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

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

Definition 2.1 (Full security)**.** We say that a NIAR scheme satisfies full security (i.e., security with receiver-insider protection), iff for any non-uniform p.p.t. admissible $\mathcal{A}$, its views in the two experiments $\mathsf{NIAR\text{-}Expt}^{0,\mathcal{A}}(1^\lambda, \mathsf{tlen}, \mathsf{len})$ and $\mathsf{NIAR\text{-}Expt}^{1,\mathcal{A}}(1^\lambda, \mathsf{tlen}, \mathsf{len})$ are computationally indistinguishable.

More intuitively, in experiment $\mathsf{NIAR\text{-}Expt}^{b,\mathcal{A}}$, an adversary submits two permutations $\pi^{(0)}$ and $\pi^{(1)}$, as well as two sets of honest plaintexts $\{x^{(0)}_{u,t}\}_{u\in\mathcal{H}_S}$ and $\{x^{(1)}_{u,t}\}_{u\in\mathcal{H}_S}$ in each time step. The challenger returns a token for $\pi^{(b)}$, and ciphertexts for the vector $\{x^{(b)}_{u,t}\}_{u\in\mathcal{H}_S}$ on behalf of honest senders in every time step. Full security requires that as long as the adversary's queries do not allow it to trivially differentiate between the two worlds $b = 0$ and $b = 1$, it should not be able to distinguish the two worlds. In particular, the adversary can trivially tell the two worlds apart if the corrupt senders have different destinations, or if the corrupt receivers receive different messages from honest senders in some time step in the two worlds. Therefore, an *admissible* adversary should never submit such queries.

2.2 Strawman: A Single VBB-Obfuscated Program

If we had virtual blackbox (VBB) obfuscation, then a most straightforward strawman approach to build a NIAR scheme is to use VBB obfuscation to obfuscate the following program which results in the routing token:

1. decrypt the all senders' incoming ciphertexts;
2. permute the plaintexts according to the routing permutation π; and
3. re-encrypt the permuted messages under the corresponding receivers' keys.

Here, we want the obfuscation to hide 1) all sender and receiver keys; and 2) the routing permutation π itself.

This approach, however, has two significant drawbacks. First, VBB obfuscation is known to be impossible [BGI+01b]. Second, even if we can somehow replace the VBB obfucation with a weaker alternative such as indistinguishability obfuscation [GGH+13], this approach still completely fails in terms of efficiency: because the obfuscated program has size linear in n, and the size and runtime of the obfuscated program would be $\mathsf{poly}(n)$ — this defeats our purpose of achieving subquadratic router computation.

2.3 Second Attempt: A Network of VBB-Obfuscated Programs

We first sketch an idea to address the efficiency concern, still using VBB obfuscation. Later in this section, we will discuss how to replace the VBB obfuscation with indistinguishability obfuscation — doing so raises many non-trivial challenges, and we need new techniques to resolve them.

To achieve quasilinear router computation, our idea is to use a network of obfuscated programs rather than a single one. At a very high level, we will rely on a routing network.

Background on congestion-free routing network. A routing network is a layered directed acyclic graph where the layers are numbered $0, 1, \ldots, L$. Directed edges exist between only adjacent layers. The senders reside at layer 0 (also called the source layer) and the receivers reside at layer L (also called the destination layer). Each internal vertex is called a *gate*; the incoming and outgoing edges of a gate are called the gate's input and output *wires*, respectively. We want each sender to route a message to its designated receiver over a set of *edge-disjoint* paths. The path a sender u takes to reach its receiver is also called its *route*, which consists an ordered list of wires/edges in every level that the sender traverses to reach its destination. Now, if the sender attaches the route to its message, each gate along the way will know which outgoing wire to route this message.

We want to obfuscate each gate inside this routing network, forming a network of obfuscated programs. To make this idea work, the routing network must satisfy two properties:

- *Efficiency.* The routing network must be efficient for the resulting NIAR scheme to enjoy quasilinear router computation. First, the total size of the routing network must be small — we will use a routing network of size $O(n \log n)$. Second, each gate must have a small number of inputs and outputs. In our routing network, each gate has $O(\log^2 \lambda)$ inputs and outputs where λ is the security parameter.

- *Obliviousness.* For security, it is important that the corrupt senders' routes reveal no information about honest senders' destinations (beyond what is already leaked by the corrupt senders' own destinations). We formally define this notion in Definition 3.1 in the subsequent technical sections.

Indeed, a routing network satisfying these properties has been suggested in prior works [ACN+20, RS21]. Their construction relies on a butterfly network of $O(n \log n)$ size, where each gate has $O(\log^2 \lambda)$ incoming and outgoing wires. The source and destination layers (i.e., layers 0 and L) have $2n$ vertices each, whereas every intermediate layer has at most $O(n/\log^2 \lambda)$ gates. Therefore, the total number of wires in between adjacent layers must be upper bounded by $O(n)$. Abstractly, their construction can be thought of as the following. Given a routing permutation π that maps the senders to the receivers, a randomized algorithm $\mathsf{AssignRoutes}(1^\lambda, n, \pi)$ performs the following route assignment and it succeeds with all but negligible in λ probability:

- First, the $\mathsf{AssignRoutes}$ algorithm first maps each sender to a random position in the source layer, and maps the i-th receiver to a fixed position $2i - 1$ in the destination layer — henceforth we may assume that each sender's route also includes which source-layer vertex it is mapped to.

- Next, the $\mathsf{AssignRoutes}$ algorithm finds a set of edge-disjoint paths for each sender to route to its receiver.

The *obliviousness* property is captured with respect to $\mathsf{AssignRoutes}$ algorithm as follows. We say that a routing network satisfies obliviousness, iff there exists another simulated $\mathsf{AssignRoutes}^*$ algorithm and a negligible function $\mathsf{negl}(\cdot)$, such that for any corrupt set $\mathcal{K} \subseteq [n]$, for any two routing permutations π_0 and π_1 such that $\pi_0(i) = \pi_1(i)$ for any $i \in \mathcal{K}$, for either $b \in \{0, 1\}$,

$$\{(\{\mathsf{rte}_u\}_{u\in\mathcal{K}}, \{\mathsf{rte}_u\}_{u\notin\mathcal{K}}) : (\mathsf{rte}_1, \ldots, \mathsf{rte}_n) \leftarrow \mathsf{AssignRoutes}(1^\lambda, n, \pi_b)\}$$
$$\approx_{\mathsf{negl}(\lambda)}$$
$$\left\{\left(\{\mathsf{rte}_u\}_{u\in\mathcal{K}}, \left\{\mathsf{rte}^b_u\right\}_{u\notin\mathcal{K}}\right) : \left(\{\mathsf{rte}_u\}_{u\in\mathcal{K}}, \left\{\mathsf{rte}^0_u, \mathsf{rte}^1_u\right\}_{u\notin\mathcal{K}}\right) \leftarrow \mathsf{AssignRoutes}^*(1^\lambda, n, \pi_0, \pi_1, \mathcal{K})\right\}.$$

A network of VBB obfuscated programs. Using the above routing network, each gate is a circuitry that takes in $O(\log^2 \lambda)$ inputs, where each input contains a sender's message attached with a route. The next hop on the route uniquely specifies which outgoing wire to route this message. Therefore, the gate will then route every incoming message to an appropriate outgoing wire based on the attached route information.

We can therefore construct an efficient NIAR scheme using VBB obfuscation as follows. The **Setup** algorithm generates a master signing key sk and a verification key vk, as well as an authenticated encryption key for each wire in the routing network — henceforth we use sk_w to denote the wire key on some wire w. Each sender u's encryption key includes 1) a wire key sk_w where w denotes the wire of the source vertex corresponding to the sender; 2) its route denoted rte_u, along with a signature rsig_u that vouches for the route information.

Next, for each gate in the routing network, the **Setup** algorithm obfuscates the following program, which is parametrized with the wire keys of all its incoming and outgoing wires, as well as the global verification key vk:

- Receive an authenticated ciphertext from every incoming wire w. Use the corresponding wire key sk_w to verify and decrypt the incoming ciphertext. Abort if decryption fails; otherwise, we obtain either a filler $\perp$ or a tuple $(m, t, \mathsf{rte}, \mathsf{rsig})$ where m denotes a payload message, t denotes the time, and $(\mathsf{rte}, \mathsf{rsig})$ is an authenticated route. Abort if rsig does not verify for rte.

- For incoming wires that do not decrypt to $\perp$, check that they all have the same time step t. Abort if the check fails.

- Assign each incoming $(m, t, \mathsf{rte}, \mathsf{rsig})$ tuple to the corresponding output wire specified by the next hop on the rte. For all unpopulated outgoing wires, assign a filler $\perp$ to it.

- At this moment, every outgoing wire w either carries a real tuple of the form $(m, t, \mathsf{rte}, \mathsf{rsig})$ or a filler $\perp$. Using the corresponding wire key sk_w, encrypt the tuple or filler on all outgoing wires with an authenticated encryption scheme, and output the resulting ciphertexts.

Now, for sender u to send a message m to its receiver during time step t, the sender only has to use its sk_w to encrypt $(m, t, \mathsf{rte}_u, \mathsf{rsig}_u)$ using authenticated encryption. The algorithm for the router to perform routing and for receivers to decrypt their received ciphertexts are defined in the most natural manner.

In the above scheme, we stress that it is important to *authenticate* the inputs and outputs passed to the obfuscated programs. Otherwise, a malicious router could easily swap one or more of the inputs and observe whether any corrupt receivers' outputs are affected. This can allow the malicious router to infer which honest sender is sending to a corrupt receiver.

2.4 SSU Signatures

We want to replace the VBB obfuscation with indistinguishability obfuscation iO. This is the biggest technical challenge and the most sophisticated part of our construction. The first challenge is a well-known one: iO only provides indistinguishability for two *functionally equivalent* programs, and this property is not the easiest to work with in security reductions. To make it even more challenging, we need to work with a *network of* iO*s*, where the previous iO produces encrypted and authenticated outputs which are then forwarded to the next iO as input — this is where we need to devise novel proof techniques. To the best of our knowledge, the only previous work that uses iO with such a multi-hop structure is the work by Canetti et al. [CLTV15], who used a sequence of iOs to construct a fully homomorphic encryption scheme. However, we stress that our setting is much more challenging from a technical perspective, since we have to handle corrupt encrypters whereas in the setting of Canetti et al. [CLTV15], the encrypter is always honest. From another perspective, our work can also be viewed as using a network of obfuscated programs to asymptotically speed up a special type of function-hiding Multi-Client Functional Encryption (MCFE) — indeed, the earlier work of Shi and Wu [SW21] pointed out that NIAR can be viewed as a function-hiding MCFE scheme.

A key challenge in our proof is that we have to show that for the challenge time step,[2] the adversary has to use the correctly evaluated ciphertexts on honest and filler wires during evaluation, and it is unable to swap them to any other ciphertext of its choice. One technicality is that the "functional equivalence" requirement iO does not work well with traditional computationally authentication or signature schemes.

To make our proofs work, we need to introduce a special type of signature scheme called Somewhere Statistically Unforgeable (SSU) Signature. Informally speaking, we want to have a simulated setup algorithm that outputs a *punctured signing key* that can sign normally for all non-challenge time steps, but for the challenge time step t^* alone, it removes the ability to sign any other message except the intended challenge message. Further, we need two important properties. First, under the simulated setup, we want to guarantee that no other valid signatures exist for the challenge time step t^* except a unique signature for the challenge message alone (i.e., statistical unforgeability). Second, we want to guarantee that the joint distribution of the (punctured signing key, *simulated* verification key) pair is computationally indistinguishable from the joint distribution of (punctured signing key, *real* verification key).

We are not aware of any existing signature scheme that satisfies the above properties. Among these two properties, the first one is not too hard to achieve by modifying existing punctured signature schemes such as the one by Bellare et al. [BSW16] which is a modification of the signature scheme of Sahai and Waters [SW14]. However, the second property, which turns out essential for our final proof to work, requires non-trivial techniques to prove, and this is where we need the cryptographic primitives to satisfy sub-exponential security. We now define a SSU signature scheme more formally, and then describe a new construction that satisfies these security definitions.

Definition. An SSU signature scheme has similar syntax to a standard signature scheme, except that the signing and verification keys both take a counter t (i.e., time step) along with the message to be signed. We refer to t as the *round.* It contains the following algorithms:

- $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len})$: takes as input the security parameter 1^λ, the length of the round tlen, the length of the messages to be signed len, and outputs a signing key sk and a verification key vk.

- $\mathsf{sk}^* \leftarrow \mathbf{Puncture}(\mathsf{sk}, t^*, x^*)$: takes as input a signing key sk generated by the $\mathbf{Setup}$ algorithm, a round $t^* \in \{0,1\}^{\mathsf{tlen}}$ and message $x^* \in \{0,1\}^{\mathsf{len}}$, and outputs a punctured signing key sk^*.

[2]Looking ahead, our selective security notion for NIAR forces the adversary to commit to a challenge time step before receiving keys from the challenger. We defer the formal definition to Section 2.6.

- $\sigma \leftarrow \mathbf{Sign}(\mathsf{sk}, t, x)$: a *deterministic* algorithm that takes as input a signing key sk generated by **Setup** along with a round $t \in \{0,1\}^{\mathsf{tlen}}$ and a message $x \in \{0,1\}^{\mathsf{len}}$ and outputs a signature σ for x w.r.t. t.

- $\sigma \leftarrow \mathbf{PSign}(\mathsf{sk}^*, t, x)$: a *deterministic* algorithm that takes as input a punctured signing key sk^* along with a round t and a message x, and outputs a signature σ for x w.r.t. t.

- 0 or $1 \leftarrow \mathbf{Vf}(\mathsf{vk}, t, x, \sigma)$: takes as input a verification key vk, a round t, a message x, and a signature σ, and outputs 1 for accept and 0 for reject.

- $(\mathsf{sk}^*, \mathsf{vk}^*) \leftarrow \mathbf{SimSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*)$: takes as input the security parameter 1^λ, the length of the round tlen, the length of the messages to be signed len, a round $t^* \in \{0,1\}^{\mathsf{tlen}}$ and message $x^* \in \{0,1\}^{\mathsf{len}}$, and outputs a punctured signing key sk^*, and a punctured verification key vk^*.

For correctness, we want that for all (t^*, x^*) the **Sign** and **PSign** on their respective keys behave identically on all (t, x) *not of the form* $t = t^*$ and $x \neq x^*$. We defer the formal definition to Section 4 and focus on formally defining the required security properties next:

Definition 2.2 (Security for SSU Signatures)**.** An SSU signature is said to be secure if it has the following properties:

- **Computational indistinguishability of simulated setup.** For any $\lambda, \mathsf{len}, \mathsf{tlen} \in \mathbb{N}$, any $t^* \in \{0,1\}^{\mathsf{tlen}}$, any $x^* \in \{0,1\}^{\mathsf{len}}$, the following two probability ensembles indexed by λ are computationally indistinguishable:

  1. **Real**: Let $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len})$, $\mathsf{sk}' \leftarrow \mathbf{Puncture}(\mathsf{sk}, t^*, x^*)$, output $(\mathsf{sk}', \mathsf{vk})$.
  2. **Ideal**: Let $(\mathsf{sk}^*, \mathsf{vk}^*) \leftarrow \mathbf{SimSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*)$, and output $(\mathsf{sk}^*, \mathsf{vk}^*)$.

- **Statistical unforgeability at** (t^*, x^*)**.** For all $\lambda, \mathsf{len}, \mathsf{tlen} \in \mathbb{N}$, $t^* \in \{0,1\}^{\mathsf{tlen}}$, $x^* \in \{0,1\}^{\mathsf{len}}$,

$$\Pr\left[\begin{array}{ll} (\mathsf{sk}^*, \mathsf{vk}^*) & \leftarrow \mathbf{SimSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*): \\ & \exists\, \sigma \ \&\ x \neq x^* \text{ s.t. } \mathbf{Vf}(\mathsf{vk}^*, t^*, x, \sigma) = 1 \end{array}\right] = 0,$$

$$\Pr\left[\begin{array}{ll} (\mathsf{sk}^*, \mathsf{vk}^*) & \leftarrow \mathbf{SimSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*): \\ & \exists\, \sigma \neq \mathbf{PSign}(\mathsf{sk}^*, t^*, x^*) \text{ s.t. } \mathbf{Vf}(\mathsf{vk}^*, t^*, x^*, \sigma) = 1 \end{array}\right] = 0.$$

Special constrained PRF. To construct an SSU signature scheme satisfying the aforementioned properties, we will need a special constrained PRF scheme as a building block. A constrained PRF satisfies the following: given a PRF key sk and a constraint $\mathcal{C}$, it is possible to compute a constrained key sk^* that can evaluate the PRF only on points that satisfy the constraint $\mathcal{C}$. In terms of security, we require that even given the constrained key, the PRF evaluation on all points that do not satisfy the constraint remain pseudorandom. For our purposes, the constraint $\mathcal{C}$ can be specified by the tuple $(t^*, x^*, y) \in \{0,1\}^{\mathsf{tlen}} \times \{0,1\}^{\mathsf{len}} \times \{\{0,1\}^{\mathsf{len}} \cup \{2^{\mathsf{len}}\}\}$ where we say that (t, x) satifies the contraint only if

$$t \neq t^* \text{ or } ((t = t^*) \text{ and } (x \geq y \text{ or } x = x^*))$$

In Appendix A, we show how to instantiate such a constrained PRF from one-way functions, using ideas inspired by the well-known GGM construction [GGM84].

SSU signature construction. We construct an SSU signature scheme from the subexponential security of iO, a length-doubling *injective* PRG (which is implied by the existence of a one-way permutation), and a *constrained* PRF with domain $\{0,1\}^{\mathsf{tlen}} \times \{0,1\}^{\mathsf{len}}$.

Our starting point is the Sahai-Waters signature scheme [SW14] that satisfies computational unforgeability. Since we are interested in achieving the stronger SSU notion, we need to modify their construction and require subexponential security from our underlying primitives. More specifically, our construction is as follows:

- **Setup**$(1^\lambda, \mathsf{tlen}, \mathsf{len})$ samples a PRF key sk whose domain is the set $\{0,1\}^{\mathsf{tlen}} \times \{0,1\}^{\mathsf{len}}$ as the signing key, and the verification key is an iO obfuscation of a circuit $C[\mathsf{sk}]$ parametrized by sk defined as follows:

  <u>*Circuit* $C[\mathsf{sk}](t, x, \sigma)$</u>:

  if $G(\mathsf{PRF}(\mathsf{sk}, t, x)) = G(\sigma)$, output 1. Otherwise, output 0.

- **Sign**(sk, t, x) computes the signature σ as the evaluation $\mathsf{PRF}(\mathsf{sk}, t, x)$.

- **Vf**$(\mathsf{vk}, t, x, \sigma)$ treats vk as a program, and outputs $\mathsf{vk}(t, x, \sigma)$.

- **Puncture**(sk, t^*, x^*): Output a constrained PRF key sk^* for the key sk and the constraint $(t^*, x^*, 2^{\mathsf{len}})$.

- **PSign**(sk^*, t, x) computes the signature σ by evaluating the PRF on (t, x) using the constrained key sk^*.

- **SimSetup**$(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*)$ samples a PRF key sk and sets sk^* as the output of **Puncture**(sk, t^*, x^*), the punctured verification key vk^* is an iO obfuscation of the circuit $\widetilde{C}[\mathsf{sk}^*, t^*, x^*]$ parametrized by $(\mathsf{sk}^*, t^*, x^*)$ defined as follows:

  <u>*Circuit* $\widetilde{C}[\mathsf{sk}^*, t^*, x^*](t, x, \sigma)$</u>:

  – if $t = t^*$ and $x \neq x^*$, output 0.

  – else: if $G(\mathsf{PRF}.\mathbf{CEval}(\mathsf{sk}^*, t, x)) = G(\sigma)$, output 1, otherwise 0.

Proof Idea. First, let us discuss statistical unforgeability: if G is an injective PRG then our signature scheme is unique, that is, for all points (t, x) there exists exactly one signature that will be accepted by the verification key. So, there exists a unique signature for (t^*, x^*). This combined with the fact that vk^* rejects signatures for all points of the form $t = t^*$ and $x \neq x^*$ implies the required statistical unforgeability property. We empahsize that the required injective PRGs can be instantiated from OWPs via the Goldreich-Levin construction [GL89].

Showing indistinguishability of setups is more challenging. At a high level, to switch from vk to the punctured verification key vk^* requires 2^{len} number of steps where in each step we puncture the PRF key on a new message. To deal with such an exponential sequence of hybrids we require subexponential security from both iO and PRGs. The actual proof additionally needs to be deal with more subtleties including showing that all hybrids verification keys are succinct. We defer the formal proof to Section 4.

2.5 Our Router Construction

Our construction follows the "network of obfuscated circuits" approach described in Section 2.3. In particular, we need the following tools: (a) puncturable PRF that is used for encryption, (b) an SSU signature scheme to authenticate routes and messages, (c) an iO obfuscator, and (d) a routing network.

Notation. To describe our construction more formally, it will be helpful to introduce some notation for the routing network. Recall that a routing network for n senders and n receivers is a layered directed acyclic graph that has $O(\log n)$ layers numbered from $0, 1, \ldots, L$. Senders are assigned to the input layer (i.e., layer-0) and receivers are assigned to the output layer (i.e., layer-L). Let G be the number of gates contained in each of the $L-1$ intermediate layers. There are $(L-1) \cdot G$ gates overall, and we refer to the g-th gate in the ℓ-th layer by the tuple $(\ell, g) \in [L-1] \times [G]$. Let $W = O(\log^2 \lambda)$ be the number of incoming and outgoing wires in each gate. Overall, there are $L \times [2 \cdot n]$ wires where we index the i-th wire in the ℓ-th layer by the tuple $(\ell, i) \in [L] \times [2n]$.[3] We refer to the W incoming wires of every gate (ℓ, g) by the set $\mathsf{Input}_{(\ell,g)} \subseteq [2n]$ and the W outgoing wires by the set $\mathsf{Output}_{(\ell,g)} \subseteq [2n]$. Finally, recall that a route $\overline{\mathsf{rte}}_u$ from sender u to receiver v is a sequence of wires $(j_1, \ldots, j_L)$ where j_ℓ is a wire in the ℓ-th layer for all $\ell \in [L]$.

Next, we describe our routing scheme starting with the **Setup** algorithm.

Setup Algorithm. Given a routing permutation π, the **Setup** algorithm first runs the $\mathsf{AssignRoutes}$ algorithm to sample a set of edge-disjoint routes $\{\overline{\mathsf{rte}}_u\}_{u \in [n]}$ between each sender/receiver pair. Then, for every wire $(\ell, i) \in [L] \times [2n]$ in the routing network we sample (a) PRF key $k_{(\ell,i)}$ for encryption, (b) a signature key pair $(\mathsf{msk}_{(\ell,i)}, \mathsf{mvk}_{(\ell,i)}) \leftarrow \mathsf{Sig}.\mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len})$ for signing messages, and (c) a signature key pair $(\mathsf{rsk}_{(\ell,i)}, \mathsf{rvk}_{(\ell,i)}) \leftarrow \mathsf{Sig}.\mathbf{Setup}(1^\lambda, 0, \mathsf{len})$ for signing routes for $\mathsf{len} = \widetilde{O}_\lambda(1)$. Looking ahead, the route signatures keys for wires assigned to corrupt senders' routes, and the message signature keys for all other wires will be punctured to ensure "uniqueness of routes" and "uniqueness of plaintext" properties respectively.

Given the above set of keys, consider a sender/receiver pair (u, v) with route $\overline{\mathsf{rte}}_u = (j_1, \ldots, j_L)$. Then sender u's sender key ek_u and receiver v's decryption key rk_v are defined as follows:

$$\mathsf{ek}_u = \left(k_{(1,j_1)}, \mathsf{msk}_{(1,j_1)}, \mathsf{rte}_u = (\overline{\mathsf{rte}}_u, \overline{\mathsf{rsig}_u} = (\mathsf{rsig}_1, \ldots, \mathsf{rsig}_L))\right) \ , \mathsf{rk}_v = k_{(L,j_L)} \ ,$$

where rsig_ℓ is the signature on $\overline{\mathsf{rte}}_u$ computed using the route signing key $\mathsf{rsk}_{(\ell,j_\ell)}$.[4]

To define the routing token tk, we attribute each gate $(\ell, g) \in [L-1] \times [G]$ of the routing network with a circuit $\mathsf{Gate}_{(\ell,g)}$. Each of the circuits $\mathsf{Gate}_{(\ell,g)}$ take as input a time step t and a set of W ciphertexts (one per incoming wire in $\mathsf{Input}_{(\ell,g)}$), and outputs W output ciphertexts (one per outgoing wire in $\mathsf{Output}_{(\ell,g)}$). In Figure 1 we describe the circuit in more detail. The routing token tk is then defined as follows:

$$\mathsf{tk} = \{\mathsf{iO}(\mathsf{Gate}_{(\ell,g)}) : (\ell, g) \times [L-1] \times [G]\} \ .$$

We conclude the description of our scheme by discussing how encryption, routing and decryption work.

Encryption Algorithm. For a sender u to send a message x to its receiver for time step t, the sender first computes a message signature msig for the tuple (x, rte_u) for round t, and encrypts the tuple $(x, \mathsf{rte}_u, \mathsf{msig})$ using its PRF key.

[3] To be more precise there are $c \cdot n$ wires in each layer for constant $c \geq 2$, but for simplicity we assume $c = 2$ as this is achieved by our proposed instantiation.

[4] For route signature we let $\mathsf{tlen} = 0$, henceforth we will ignore the round parameter for route signatures.

Hardcoded values. $\mathsf{Gate}_{(\ell,g)}$ has hardcoded the following values:

- For each wire $i \in \mathsf{Input}_{(\ell,g)}$, the PRF key $k_{(\ell,i)}$, the message verification key $\mathsf{mvk}_{(\ell,i)}$ and the route verification key $\mathsf{rvk}_{(\ell,i)}$.
- For each wire $i \in \mathsf{Output}_{(\ell,g)}$ in layer $\ell+1$, the PRF key $k_{(\ell+1,i)}$ and the message signing key $\mathsf{msk}_{(\ell+1,i)}$.
- For the first layer $\ell = 1$, we also hardwire the set F_g which contains wires $i \in \mathsf{Input}_{(\ell,g)}$ such that $(1,i)$ was not assigned to any users' route.

Procedure. $\mathsf{Gate}_{(\ell,g)}$ takes as input a round t and a set of ciphertexts $\{\mathsf{CT}_{(\ell,i)} : i \in \mathsf{Input}_{(\ell,g)}\}$ corresponding to the input wires. It computes as follows.

1. For each input wire $i \in \mathsf{Input}_{(\ell,g)}$:
   (a) For the first layer (i.e., $\ell = 1$) if $i \in F_g$, we continue to the next i. //This is a filler element and is ignored.
   (b) **Decrypt and authenticate the message/route:**
      i. Decrypt $\mathsf{CT}_{(\ell,i)}$ by unmasking it with the value $\mathsf{PRF}(k_{(\ell,i)}, t)$. Let $(x, \mathsf{rte} = (\overline{\mathsf{rte}}, \overline{\mathsf{rsig}}), \mathsf{msig})$ be the plaintext.
      ii. Abort if msig is an insvalid signature of (x, rte) w.r.t. $\mathsf{mvk}_{(\ell,i)}$ and round t.
      iii. If $x = \mathsf{rte} = \bot_{\mathsf{filler}}$, go to the next i (It is a filler element and ignored).
      iv. Parse $\overline{\mathsf{rte}}$ as $(j_1, \ldots, j_L)$ and $\overline{\mathsf{rsig}} = (\mathsf{rsig}_1, \ldots, \mathsf{rsig}_L)$. Abort if either $j_\ell \neq i$ or the next hop $j_{\ell+1}$ is not in the set $\mathsf{Output}_{(\ell,g)}$ or rsig_ℓ is an invalid signature of $(j_1, \ldots, j_L)$ w.r.t. $\mathsf{rvk}_{(\ell,i)}$.
   (c) **Prepare the output ciphertext $\mathsf{CT}_{(\ell+1,j_{\ell+1})}$:**
      i. For convenience, set $j = j_{\ell+1}$.
      ii. If $\mathsf{CT}_{(\ell+1,j)}$ has already been computed, then abort.
      iii. For any intermediate layer (i.e., $\ell < L-1$), first compute a new message signature msig' on (x, rte) using $\mathsf{msk}_{(\ell+1,j)}$ and round t. Then, set $\mathsf{CT}_{(\ell+1,j)}$ as an encryption of the tuple $(x, \mathsf{rte}, \mathsf{msig}')$ by masking it with the value $\mathsf{PRF}(k_{(\ell+1,j)}, t)$.
      iv. For the output layer ($\ell = L-1$), $\mathsf{CT}_{(L,j)}$ is set to an encryption of the message x by masking it with the value $\mathsf{PRF}(k_{(L,j)}, t)$.
2. For each $j \in \mathsf{Output}_{(\ell,g)}$ such that $\mathsf{CT}_{(\ell+1,j)}$ has not been computed yet, compute filler ciphertexts:
   (a) Set $x = \mathsf{rte} = \bot_{\mathsf{filler}}$.
   (b) Compute msig' on the (x, rte) using the key $\mathsf{msk}_{(\ell+1,j)}$ for round t.
   (c) Set $\mathsf{CT}_{(\ell+1,j)}$ as an encryption of $(x, \mathsf{rte}, \mathsf{msig}')$ by masking it with the value $\mathsf{PRF}(k_{(\ell+1,j)}, t)$.
3. Output $\{\mathsf{CT}_{(\ell+1,i)} : i \in \mathsf{Output}_{(\ell,g)}\}$.

Figure 1: The circuit $\mathsf{Gate}_{(\ell,g)}$.

Enc(ek_u, x_u, t) on input user u's encryption key ek_u and plaintext x_u and the round t, does the following:

1. Parse ek_u as $(k, \mathsf{msk}, \mathsf{rte}_u)$.
2. Compute a signature msig of (x_u, rte_u) with the key msk for round t.
3. Compute the ciphertext CT_u as an encryption of $(x_u, \mathsf{rte}_u, \mathsf{msig})$ by masking it with the value $\mathsf{PRF}(k, t)$.
4. Output CT_u along with the first wire i specified in rte.

Routing Algorithm. Then, in each time step, the router collects all n ciphertexts all applies the obfuscated gate layer-by-layer to compute n ciphertexts for the receivers. More formally,

Rte$(\mathsf{tk}, t, (\mathsf{CT}_1, i_1), (\mathsf{CT}_2, i_2), \ldots, (\mathsf{CT}_n, i_n))$ on input the router token tk along with the round number t, ciphertexts $\mathsf{CT}_1, \ldots, \mathsf{CT}_n$ along with wire indices $i_1, \ldots, i_n$ for round t, does the following:

1. Parse $\mathsf{tk} = \{\overline{\mathsf{Gate}}_{(\ell,g)} : \ell \in [L-1], g \in [G]\}$.
2. Compute ciphertexts for the input layer:
   (a) For all $k \in [n]$, set $\mathsf{CT}_{(1,i_k)} = \mathsf{CT}_k$. // *Real ciphertexts*
   (b) For each $i \in [2n]$ where $\mathsf{CT}_{(1,i)}$ is not defined, set $\mathsf{CT}_{(1,i)} = \perp_{\mathsf{filler}}$. // *Filler ciphertexts*
3. Compute network of iO obfuscated gates layer-by-layer. That is, for layer $\ell \in 1, \ldots, L-1$, evaluate all the obfuscated gates at this layer as follows. For each $g \in [G]$, let $\mathsf{Input}_{(\ell,g)}$ and $\mathsf{Output}_{(\ell,g)}$ be the set of input and output wires of gate $\mathsf{Gate}_{(\ell,g)}$. Then, evaluate the circuit
$$\{\mathsf{CT}_{(\ell+1,i)} : i \in \mathsf{Output}_{(\ell,g)}\} = \overline{\mathsf{Gate}}_{(\ell,g)}(t, \{\mathsf{CT}_{(\ell,i)} : i \in \mathsf{Input}_{(\ell,g)}\}).$$
4. Output $(\mathsf{CT}'_1 = \mathsf{CT}_{(L,1)}, \mathsf{CT}'_2 = \mathsf{CT}_{(L,3)}, \ldots, \mathsf{CT}'_n = \mathsf{CT}_{(L,2n-1)})$.

Decryption Algorithm. A receiver v learns its intended message by just decrypting the received ciphertext using its PRF key. More formally,

Dec$(\mathsf{rk}_u, \mathsf{CT}'_u, t)$ on input user u's receiver key rk_u, output ciphertext CT_u, and a time step t, does the following: Decrypt CT'_u with the PRF key rk_u by unmasking it with the value $\mathsf{PRF}(\mathsf{rk}_u, t)$. Output the obtained plaintext y.

Efficiency Analysis. Before we discuss the security proof, we address the efficiency of the above scheme. First, observe that for each time step, the router computes each of the obfuscated circuits at most once. Second, observe that the Gate circuit described above is of size $\mathsf{poly}(\lambda, \log n)$: the hardwired keys, the size of the ciphertexts are $\mathsf{poly}(\lambda, \log n)$, hence operating over them requires at most $\mathsf{poly}(\lambda, \log n)$ size. Then, accounting for the polynomial blowup of the iO obfuscator, we can conclude that the router can run each obfuscated circuit in time $\mathsf{poly}(\lambda, \log n)$. Since there are at most $\widetilde{O}(n)$ gates, we can conclude that the router computation per time step is bounded by $\widetilde{O}_\lambda(n)$ where $\widetilde{O}_\lambda$ hides $\mathsf{poly}(\lambda, \log n)$ factors for some fixed $\mathsf{poly}(\cdot)$. Secondly, notice that each receiver's key contains a PRF key which is $O_\lambda(1)$ in size. Sender key size is bounded by the size of the route which is $\widetilde{O}_\lambda(1)$. Finally, per sender communication per time step is $\widetilde{O}_\lambda(1)$.

2.6 Defining Selectively Secure NIAR

We prove that our NIAR scheme satisfies selective security. Before we give a proof roadmap, we first define selective security — it turns out that even the definition of selective security has non-trivial technicalities.

First attempt. The first attempt is to just extend the full security definition (Definition 2.1) by requiring the adversary to commit to a challenge time step t^* upfront, as well as two challenge plaintext vectors $\{x^{(0)}_{u,t^*}\}_{i\in\mathcal{H}_S}$ and $\{x^{(1)}_{u,t^*}\}_{i\in\mathcal{H}_S}$. Unfortunately, upon closer examination, this approach does not actually constitute any relaxation — the resulting definition would still be equivalent to full security. Intuitively, the reason is that the adversary is still able to submit plaintext vectors corresponding to the two different worlds $b = 0$ and $b = 1$ in all non-challenge time steps, and it need not rely on the challenge time step to differentiate between the two worlds.

Recognizing that the naïve approach does not actually constitute any relaxation, we want to have a *single-challenge* version of the definition where only in the challenge time step can the adversary submit plaintext vectors of both worlds. In all other time steps, the adversary should submit only one plaintext vector. It turns out non-trivial to define a well-formed, single-challenge notion. To make such a definition well-formed, we have to first introduce a simulated setup algorithm which is never used in the real-world, but needed for defining selective security.

Simulated setup algorithm. As mentioned, we need to introduce a simulated setup algorithm for defining a meaningful selective relaxation. Specifically, the simulated setup algorithm $\mathbf{Setup}^*(1^\lambda$, tlen, len, n, t^*, $\pi^{(0)}$, $\pi^{(1)}$, $\mathcal{K}_S$, $\{x^{(0)}_{u,t^*}, x^{(1)}_{u,t^*}\}_{u\in\mathcal{H}_S})$ takes in the set of corrupt senders $\mathcal{K}_S$, the challenge time step t^*, both permutations $\pi^{(0)}, \pi^{(1)}$, as well as the two challenge plaintexts $\{x^{(0)}_{u,t^*}, x^{(1)}_{u,t^*}\}_{u\in\mathcal{H}_S})$, and it outputs $\{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}$, $\{\mathsf{ek}^{(0)}_u\}_{u\in\mathcal{H}_S}$, $\{\mathsf{ek}^{(1)}_u\}_{u\in\mathcal{H}_S}$, $\{\mathsf{rk}_u\}_{u\in[n]}$, and tk. Importantly, observe that a single set of corrupt sender keys $\{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}$ and the routing token tk must be simultaneously compatible with two different sets of honest sender keys $\{\mathsf{ek}^{(0)}_u\}_{u\in\mathcal{H}_S}$ and $\{\mathsf{ek}^{(1)}_u\}_{u\in\mathcal{H}_S}$ corresponding to the two worlds $b = 0$ and $b = 1$, respectively, as long as the corrupt senders have the same destinations in $\pi^{(0)}$ and $\pi^{(1)}$. More precisely, we want that as long as the corrupt senders have the same destinations in $\pi^{(0)}$ and $\pi^{(1)}$, it must be that

- for either $b \in \{0, 1\}$, the terms $\left(\{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{ek}^{(b)}_u\}_{u\in\mathcal{H}_S}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}\right)$ output by $\mathbf{Setup}^*$ have the same joint distribution as the output of the real $\mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, n, \pi^{(b)})$.

With this additional simulated setup algorithm, we are ready to define a meaningful selective relaxation. Consider the following single-challenge, selective security experiment.

Selective security experiment $\mathsf{SelSingleCh}^{b,\mathcal{A}}(1^\lambda, \mathsf{tlen}, \mathsf{len})$.

- $n, \mathcal{K}_S, \mathcal{K}_R, \pi^{(0)}, \pi^{(1)}, t^*, \{x^{(0)}_{u,t^*}, x^{(1)}_{u,t^*}\}_{u\in\mathcal{H}_S} \leftarrow \mathcal{A}(1^\lambda, \mathsf{tlen}, \mathsf{len})$;

- $(\{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{ek}_u^{(0)}\}_{u\in\mathcal{H}_S}, \{\mathsf{ek}_u^{(1)}\}_{u\in\mathcal{H}_S}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk})$
  $\leftarrow \mathbf{Setup}^*(1^\lambda, \mathsf{tlen}, \mathsf{len}, n, t^*, \pi^{(0)}, \pi^{(1)}, \mathcal{K}_S, \{x_{u,t^*}^{(0)}, x_{u,t^*}^{(1)}\}_{u\in\mathcal{H}_S})$;
- $\perp \leftarrow \mathcal{A}(\mathsf{tk}, \{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{rk}_u\}_{u\in\mathcal{K}_R})$;
- For $t = 1, 2, \ldots$:
  - if $t \neq t^*$: $(\{x_{u,t}\}_{u\in\mathcal{H}_S}, \delta_t) \leftarrow \mathcal{A}(\perp)$, and for $u \in \mathcal{H}_S$, let $\mathsf{CT}_{u,t} := \mathbf{Enc}(\mathsf{ek}_u^{(\delta_t)}, x_{u,t}, t)$;
  - else if $t = t^*$: for $u \in \mathcal{H}_S$, let $\mathsf{CT}_{u,t^*} := \mathbf{Enc}(\mathsf{ek}_u^{(b)}, x_{u,t^*}^{(b)}, t^*)$;
  - $\perp \leftarrow \mathcal{A}(\{\mathsf{CT}_{u,t-1}\}_{u\in\mathcal{H}_S})$;

The adversary is said to be admissible, iff with probability 1, $\mathsf{Leak}^*(\pi^{(0)}, \mathcal{K}_S, \mathcal{K}_R, \{x_{u,t^*}^{(0)}\}_{u\in\mathcal{H}_S}) = \mathsf{Leak}^*(\pi^{(1)}, \mathcal{K}_S, \mathcal{K}_R, \{x_{u,t^*}^{(1)}\}_{u\in\mathcal{H}_S})$ where where the function Leak^* $(\pi, \mathcal{K}_S, \mathcal{K}_R), \{x_{u,t^*}\}_{u\in\mathcal{H}_S})$ contains the destination of each corrupt sender and the contents of the messages from honest senders to corrupt receivers, as defined below:

$$\mathsf{Leak}^*(\pi, \mathcal{K}_S, \mathcal{K}_R, \{x_{u,t^*}\}_{u\in\mathcal{H}_S}) := (\{(u, \pi(u))\}_{u\in\mathcal{K}_S}, \{(u, x_{\pi^{-1}(u),t^*})\}_{u\in\mathcal{K}_R, \pi^{-1}(u)\in\mathcal{H}_S})$$

Intuitively, the admissibility rule requires that the corrupt senders have the same destinations in the two worlds, and that corrupt receivers receive the same messages from honest senders in the two worlds.

Definition 2.3 (Selective security for NIAR)**.** We say that a NIAR scheme satisfies selective (single-challenge) security (with receiver insider protection), iff for any non-uniform p.p.t. admissible adversary $\mathcal{A}$, $\mathcal{A}$'s views in $\mathsf{SelSingleCh}^{0,\mathcal{A}}(1^\lambda, \mathsf{tlen}, \mathsf{len})$ and $\mathsf{SelSingleCh}^{1,\mathcal{A}}(1^\lambda, \mathsf{tlen}, \mathsf{len})$ are computationally indistinguishable.

To better understand this definition, it helps to think about an *adaptive* single-challenge counterpart which is very similar to the above selective single-challenge definition, except that the adversary need not commit to the challenge time step t^* and the challenge honest plaintexts ahead of time. In our subsequent technical sections, we prove the following lemma in Appendix B, which says that adaptive, single challenge security implies full security (Definition 2.1).

Lemma 2.4. *A NIAR scheme that is secure in the adaptive, single-challenge setting is also secure by Definition 2.1.*

Because our single-challenge, selective notion is clearly a relaxation of the adaptive, single-challenge notion, it is also a meaningful relaxation of Definition 2.1.

2.7 Proof Overview

We now give an informal proof overview of our NIAR scheme. The formal hybrids and proofs can be found in Section 5.1. The selective security proof is with respect to a **Setup*** algorithm. So, let us introduce it first.

Setup* **Algorithm.** Given sets of corrupt senders and receivers $\mathcal{K}_S, \mathcal{K}_R$, challenge round t^*, routing permutations $\pi^{(0)}, \pi^{(1)}$, and challenge messages $\{x_{u,t^*}^{(0)}\}_{u\in\mathcal{H}_S}$, $\{x_{u,t^*}^{(1)}\}_{u\in\mathcal{H}_S}$, the **Setup*** algorithm first runs the $\mathsf{AssignRoutes}^*$ algorithm to sample a set of edge-disjoint routes $(\{\overline{\mathsf{rte}}_u\}_{u\in\mathcal{K}_S}, \{\overline{\mathsf{rte}}_u^{(0)}, \overline{\mathsf{rte}}_u^{(1)}\}_{u\in\mathcal{H}_S})$ between each sender/receiver pair. Then, for every wire in the routing network, PRF key, message signature key pair, route signature key pair are sampled as in **Setup**.

For $\beta \in \{0,1\}$, set the sender keys as follows: For each $u \in \mathcal{K}_S$, set $\mathsf{ek}_u = (k_{(1,j_1)}, \mathsf{msk}_{(1,j_1)}, \mathsf{rte}_u)$. For each $u \in \mathcal{H}_S$ and $\beta \in \{0,1\}$, set $\mathsf{ek}_u^{(\beta)} = (k_{(1,j_1^{(\beta)})}, \mathsf{msk}_{(1,j_1^{(\beta)})}, \mathsf{rte}_u^{(\beta)})$. For each $v \in [n]$, set the receiver keys $\mathsf{rk}_v = k_{(L,2v-1)}$. The routing token tk is defined the same way as in **Setup**.

Indistinguishability of Setup and Setup*. If the routing network satsifies obliviousness as defined above, then, for either $b \in \{0,1\}$, the terms $\left(\{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{ek}_u^{(b)}\}_{u\in\mathcal{H}_S}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}\right)$ output by **Setup**$^*(1^\lambda, \mathsf{tlen}, \mathsf{len}, n, t^*, \pi^{(0)}, \pi^{(1)}, \mathcal{K}_S, \{x_{u,t^*}^{(0)}, x_{u,t^*}^{(1)}\}_{u\in\mathcal{H}_S})$ as described above have the same joint distribution as the output of the real **Setup**$(1^\lambda, \mathsf{tlen}, \mathsf{len}, n, \pi^{(b)})$ as described above. This can be proven by creating a simple reduction such that if for any $b \in \{0,1\}$ the aforementioned joint distributions of **Setup** and **Setup*** could be distinguished with some noticeable probability, then, the reduction can distinguish the following two distributions with noticeable probability and thus break the obliviousness of the routing network.

- $\{(\{\mathsf{rte}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{rte}_u\}_{u\in\mathcal{H}_S}) : (\mathsf{rte}_1, \ldots, \mathsf{rte}_n) \leftarrow \mathsf{AssignRoutes}(1^\lambda, n, \pi^{(b)})\}$
- $\left\{ \begin{array}{l} \left(\{\mathsf{rte}_u\}_{u\in\mathcal{K}_S}, \left\{\mathsf{rte}_u^{(b)}\right\}_{u\in\mathcal{H}_S}\right) : \\ \left(\{\mathsf{rte}_u\}_{u\in\mathcal{K}_S}, \left\{\mathsf{rte}_u^{(0)}, \mathsf{rte}_u^{(1)}\right\}_{u\in\mathcal{H}_S}\right) \leftarrow \mathsf{AssignRoutes}^*(1^\lambda, n, \pi^{(0)}, \pi^{(1)}, \mathcal{K}_S) \end{array} \right\}$

Notation. Let $\{\overline{\mathsf{rte}}_u\}_{u\in\mathcal{K}_S}$ be routes sampled for corrupt senders' and let $\{\overline{\mathsf{rte}}_u^{(b)}\}_{u\in\mathcal{H}_S}$ be routes sampled for honest senders' both consistent with the permutation $\pi^{(b)}$. We define the notion of a *corrupt*, *honest* and *filler* wires w.r.t. the sampled routes. A wire (ℓ, i) is corrupt if it assigned by **Setup*** to a corrupt senders' route $\overline{\mathsf{rte}}_u$, analogously a wire assigned to honest senders' route $\overline{\mathsf{rte}}_u^{(b)}$ is referred to as honest wire, the rest are referred to as filler wires. We emphasize that it is possible for a wire to be an honest wire in $b = 0$ but act as a filler in $b = 1$, and vice versa. However, if a wire is corrupt in $b = 0$ then it is also corrupt in $b = 1$. Next, we define the notion of an "expected route" for a honest/filler wire (ℓ, j) w.r.t. the challenge time step t^* and challenge bit b. This is exactly the route that contains the wire (ℓ, j) as computed by **Setup***. Similarly, we define the notion of an "expected plaintext" for a filler/honest wire (ℓ, j) w.r.t. the challenge time step t^* and challenge bit b. This is exactly the plaintext that would appear on the wire had the router, after getting honest senders' ciphertexts for the challenge time step, run **Rte** honestly. For example, if (ℓ, j) was assigned by **Setup*** to $\overline{\mathsf{rte}}_u^{(b)}$ for honest sender u then the expected plaintext for this wire is the tuple $(x_{u,t^*}^{(b)}, \mathsf{rte}_u^{(b)})$ where $\mathsf{rte}_u^{(b)} = (\overline{\mathsf{rte}}_u^{(b)}, \overline{\mathsf{rsig}})$ was computed by **Setup***. In fact, the notion of "expected plaintext" can be generalized to the notion of "expected ciphertexts" as for a plaintext tuple $(x_{u,t^*}^{(b)}, \mathsf{rte}_u^{(b)}, \mathsf{msig})$ and time step t, the unique ciphertext encrypting the plaintext tuple using a PRF key k is $(x_{u,t^*}^{(b)}, \mathsf{rte}_u^{(b)}, \mathsf{msig}) \oplus \mathsf{PRF}(k, t^*)$.

Intuition of Hybrids. We are now ready to discuss our sequence of hybrids. The high level idea is to start with the real world hybrid $\mathsf{Hyb}_0^{(b)}$ where the challenge bit is b and go through a sequence of transitions to arrive at a hybrid $\mathsf{Hyb}_9^{(b)}$ in which the adversary $\mathcal{A}$'s view contains no information about the challenge bit b, that is, $\mathsf{Hyb}_9^{(0)}$ and $\mathsf{Hyb}_9^{(1)}$ are identical.

In the real world hybrid $\mathsf{Hyb}_0^{(b)}$, information about the challenge bit b is present as follows: the challenger provides the challenge ciphertexts $\mathsf{CT}_{u,t^*} = (x_{u,t^*}^{(b)}, \mathsf{rte}^{(b)}, \mathsf{msig}^{(b)}) \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell,i)}, t^*)$ for all $u \in \mathcal{H}_S$ to the adversary.

To remove information about the bit b, we want to invoke semantic security of the encryption scheme and change ciphertext to random values. Towards this end, we need to overcome a number of issues.

- As ciphertext computation is done by XORing with some PRF output, hence, to invoke semantic security we will puncture the PRF keys for filler/honest wires at t^* so as to use the pseudorandomness of PRF at punctured points property for invoking semantic security.
- After semantic security is invoked, it would not be possible to perform message authentication checks inside the circuits for routing network gates. So, instead of decrypting and performing message authentication checks, we want to directly do ciphertext comparison.
- In order to hope for ciphertext comparison, not only do we need the encryptions to be unique, we also need that only one particular plaintext (the expected plaintext) is possible on a filler/honest wire. Towards this end, we want to puncture the message signing keys to only sign the expected message at the challenge time step t^* and the message verification keys to only accept the expected message at the challenge timestep t^*. But, doing this is not straightforward and needs careful transitions in the proof as discussed next.
- In an obfuscated gate, message verification keys are present for all the input wires and message signing keys are present for all output wires. Changing the keys to be punctured is a two step process. For SSU signatures, the computational indistinguishability of simulated setup assumes that the message signing key has already been punctured. So, the first step is to only puncture the message signing key at challenge message and challenge round t^* and the second step is to also puncture out the message verification key. For the second step, while the circuit functionality is changed, it is still okay as we only rely on the computational indisntiguishability of simulated setup of SSU signatures and not on the security of indistinguishability obfuscation. But, this is not the case for the first step: message signing keys are present inside the obfuscated gates, hence, they may potentially change the circuit functionality. But for the security, all we can rely on is the security of indistinguishability obfuscation. Hence, it is imperative that we ensure the circuit functionality is preserved when we puncture the message signing keys.
- If we try to puncture out all the message signing keys of filler/honest wires in all layers in a single shot, then, clearly the gate's functionality is not preserved as the gate with punctured keys cannot sign messages other than the expected message for the challenge timestep t^*. So, how do we resolve this dilemma? We take a layered approach based on the key observation that none of the message signing keys for the first layer for the filler/honest wires are in the view of the adversary. This is because the first layer of obfuscated gates contains the message signing keys for the second layer and subsequently layer ℓ gates contain message signing keys for layer $\ell + 1$.

Based on the above issues, we come up with the following sequnece of hybrids:

Hybrid $\mathsf{Hyb}_0^{(b)}$: This is the real world experiment $\mathsf{SelSingleCh}^{b,\mathcal{A}}$.

Hybrids $\mathsf{Hyb}_1^{(b)}, \mathsf{Hyb}_2^{(b)}, \mathsf{Hyb}_3^{(b)}$: The foremost change we make is to ensure that no subversion of messages from corrupt to filler/honest wire is possible by the adversary. This property will be necessary later on in hybrids $\mathsf{Hyb}_{6,\ell,3}^{(b)}$ and $\mathsf{Hyb}_{6,\ell,5}^{(b)}$ for all $\ell \in [L-1]$ as will be highlighted below. This change is accomplished in three steps - (i) puncture the route signing keys for all the corrupt wires at the routes sampled by **Setup*** for those wires respectively, (ii) puncture the route verification keys for all the corrupt wires, (iii) hardwire the expected route and expected route signatures and update the route authentication to directly compare routes and route signatures against hardcoded values.

Hybrid $\mathsf{Hyb}_4^{(b)}$: Puncture out the message signing keys for all filler/honest wires of layer $\ell = 1$. Then, the adversary's view is identical as before (formal argument can be found in the transition from $\mathsf{Hyb}_3^{(b)}$ to $\mathsf{Hyb}_4^{(b)}$ and Claim 5.8).

Hybrids $\mathsf{Hyb}_5^{(b)}, \mathsf{Hyb}_{6,1,1}^{(b)}, \mathsf{Hyb}_{6,1,2}^{(b)}$: Puncture out the message verficitaion keys for all filler/honest wires of layer $\ell = 1$ ($\mathsf{Hyb}_5^{(b)}$), then, hardcode the expected message signatures and messages ($\mathsf{Hyb}_{6,1,1}^{(b)}$) and then go on to hardcode the expected input ciphertext and puncture the PRF keys ($\mathsf{Hyb}_{6,1,2}^{(b)}$).

Hybrid $\mathsf{Hyb}_{6,1,3}^{(b)}$: Once this is done, we are ready to try to puncture out the message signing keys for all filler/honest wires of layer $\ell = 2$ ($\mathsf{Hyb}_{6,1,3}^{(b)}$). These are hardwired in the gates in the first layer $\ell = 1$. Even though these punctured keys cannot sign messages other than the challenge messages for the challenge round t^*, now unlike the single shot approach, we argue that no other message for the challenge round t^* could have ever been signed by the unpunctured signing keys for layer $\ell = 2$ in the previous hybrid $\mathsf{Hyb}_{6,1,2}^{(b)}$. This is for a couple of reasons. Firstly, if a message is being sent from filler/honest wire $(1, i)$ to filler/honest wire $(2, j)$, then, the punctured route verification key for layer $\ell = 1$ would have ensured that for the challenge round it was indeed the challenge message. Secondly, if the adversary is trying to subvert a message from a corrupt wire in layer $\ell = 2$ to a filler/honest wire $(2, j)$, then, this is not possible because of the first set of changes we made ($\mathsf{Hyb}_1^{(b)}, \mathsf{Hyb}_2^{(b)}, \mathsf{Hyb}_3^{(b)}$). Consequently, the security of indistinguishability obfuscation can be invoked to transition from $\mathsf{Hyb}_{6,1,2}^{(b)}$ to $\mathsf{Hyb}_{6,1,3}^{(b)}$ (formal proof in Claim 5.12).

Hybrids $\mathsf{Hyb}_{6,1,4}^{(b)}, \mathsf{Hyb}_{6,1,5}^{(b)}$: Similar to $\mathsf{Hyb}_4^{(b)}$, we can puncture out the message verficitaion keys for all filler/honest wires of layer $\ell = 2$ ($\mathsf{Hyb}_{6,1,4}^{(b)}$) and then, hardcode the expected message signatures and messages and then go on to hardcode the expected output ciphertext and puncture the PRF keys ($\mathsf{Hyb}_{6,1,5}^{(b)}$).

Cascading effect. Once layer $\ell = 2$ is done, similiar arguments can be made for layer $\ell = 3$ and so on. This cascading effect carries on for all the rest of the layers and we make simiar changes one layer at a time to finally arrive at hybrid $\mathsf{Hyb}_{6,L-1,5}^{(b)}$.

Hybrid $\mathsf{Hyb}_7^{(b)}$: In this hybrid, we invoke the pseudorandomness of PRF at punctured points to change all the hardcoded ciphertexts to be uniformly random values except for a select few following wires. If an honest wire in the last layer has the destination that is corrupt, then, we do not make any change to the hardwired outgoing ciphertexts.

Hybrids $\mathsf{Hyb}_8^{(b)}, \mathsf{Hyb}_9^{(b)}$: In $\mathsf{Hyb}_7^{(b)}$, the only sources of information of challenge bit b are the hardwired message signing and verification keys for all filler/honest wires in all the circuits. So, in these hybrids, we unpuncture all the message signing and verification keys.

Analysis of the final hybrid: In hybrid $\mathsf{Hyb}_9^{(b)}$, we claim that everything can be simulated from the leakage function which is identical in both worlds. In other words, this hybrid contains no information about the challenge bit b. This is not too difficult to see. Observe that for all corrupt wires, while the punctured route verification keys and expected routes are hardwired in the obfuscated circuits, they are the same across the two worlds $b = 0$ and $b = 1$. Further, punctured

PRF keys that are hardwired contain no information about b. Most hardwired ciphertexts in the obfuscated gates are uniformly random values and the ones that are not random (honest wires with corrupt receiver as destination) have the same value across the two worlds by the admissibility criteria. Lastly, for the challenge round t^*, the circuit description (Figure 13) treats filler and honest wires exactly the same. Hence, $\mathsf{Hyb}_9^{(0)}$ and $\mathsf{Hyb}_9^{(1)}$ are identical. Formal arguments for this can be found in Claim 5.18.

3 Preliminaries

In this section we are going to define the building blocks necessary for our NIAR construction.

3.1 Notations

We use '_' to denote that a value is irrelevant. For instance, in $(a, \_, c)$ the second value is irrelevant and can be anything. Often times, we use a short hand $\{y_i : i \in [n]\}$ to denote an ordered sequence $(y_1, \ldots, y_n)$. For instance $\{y_i : i \in [n]\} \leftarrow f(t, \{x_i : i \in [n]\})$ means $(y_1, \ldots, y_n) \leftarrow f(t, x_1, \ldots, x_n)$.

Notation for subexponential security. In this paper, whenever we say that a cryptographic building block satisfies subexponential security, we only need the version where the adversary is *probabilistic polynomial-time* (p.p.t.), but the security failure probability is *subexponentially small*. More specifically, we want that there exists $0 < \gamma < 1$ such that for any *probabilistic polynomial-time* (p.p.t.) time adversary $\mathcal{A}$, there is some sufficiently large security parameter λ such that the probability that $\mathcal{A}$ breaks the the scheme's security is upper bounded by $2^{-\lambda^\gamma}$.

Throughout the paper, we often say that some cryptographic building block *satisfies subexponential security w.r.t. the parameter* $0 < \gamma < 1$, to explicitly denote the γ parameter in the above notion.

3.2 Correctness of NIAR

The syntax and security of NIAR was defined earlier in Section 2.1 and Section 2.6. For completeness, we define correctness below. 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 $\lambda \in \mathbb{N}$, any $(x_1, x_2, \ldots, x_n) \in \{0,1\}^n$, and any $t \in \{0,1\}^{\mathsf{tlen}}$: let $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, n, \pi)$, let $\mathsf{CT}_{u,t} \leftarrow \mathbf{Enc}(\mathsf{ek}_u, x_u, t)$ for $u \in [n]$, let $(\mathsf{CT}'_{1,t}, \mathsf{CT}'_{2,t}, \ldots, \mathsf{CT}'_{n,t}) \leftarrow \mathbf{Rte}(\mathsf{tk}, \mathsf{CT}_{1,t}, \mathsf{CT}_{2,t}, \ldots, \mathsf{CT}_{n,t})$, and let $x'_u \leftarrow \mathbf{Dec}(\mathsf{rk}_u, \mathsf{CT}'_{u,t})$ for $u \in [n]$; it must be that $x'_{\pi(u)} = x_u$ for every $u \in [n]$.

3.3 Routing Networks

Imagine we have a direct acyclic graph henceforth called a *routing network* with $2n$ sources and $2n$ destinations. Suppose we have n *producers*, each of whom assigned to a distinct source vertex. We also have n *consumers*, each of whom assigned to a distinct destination vertex. Now, each producer wants to route one product to a distinct consumer, and the desired mapping between the producers and consumers is called the routing permutation π. To avoid congestion, we want all n routes to be over *edge-disjoint* paths. Earlier works [ACN+20, RS21] have constructed such a routing network with the following properties.

- *Congestion-free routing.* There exists a randomized algorithm $\mathsf{AssignRoutes}(1^\lambda, n, \pi)$ that takes in the security parameter λ and the routing permutation π, and with $1 - \mathsf{negl}(\lambda)$ probability, outputs the following information for each producer $u \in [n]$: 1) which source vertex the producer u is mapped to; and 2) the path that producer u traverses to reach its consumer which is assigned to some destination vertex. Henceforth, the above information is called the *route* for producer u, often denoted rte_u. As mentioned, all producers' routes are edge-disjoint. We allow the $\mathsf{AssignRoutes}(1^\lambda, n, \pi)$ algorithm to have a negligibly small failure probability in which case it outputs $\perp$.

- *Layered construction.* The network is layered. We may imagine that the source and destination vertices form two special layers numbered 0 and L, respectively, and all other intemediate-layer vertices are henceforth called *gates*. Directed edges, henceforth called *wires*, exist only between adjacent layers ℓ and $\ell + 1$.

- *Efficiency.* The network has $O(\log n)$ layers, and each intermediate layer has at most $O(n/\log^2 \lambda)$ gates. Each gate has $O(\log^2 \lambda)$ incoming wires and $O(\log^2 \lambda)$ outgoing wires. Thus, the number of wires between any two adjacent layers is guaranteed to be $O(n)$.

- *Obliviousness.* The network and the corresponding $\mathsf{AssignRoutes}(1^\lambda, \pi)$ algorithm satisfies a privacy property. Informally speaking, imagine that a subset of the producers are corrupt, and they can learn their routes to their respective destinations (including which source nodes the corrupt producers are assigned to). We want that the choice of the corrupt producers' routes are independent of the honest producers' destinations. We will formally define this privacy property below.

Definition 3.1 (Obliviousness of a routing network)**.** We say that a routing network satisfies obliviousness, iff there exists another simulated $\mathsf{AssignRoutes}^*$ algorithm and a negligible function $\mathsf{negl}(\cdot)$, such that for any corrupt set $\mathcal{K} \subseteq [n]$ of producers, for any two routing permutations π_0 and π_1 such that $\pi_0(i) = \pi_1(i)$ for any $i \in \mathcal{K}$, for either $b \in \{0, 1\}$,

$$\{(\{\mathsf{rte}_u\}_{u\in\mathcal{K}}, \{\mathsf{rte}_u\}_{u\notin\mathcal{K}}) : (\mathsf{rte}_1, \ldots, \mathsf{rte}_n) \leftarrow \mathsf{AssignRoutes}(1^\lambda, n, \pi_b)\}$$

$$\approx_{\mathsf{negl}(\lambda)}$$

$$\left\{\left(\{\mathsf{rte}_u\}_{u\in\mathcal{K}}, \left\{\mathsf{rte}_u^b\right\}_{u\notin\mathcal{K}}\right) : \left(\{\mathsf{rte}_u\}_{u\in\mathcal{K}}, \left\{\mathsf{rte}_u^0, \mathsf{rte}_u^1\right\}_{u\notin\mathcal{K}}\right) \leftarrow \mathsf{AssignRoutes}^*(1^\lambda, n, \pi_0, \pi_1, \mathcal{K})\right\},$$

where $\approx_{\mathsf{negl}(\lambda)}$ means that the left-hand-side and the right-hand-side have statistical difference at most $\mathsf{negl}(\lambda)$.

The definition says that the simulated $\mathsf{AssignRoutes}^*$ algorithm should take in two routing permutations π_0 and π_1 that are consistent regarding all corrupt producers' destinations. Now, $\mathsf{AssignRoutes}^*$ must output two sets of routes for π_0 and π_1 respectively, such that the routes for the corrupt producers are shared between the both routing permutations π_0 and π_1, and then the routes for honest producers are generated in a compatible way for each of π_0 and π_1. Not only so, for either $b \in \{0, 1\}$, the union of the corrupt producers' routes $\{\mathsf{rte}_u\}_{u\in\mathcal{K}}$ and the honest producers' routes $\{\mathsf{rte}_u^b\}_{u\notin\mathcal{K}}$ output by the simulated $\mathsf{AssignRoutes}^*$ must be indistinguishable from running the real-world $\mathsf{AssignRoutes}$ using permutation π_b. Intuitively, the definition wants that the generation of the corrupt producers' routes be decomposed from the honest producers' destinations. In this sense, the corrupt producers' routes do not leak information about honest producers' destinations (beyond what is already leaked by the corrupt producers' destinations, and barring a negligibly small statistical difference).

Remark 3.2. *Our definition of obliviousness is not the same as the "data obliviousness" notion of Asharov et al. [ACN+20] and Ramachandran and Shi [RS21] — their notion requires that the access patterns of the routing algorithm not depend on the input data when executed on a RAM. On the other hand, our notion is closely related to a line of work called "oblivious routing" from the standard algorithms literature [RÖ2, HKLR05], where roughly speaking, we want that a player's route be independent of others' destinations. Interestingly, it turns out that the bucket-based butterfly network construction of Asharov et al. [ACN+20] and Ramachandran and Shi [RS21] also satisfies the latter notion of obliviousness, or more formally, Definition 3.1 — this is directly implied by their proofs [ACN+20, RS21] even though not explicitly noted in their works.*

Useful notations. Throughout the paper, we will use G to denote the maximum number of gates in each intermediate layer, and we use $L-1$ to denote the total number of layers for gates. We use a tuple $(\ell, g) \in [L-1] \times [G]$ to refer to the g-th gate in layer ℓ. We use a tuple $(\ell, i) \in [L] \times [2n]$ to index the wires incoming into layer-ℓ gates, which are also the wires outgoing from layer-$(\ell-1)$ gates. The wires outgoing from the final layer-$(L-1)$ gates will be indexed as (L, i) where $i \in [2n]$. For convenience, we introduce the notation $\mathsf{Input}_{(\ell,g)}$ to denote the indices of wires incoming into the gate indexed (ℓ, g), and we use $\mathsf{Output}_{(\ell,g)}$ to denote the indices of wires outgoing from the gate (ℓ, g). In other words, the wires coming into gate (ℓ, g) are the set $\{(\ell, w)\}_{w \in \mathsf{Input}_{(\ell,g)}}$, and the wires outgoing from gate (ℓ, g) are the set $\{(\ell+1, w)\}_{w \in \mathsf{Output}_{(\ell,g)}}$.

As mentioned, the route of producer $u \in [n]$, denoted rte_u, includes which source vertex u is assigned to, and the set of wires it traverses to reach the destination vertex that corresponds to its consumer. We often denote $\mathsf{rte}_u := (j_1, \ldots, j_L)$ where j_1 denotes the source vertex (which can also be thought of as a wire) the producer u is assigned to, j_L denotes the destination vertex $2\pi(u)-1$ that corresponds to u's consumer, and for every $\ell \in [2, L-1]$, j_ℓ denotes a wire incoming into a layer-ℓ gate that the route traverses.

In our routing network, there are more wires in each layer than the number of producers or consumers. Therefore, some wires do not carry load. Henceforth in our paper, we also call such wires that do not carry actual load *filler* wires.

3.4 Constrained PRF

We need a constrained PRF where the input consists of a pair $t \in \{0,1\}^{\mathsf{tlen}}$ and $x \in \{0,1\}^{\mathsf{len}}$. We also call t the round and x the message. Each constraint can be specified by a tuple $(t^*, x^*, y) \in \{0,1\}^{\mathsf{tlen}} \times \{0,1\}^{\mathsf{len}} \times \{\{0,1\}^{\mathsf{len}} \cup \{2^{\mathsf{len}}\}\}$ — it means that the constrained key should be able to evaluate the PRF's outcome at any point (t, x) such that

$$t \neq t^* \text{ or } ((t = t^*) \text{ and } (x \in [y{:}] \cup \{x^*\}))$$

where $[y{:}]$ denotes the set of all messages of length len that are lexicographically greater than or equal to y. In other words, for some constrained round t^*, the constrained key only works for messages that are lexicographically greater than or equal to $y \in \{0,1\}^{\mathsf{len}}$ or equal to x^*.

A constrained pseudorandom function consists of the following algorithms:

- $\mathsf{sk} \leftarrow \mathbf{Gen}(1^\lambda, \mathsf{tlen}, \mathsf{len})$: takes in a security parameter 1^λ, the length tlen of the round, the length len of the input message, and outputs a secret key sk.

- $\sigma \leftarrow \mathbf{Eval}(\mathsf{sk}, t, x)$: a *deterministic* function that takes in a secret key sk, a round number $t \in \{0,1\}^{\mathsf{tlen}}$, a message $x \in \{0,1\}^{\mathsf{len}}$, and outputs the evaluation outcome σ.

- $\mathsf{sk}^* \leftarrow \mathbf{Constr}(\mathsf{sk}, t^*, [y:] \cup \{x^*\})$: takes in a secret key sk, a constrained round t^*, the set of messages $[y:] \cup \{x^*\}$ that can be evaluated for the constrained round t^*, and outputs a constrained key sk^*.

  For the special case when $y = 2^{\mathsf{len}}$, we sometimes also simply write $\mathsf{sk}^* \leftarrow \mathbf{Constr}(\mathsf{sk}, t^*, \{x^*\})$ or $\mathsf{sk}^* \leftarrow \mathbf{Constr}(\mathsf{sk}, t^*, x^*)$ to mean that only x^* can be evaluated for the constrained round t^*.

- $\sigma \leftarrow \mathbf{CEval}(\mathsf{sk}^*, t, x)$: a *deterministic* function that takes in a constrained key sk^*, a round $t \in \{0,1\}^{\mathsf{tlen}}$ and a message $x \in \{0,1\}^{\mathsf{len}}$, outputs the evaluation outcome σ.

Correctness. We say that a constrained PRF scheme satisfies correctness if the constrained key preserves functionality when evaluated at permitted points. Formally, we require that for any $\lambda, \mathsf{len}, t^* \in \{0,1\}^{\mathsf{tlen}}$, any $y \in \{0,1\}^{\mathsf{len}} \cup \{2^{\mathsf{len}}\}$, any $x^* \in \{0,1\}^{\mathsf{len}}$, any $(t, x) \in \mathbb{N} \times \{0,1\}^{\mathsf{len}}$ such that $t \neq t^*$ or $x \in [y:] \cup \{x^*\}$,

$$\Pr\left[\begin{array}{l}\mathsf{sk} \leftarrow \mathbf{Gen}(1^\lambda, \mathsf{tlen}, \mathsf{len}), \\ \mathsf{sk}^* \leftarrow \mathbf{Constr}(\mathsf{sk}, t^*, [y:] \cup \{x^*\})\end{array} : \mathbf{Eval}(\mathsf{sk}, t, x) = \mathbf{CEval}(\mathsf{sk}^*, t, x)\right] = 1$$

Security. We say that a constrained PRF scheme is secure, if given some constrained keys, the original PRF's evaluation outcomes at forbidden points (i.e., points that the constrained keys cannot evaluate) remain pseudorandom. Formally, consider the following experiment $\mathsf{ExptCPRF}^{\mathcal{A},b}(1^\lambda, \mathsf{tlen}, \mathsf{len})$ parametrized by a bit $b \in \{0,1\}$:

- The challenger calls $\mathsf{sk} \leftarrow \mathbf{Gen}(1^\lambda, \mathsf{tlen}, \mathsf{len})$.
- The adversary $\mathcal{A}$ can adaptively make the following queries:
  - **Eval**: $\mathcal{A}$ submits a pair (t, x), and the challenger returns to $\mathcal{A}$ the evaluation outcome $\mathbf{Eval}(\mathsf{sk}, t, x)$.
  - **Constr**: $\mathcal{A}$ submits a tuple (t^*, x^*, y), and the challenger computes a constrained key $\mathsf{sk}^* \leftarrow \mathbf{Constr}(\mathsf{sk}, t^*, [y:] \cup \{x^*\})$ and returns sk^* to $\mathcal{A}$.
  - **Challenge**: $\mathcal{A}$ submits a challenge pair $(\widetilde{t}, \widetilde{x})$. If $b = 0$, the challenger returns a random string of appropriate length to $\mathcal{A}$. Else, the challenger computes $\mathbf{Eval}(\mathsf{sk}, \widetilde{t}, \widetilde{x})$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$, the experiment outputs b'.

We say that $\mathcal{A}$ is *admissible* iff the following constraints hold with probability 1:

1. For any (t, x) submitted in an **Eval** query, the challenge $(\widetilde{t}, \widetilde{x}) \neq (t, x)$;
2. For any tuple (t^*, x^*, y) submitted during an **Constr** query, it must be that $\widetilde{t} = t^*$ and $\widetilde{x} \notin [y:] \cup \{x^*\}$.

Definition 3.3 (γ-subexponential security of constrained PRF)**.** We say that a constrained PRF scheme is γ-subexponentially secure, iff for any probabilistic polynomial-time (p.p.t.) *admissible* adversary $\mathcal{A}$, for any len that is polynomially bounded in λ, there exists λ_0 such that for any $\lambda > \lambda_0$, $|\Pr[\mathsf{ExptCPRF}^{\mathcal{A},0}(1^\lambda, \mathsf{tlen}, \mathsf{len}) = 1] - \Pr[\mathsf{ExptCPRF}^{\mathcal{A},1}(1^\lambda, \mathsf{tlen}, \mathsf{len}) = 1]| \leq 2^{-\lambda^\gamma}$.

3.5 Puncturable PRF

A puncturable PRF is a special case of constrained PRF defined as follows:

- $\mathsf{sk} \leftarrow \mathbf{Gen}(1^\lambda, \mathsf{len}) := \mathsf{CPRF}.\mathbf{Gen}(1^\lambda, \mathsf{len}, 0)$: takes in a security parameter 1^λ, the length len of the input message, and outputs a secret key sk
- $\sigma \leftarrow \mathbf{Eval}(\mathsf{sk}, x) := \mathsf{CPRF}.\mathbf{Eval}(\mathsf{sk}, x, \perp)$: a *deterministic* function that takes in a secret key sk, a message $x \in \{0,1\}^{\mathsf{len}}$, and outputs the evaluation σ.
- $\mathsf{sk}^* \leftarrow \mathbf{Puncture}(\mathsf{sk}, x^*) := \mathsf{CPRF}.\mathbf{Constr}(\mathsf{sk}, x^*, \perp)$: takes in a secret key sk, a point x^* to be punctured, and outputs a punctured key sk^*. This means that the punctured key can only be used to evaluate at $x \neq x^*$.
- $\sigma \leftarrow \mathbf{PEval}(\mathsf{sk}^*, x) := \mathsf{CPRF}.\mathbf{CEval}(\mathsf{sk}^*, x, \perp)$: a *deterministic* function that takes in a punctured key sk^*, and a message $x \in \{0,1\}^{\mathsf{len}}$, outputs the evaluation outcome σ.

3.6 Indistinguishability Obfuscation

The notion of indistinguishability obfuscation (iO) was first defined in [BGI+01a]. Recent works have shown constructions from well-founded assumptions [JLS21, GP20, WW20, BDGM20]. We give the formal definition below, taken almost verbatim from Jain et al. [JLS21].

Definition 3.4 (Indistinguishability Obfuscator (iO))**.** A uniform p.p.t. algorithm iO is called an indistinguishability obfuscator for polynomial-sized circuits if the following holds:

- **Completeness:** For every $\lambda \in \mathbb{N}$, every circuit C with input length n, every input $x \in \{0,1\}^n$, we have that
$$\Pr\left[C'(x) = C(x) : C' \leftarrow \mathsf{iO}(1^\lambda, C)\right] = 1.$$
- **γ-Subexponential Security:** For any two ensembles $\{C_{0,\lambda}\}_\lambda$, $\{C_{1,\lambda}\}_\lambda$ of polynomial-sized circuits that have the same size, input length, and output length, and are functionally equivalent (i.e., $C_{0,\lambda}(x) = C_{1,\lambda}(x)$ for every λ and x), for any p.p.t. adversary $\mathcal{A}$, there exists λ' such that for any $\lambda > \lambda'$,
$$\left|\Pr[\mathcal{A}(\mathsf{iO}(1^\lambda, C_{0,\lambda})) = 1] - \Pr[\mathcal{A}(\mathsf{iO}(1^\lambda, C_{1,\lambda})) = 1]\right| < 2^{-\lambda^\gamma}$$
.

3.7 Pseudorandom Generators

Definition 3.5 (γ-Subexponential Pseudorandom Generator)**.** We say that a function $G\colon \{0,1\}^* \to \{0,1\}^*$ is a *γ-subexponential pseudorandom generator* if the following holds:

- There exists a function ℓ such that $|G(x)| = \ell(|x|) > |x|$ for all $x \in \{0,1\}^*$, and
- For all PPT adversaries $\mathcal{A}$ there exists a λ' such that for all $\lambda > \lambda'$, it holds that
$$\left|\Pr\left[\mathcal{A}(1^\lambda, G(x)) = 1 \colon x \xleftarrow{\$} \{0,1\}^\lambda\right] - \Pr\left[\mathcal{A}(1^\lambda, r) = 1 \colon r \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}\right]\right| < 2^{-\lambda^\gamma}.$$

4 Proofs for Our SSU Signature Scheme

In Section 2.4, we defined the syntax and security of an SSU signature. We complete the definitions by giving the correctness definition below.

Correctness of SSU signature. An SSU signature is said to be correct iff the following holds,

- For all $\lambda, \mathsf{len}, \mathsf{tlen} \in \mathbb{N}$, $t \in \{0,1\}^{\mathsf{tlen}}, x \in \{0,1\}^{\mathsf{len}}$,

$$\Pr\left[\begin{array}{r}(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len}) \\ \sigma \leftarrow \mathbf{Sign}(\mathsf{sk}, t, x)\end{array} : \mathbf{Vf}(\mathsf{vk}, t, x, \sigma) = 1\right] = 1.$$

- For any $\lambda, \mathsf{len}, \mathsf{tlen} \in \mathbb{N}$, $t^*, t \in \{0,1\}^{\mathsf{tlen}}$, $x^*, x \in \{0,1\}^{\mathsf{len}}$ such that it is *not* the case that $t = t^*$ and $x \neq x^*$,

$$\Pr\left[\begin{array}{r}(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len}), \\ \mathsf{sk}^* \leftarrow \mathbf{Puncture}(\mathsf{sk}, t^*, x^*)\end{array} : \mathbf{Sign}(\mathsf{sk}, t, x) = \mathbf{PSign}(\mathsf{sk}^*, t, x)\right] = 1.$$

Parameter choices for our SSU signature scheme. Earlier in Section 2.4, we described an SSU signature scheme. To prove security, we additionally need to provide the parameters of the scheme, which we describe below.

We need the following building blocks:

- A constrained PRF scheme $\mathsf{PRF} = (\mathbf{Gen}, \mathbf{Constr}, \mathbf{Eval}, \mathbf{CEval})$, satisfying the syntax and security properties given in Section 3.4. Specifically, we require that it satisfies γ_{PRF}-subexponential security (Definition 3.3) for some parameter $\gamma_{\mathsf{PRF}} > 0$. We also assume that the outputs of **Eval** and **CEval** are of length λ, where λ is the security parameter given to **Gen**.

- An injective, length-doubling γ_G-subexponential pseudorandom generator (Definition 3.5), for some $\gamma_G > 0$.

- And indistinguishability obfuscation scheme iO which satisfies γ_{iO}-subexponential security (Definition 3.4) for some $\gamma_{\mathsf{iO}} > 0$.

Then, we define $\gamma = \min\{\gamma_{\mathsf{PRF}}, \gamma_G, \gamma_{\mathsf{iO}}\}$, and we instantiate PRF, G and iO with security parameter $\lambda^{2/\gamma}$.

4.1 Proof of Security

We now prove the security of our SSU signature scheme.

Lemma 4.1. *Suppose that* iO *satisfies completeness and that* G *is injective. Then, the above SSU signature scheme satisfies statistical unforgeability at* t^*.

Proof. If iO satisfies completeness, then the verification key vk, which is an obfuscation of the circuit $\widetilde{C}[\mathsf{sk}^*, t^*, x^*]$, behaves identical to $\widetilde{C}[\mathsf{sk}^*, t^*, x^*]$ itself. On inputs (t, x, σ) where $t = t^*$ and $x \neq x^*$, $\widetilde{C}[\mathsf{sk}^*, t^*, x^*]$ always rejects. If $t = t^*$ and $x = x^*$, $\widetilde{C}[\mathsf{sk}^*, t^*, x^*]$ only accepts if $G(\mathsf{PRF}.\mathbf{CEval}(\mathsf{sk}^*, t, x)) = G(\mathbf{PSign}(\mathsf{sk}^*, t, x)) = G(\sigma)$, which by the injectivity of G implies that $\sigma = \mathbf{PSign}(\mathsf{sk}^*, t, x)$. Thus, both conditions for statistical unforgeability given in Definition 2.2 are satisfied.

□

Lemma 4.2. *Suppose that* PRF *is a* γ_{PRF}*-subexponentially-secure constrained PRF,* iO *is a* γ_{iO}*-subexponentially secure indistinguishability obfuscator, and* G *is a* γ_G*-subexponential pseudorandom generator. Then, the above SSU signature scheme satisfies computational indistinguishability of simulated setup.*

Proof. Below, we abuse notation and refer to the y-th string in $\{0,1\}^{\mathsf{len}}$, ordered lexicographically, simply as y. In addition, we write $y \geq y'$ for two strings $y, y' \in \{0,1\}^{\mathsf{len}}$ if $y = y'$ or y comes after y' lexicographically. Fix t^* and x^*, and consider a PPT adversary $\mathcal{A}$. We show that $\mathcal{A}$ has negligible distinguishing advantage for the two distributions given in Definition 2.2. We do this via a sub-exponential number of hybrid experiments $\{\mathsf{Hyb}_y\}_{y \in \{0,1\}^{\mathsf{len}}}$, which we now define.

For $y \in \{0,1\}^{\mathsf{len}}$, let sk_y be a constrained key that can sign any message (t, x) where $t \neq t^*$ or ($t = t^*$ and ($x = x^*$ or $x \geq y$)), and define circuit $C_y[\mathsf{sk}_y, t^*, x^*]$ to be the following circuit:

<u>*Circuit* $C_y[\mathsf{sk}_y, t^*, x^*](t, x, \sigma)$</u>:

- if $t = t^*$ and $x \notin [y{:}] \cup \{x^*\}$: output 0;
- else:
  - if $G(\mathsf{PRF}.\mathbf{CEval}(\mathsf{sk}_y, t, x)) = G(\sigma)$, output 1, otherwise output 0.

Experiment Hyb_y**.** Hyb_y is defined as follows:

- let $\mathsf{sk} \leftarrow \mathsf{PRF}.\mathbf{Gen}(1^\lambda, \mathsf{tlen}, \mathsf{len})$;
- let $\mathsf{sk}^* \leftarrow \mathsf{PRF}.\mathbf{Constr}(\mathsf{sk}, t^*, \{x^*\})$;
- $\mathsf{sk}_y \leftarrow \mathsf{PRF}.\mathbf{Constr}(\mathsf{sk}, t^*, [y{:}] \cup \{x^*\})$;
- let $\mathsf{vk} \leftarrow \mathsf{iO}(1^{\lambda^{2/\gamma}}, C_y[\mathsf{sk}_y, t^*, x^*])$; and
- output $(\mathsf{sk}^*, \mathsf{vk})$.

Clearly, the circuit $C_1[\mathsf{sk}_1, t^*, x^*]$ is functionally equivalent to the circuit $C[\mathsf{sk}]$. The circuit $C_{2^{\mathsf{len}}+1}[\mathsf{sk}_{2^{\mathsf{len}}+1}, t^*, x^*]$ is functionally equivalent to the circuit $\widetilde{C}[\mathsf{sk}^*, t^*, x^*]$. Moreover, $C_{x^*}[\mathsf{sk}_{x^*}, t^*, x^*]$ and $C_{x^*+1}[\mathsf{sk}_{x^*+1}, t^*, x^*, x^*+1]$ are functionally equivalent. By the security of iO, we have that **Real** is computationally indistinguishable from Hyb_1, **Ideal** is computationally indistinguishable from $\mathsf{Hyb}_{2^{\mathsf{len}}+1}$, and Hyb_{x^*} and Hyb_{x^*+1} are computationally indistinguishable.

Below, we show that for any $y \neq x^*$, an adversary has advantage less than $O(2^{-\mathsf{len}^2})$ in distinguishing Hyb_y and Hyb_{y+1}. Since the number of hybrids is $2^{\mathsf{len}} + 1$, this proves the lemma. We do this via a sequence of subhybrids, as follows.

- $\mathsf{Hyb}_{y,1}$: Runs in the same way as Hyb_y, except we now compute $\mathsf{vk} \leftarrow \mathsf{iO}(\hat{C}_y[\mathsf{sk}_{y+1}, t^*, x^*, \alpha_y])$ to be an obfuscation of a different circuit $\hat{C}_y[\mathsf{sk}_{y+1}, t^*, x^*, \alpha_y]$, which we define below, and where $\mathsf{sk}_{y+1} \leftarrow \mathsf{PRF}.\mathbf{Constr}(\mathsf{sk}, t^*, [y+1{:}] \cup \{x^*\})$, and $\alpha_y \leftarrow G(\mathsf{PRF}.\mathbf{CEval}(\mathsf{sk}_y, t^*, y))$.
- $\mathsf{Hyb}_{y,2}$: Runs in the same way as $\mathsf{Hyb}_{y,1}$, except that we now compute $\alpha_y \leftarrow G(r)$, where $r \xleftarrow{\$} \{0,1\}^{\lambda^{2/\gamma}}$.
- $\mathsf{Hyb}_{y,3}$: Runs in the same way as $\mathsf{Hyb}_{y,2}$, except that we now compute $\alpha_y \xleftarrow{\$} \{0,1\}^{2\lambda^{2/\gamma}}$.

- $\mathsf{Hyb}_{y,4}$: Runs in the same way as $\mathsf{Hyb}_{y,3}$, except that we now compute $\alpha_y \xleftarrow{\$} \{0,1\}^{2\lambda^{2/\gamma}} \setminus \left\{G(r) \colon r \in \{0,1\}^{2\lambda^{2/\gamma}}\right\}$.

<u>*Circuit* $\hat{C}_y[\mathsf{sk}_{y+1}, t^*, x^*, \alpha_y](t, x, \sigma)$:</u>

- If $t = t^*$ and $x \notin [y{:}] \cup \{x^*\}$: output 0;
- Else if $t = t^*$ and $x = y$:
  - If $G(\sigma) = \alpha_y$, output 1, otherwise output 0.
- Else:
  - if $G(\mathsf{PRF}.\mathbf{CEval}(\mathsf{sk}_{y+1}, t, x)) = G(\sigma)$, output 1, otherwise output 0.

The following claims finish the proof. □

Claim 4.3. *Assume that* iO *is* γ_{iO}*-subexponentially secure. Then* $\mathcal{A}$ *has advantage at most* $2^{-\mathsf{len}^2}$ *in distinguishing* Hyb_y *from* $\mathsf{Hyb}_{y,1}$.

Proof. The only difference between the two hybrids is the circuit which is obfuscated as vk. Fix the randomness of the experiment except for the randomness used in iO to generate vk. There must be a way to fix this randomness which preserves $\mathcal{A}$'s distinguishing advantage. In this way, we fix the two circuits C_y and $\hat{C}_y$ whose obfuscations are given to $\mathcal{A}$. If $\mathcal{A}$ distinguishes between these two experiments with probability greater than $2^{-\mathsf{len}^2}$, then a straightforward reduction shows a distinguishing adversary against the security of iO with the same advantage. Since it holds that

$$2^{-\mathsf{len}^2} \geq 2^{-\lambda^2} = 2^{-\lambda^{(2/\gamma)\cdot\gamma}} \geq 2^{-\lambda^{(2/\gamma)\cdot\gamma_{\mathsf{iO}}}},$$

and since iO is used with security parameter $\lambda^{2/\gamma}$, assuming C_y and $\hat{C}_y$ are functionally equivalent, this contradicts γ_{iO}-security of the scheme iO.

We now argue functional equivalence of C_y and $\hat{C}_y$. It is clear that $C_y(t, x, \sigma) = \hat{C}_y(t, x, \sigma)$ on all inputs except for those of the form $(t^*, y, \_)$. For such inputs, equivalence follows because $\alpha_y = G(\mathsf{PRF}.\mathbf{CEval}(\mathsf{sk}_y, t^*, y))$, and thus the equality check is the same across both circuits. □

Claim 4.4. *Assume that* PRF *is* γ_{PRF}*-subexponentially secure. Then* $\mathcal{A}$ *has advantage at most* $2^{-\mathsf{len}^2}$ *in distinguishing* $\mathsf{Hyb}_{y,1}$ *from* $\mathsf{Hyb}_{y,2}$.

Proof. The only difference between the two hybrids is the way that the value α_y which is embedded in $\hat{C}_y$ is generated. Assume $\mathcal{A}$ distinguishes between these two experiments with probability greater than $2^{-\mathsf{len}^2}$. We build a reduction $\mathcal{A}'$ to the security of PRF. $\mathcal{A}'$ interacts with the constrained PRF challenger, and at the same time runs $\mathsf{Hyb}_{y,2}$ with $\mathcal{A}$, except for the following differences:

- instead of computing sk^* directly, $\mathcal{A}'$ submits a **Constr** query $(t^*, x^*, 2^{\mathsf{len}}+1)$ to the challenger, and uses the received value as sk^*.
- instead of computing sk_{y+1} directly, $\mathcal{A}'$ submits a **Constr** query $(t^*, x^*, y+1)$ to the challenger, and uses the received value as sk_{y+1}.

- Before computing vk, $\mathcal{A}'$ submits the **Challenge** query (t^*, y). It then receives a value σ_y from the challenger, uses the hardcoded value $\alpha_y = G(\sigma_y)$ when computing the obfuscation of $\hat{C}_y[\mathsf{sk}_{y+1}, t^*, x^*, \alpha_y]$.

Note that $\mathcal{A}'$ is an admissible adversary, since it does not make any **Eval** queries to the challenger, and the **Challenge** query is at position (t^*, y), where neither sk^* or sk_{y+1} can evaluate at that point. Observe that if the challenger outputs $\sigma_y = \textbf{Eval}(\mathsf{sk}, t^*, y)$, then the view of $\mathcal{A}$ is identical to that in $\mathsf{Hyb}_{y,1}$. If the challenger outputs $\sigma_y \xleftarrow{\$} \{0,1\}^{\lambda^{2/\gamma}}$, then the view of $\mathcal{A}$ is identical to that in $\mathsf{Hyb}_{y,2}$. Thus, we have that $\mathcal{A}'$ distinguishes these two scenarios with probability greater than $2^{-\mathsf{len}^2}$. Since it holds that

$$2^{-\mathsf{len}^2} \geq 2^{-\lambda^2} = 2^{-\lambda^{(2/\gamma)\cdot\gamma}} \geq 2^{-\lambda^{(2/\gamma)\cdot\gamma_{\mathsf{PRF}}}},$$

and PRF is initialized with security parameter $\lambda^{2/\gamma}$, $\mathcal{A}'$ contradicts γ_{PRF}-subexponential security of PRF. ☐

Claim 4.5. *Assume that G is a γ_G-subexponential pseudorandom generator. Then $\mathcal{A}$ has advantage at most $2^{-\mathsf{len}^2}$ in distinguishing $\mathsf{Hyb}_{y,2}$ from $\mathsf{Hyb}_{y,3}$.*

Proof. The only difference between the two hybrids is the way that the value α_y which is embedded in $\hat{C}_y$ is generated. In $\mathsf{Hyb}_{y,2}$, α_y is the output of G on a random input in $\{0,1\}^{\lambda^{2/\gamma}}$, and in $\mathsf{Hyb}_{y,3}$, α_y is a uniform random value in $\{0,1\}^{2\lambda^{2/\gamma}}$. If $\mathcal{A}$ distinguishes between these two experiments with probability greater than $2^{-\mathsf{len}^2}$, then a straightforward reduction shows a distinguishing adversary against the security of G with the same advantage. Since it holds that

$$2^{-\mathsf{len}^2} \geq 2^{-\lambda^2} = 2^{-\lambda^{(2/\gamma)\cdot\gamma}} \geq 2^{-\lambda^{(2/\gamma)\cdot\gamma_G}},$$

and G is evaluated on inputs of size $\lambda^{2/\gamma}$, this contradicts γ_G-subexponential security of G.

☐

Claim 4.6. *$\mathcal{A}$ has advantage at most $2^{-\mathsf{len}^2}$ in distinguishing $\mathsf{Hyb}_{y,3}$ from $\mathsf{Hyb}_{y,4}$.*

Proof. The only difference between the two hybrids is the way that the value α_y which is embedded in $\hat{C}_y$ is generated. In $\mathsf{Hyb}_{y,3}$, α_y is a uniform random value in $\{0,1\}^{2\lambda^{2/\gamma}}$, and in $\mathsf{Hyb}_{y,3}$, α_y is a uniform random value in $\{0,1\}^{2\lambda^{2/\gamma}} \setminus \left\{G(r) \colon r \in \{0,1\}^{2\lambda^{2/\gamma}}\right\}$. Thus $\mathcal{A}$'s distinguishing advantage is (unconditionally) upper bounded by the probability that α_y ends up in $\left\{G(r) \colon r \in \{0,1\}^{2\lambda^{2/\gamma}}\right\}$ in $\mathsf{Hyb}_{y,3}$. We have that

$$\Pr\left[\alpha_y \in \left\{G(r) \colon r \in \{0,1\}^{2\lambda^{2/\gamma}}\right\} \text{ in } \mathsf{Hyb}_{y,3}\right] = \frac{2^{\lambda^{2/\gamma}}}{2^{2\lambda^{2/\gamma}}} = 2^{-\lambda^{2/\gamma}} < 2^{-\mathsf{len}^2},$$

which proves the claim.

☐

Claim 4.7. *Assume that iO is γ_{iO}-subexponentially secure. Then $\mathcal{A}$ has advantage at most $2^{-\mathsf{len}^2}$ in distinguishing $\mathsf{Hyb}_{y,4}$ from Hyb_{y+1}.*

Proof. The only difference between the two hybrids is the circuit which is obfuscated as vk. Fix the randomness of the experiment except for the randomness used in iO to generate vk. There must be a way to fix this randomness which preserves $\mathcal{A}$'s distinguishing advantage. In this way, we fix the two circuits $\hat{C}_y$ and C_{y+1} whose obfuscations are given to $\mathcal{A}$. If $\mathcal{A}$ distinguishes between these two experiments with probability greater than $2^{-\mathsf{len}^2}$, then a straightforward reduction shows a distinguishing adversary against the security of iO with the same advantage. Since it holds that

$$2^{-\mathsf{len}^2} \geq 2^{-\lambda^2} = 2^{-\lambda^{(2/\gamma)\cdot\gamma}} \geq 2^{-\lambda^{(2/\gamma)\cdot\gamma_{\mathsf{iO}}}},$$

and since iO is used with security parameter $\lambda^{2/\gamma}$, assuming $\hat{C}_y$ and C_{y+1} are functionally equivalent, this contradicts γ_{iO}-security of the scheme iO.

We now argue functional equivalence of $\hat{C}_y$ and C_{y+1}. It is clear that $\hat{C}_y(t, x, \sigma) = C_{y+1}(t, x, \sigma)$ on all inputs except for those of the form $(t^*, y, \_)$. For such inputs, since α_y is outside the image of G, $\hat{C}_y$ always outputs 0, which is the same behavior as that of C_{y+1}, since $y \neq x^*$.

□

5 Proofs for Our NIAR Scheme

In this section, we prove that the NIAR construction presented in Section 2.5 is selectively secure as defined in Definition 2.3. In the following sections, we say that a wire is "corrupt" or "honest" if it is on a path which originates with a corrupted or *honest* sender respectively. We say all other wires are "filler" wires. Also, we sometimes use the shorthand "wire (ℓ, j)" to refer to j^{th} wire in the ℓ^{th} layer.

To prove selective security, we first need to define the **Setup*** algorithm, as shown in Figure 2.

Claim 5.1. *If the routing network satsifies obliviousness as defined in Definition 3.1, then, for either $b \in \{0, 1\}$, the terms $\left(\{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{ek}_u^{(b)}\}_{u\in\mathcal{H}_S}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}\right)$ output by* **Setup**$^*(1^\lambda, \mathsf{tlen}, \mathsf{len}, n, t^*, \pi^{(0)}, \pi^{(1)}, \mathcal{K}_S, \{x_{u,t^*}^{(0)}, x_{u,t^*}^{(1)}\}_{u\in\mathcal{H}_S})$ *as described in Figure 2 have the same joint distribution as the output of the real* **Setup**$(1^\lambda, \mathsf{tlen}, \mathsf{len}, n, \pi^{(b)})$ *as described in Section 2.5.*

Setup$^*(1^\lambda, \mathsf{tlen}, \mathsf{len}, n, t^*, \pi^{(0)}, \pi^{(1)}, \mathcal{K}_S, \{x^{(0)}_{u,t^*}, x^{(1)}_{u,t^*}\}_{u\in\mathcal{H}_S})$: on inputs the security paramter 1^λ, the length of the time step tlen, the individual message length len, the number of parties n, the challenge round t^*, two permutations $\pi^{(0)}$, $\pi^{(1)}$, a set of corrupt senders $\mathcal{K}_S \subset [n]$, and the challenge messages $\{x^{(0)}_{u,t^*}, x^{(1)}_{u,t^*}\}_{u\in\mathcal{H}_S}$, does the following where $\mathcal{H}_S = [n] \setminus \mathcal{K}_S$:

1. **Sampling Wire Keys**: For each wire (ℓ, i) in $[L] \times [2n]$:
   (a) Sample PRF key $k_{(\ell,i)} \leftarrow \mathsf{PRF}.\mathbf{Gen}(1^\lambda)$ as the encryption key for this wire.
   (b) Sample two pairs of signing and verification keys, one for signing routes and other for signing messages,
$$(\mathsf{rsk}_{(\ell,i)}, \mathsf{rvk}_{(\ell,i)}) \leftarrow \mathsf{Sig}.\mathbf{Setup}(1^\lambda, \mathsf{tlen}, L \cdot \log(2n)) \ ,$$
$$(\mathsf{msk}_{(\ell,i)}, \mathsf{mvk}_{(\ell,i)}) \leftarrow \mathsf{Sig}.\mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{tlen} + \mathsf{len} + n \cdot 2L \cdot \log(2n)) \ .$$
2. **Sampling Routes**: Run the $\mathsf{AssignRoutes}^*$ procedure (Section 3.3) on inputs $(1^\lambda, n, \pi^{(0)}, \pi^{(1)}, \mathcal{K}_S)$ to compute the output out. Abort if $\mathsf{out} = \perp$. Else parse out as $(\{\mathsf{rte}'_u\}_{u\in\mathcal{K}_S}, \{\mathsf{rte}'^{(0)}_u, \mathsf{rte}'^{(1)}_u\}_{u\in\mathcal{H}_S})$. For each sender $u \in \mathcal{K}_S$ do the following:
   (a) Parse $\mathsf{rte}'_u = (j_1, \dots, j_L)$.
   (b) Sign rte'_u using route signing keys for each wire along rte'_u. That is, for $\ell \in [L]$ compute $\mathsf{rsig}_\ell = \mathsf{Sig}.\mathbf{Sign}(\mathsf{rsk}_{(\ell,j_\ell)}, 1, \mathsf{rte}'_u)$.
   (c) Set $\mathsf{rte}_u = (\mathsf{rte}'_u, \mathsf{rsig}_1, \dots, \mathsf{rsig}_L)$.

   For each sender $u \in \mathcal{H}_S$ and for each $\beta \in \{0, 1\}$, do the following:
   (a) Parse $\mathsf{rte}'^{(\beta)}_u = (j_1, \dots, j_L)$.
   (b) Sign $\mathsf{rte}'^{(\beta)}_u$ using route signing keys for each wire along $\mathsf{rte}'^{(\beta)}_u$. That is, for $\ell \in [L]$ compute $\mathsf{rsig}^{(\beta)}_\ell = \mathsf{Sig}.\mathbf{Sign}(\mathsf{rsk}_{(\ell,j_\ell)}, 1, \mathsf{rte}'^{(\beta)}_u)$.
   (c) Set $\mathsf{rte}^{(\beta)}_u = (\mathsf{rte}'^{(\beta)}_u, \mathsf{rsig}^{(\beta)}_1, \dots, \mathsf{rsig}^{(\beta)}_L)$.
3. **Setting Routing Token**:
   (a) For each merge-split gate (ℓ, g) in $[L-1] \times [G]$, compute an indistinguishability obfuscation $\overline{\mathsf{Gate}}_{(\ell,g)} \leftarrow \mathsf{iO}(1^\lambda, \mathsf{Gate}_{(\ell,g)})$ of the circuit $\mathsf{Gate}_{(\ell,g)}$ described in Figure 1.
   (b) Set $\mathsf{tk} = \{\overline{\mathsf{Gate}}_{(\ell,g)} : \ell \in [L-1], g \in [G]\}$.
4. **Setting Sender Keys**: For each $u \in \mathcal{K}_S$, set $\mathsf{ek}_u = (i_u, k_{(1,i_u)}, \mathsf{msk}_{(1,i_u)}, \mathsf{rte}_u)$. For each $u \in \mathcal{H}_S$ and $\beta \in \{0, 1\}$, set $\mathsf{ek}^{(\beta)}_u = (i_u, k_{(1,i_u)}, \mathsf{msk}_{(1,i_u)}, \mathsf{rte}^{(\beta)}_u)$.
5. **Setting Receiver Keys**: For each $v \in [n]$, set $\mathsf{rk}_v = k_{(L,2v-1)}$.
6. Output $(\{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{ek}^{(0)}_u\}_{u\in\mathcal{H}_S}, \{\mathsf{ek}^{(1)}_u\}_{u\in\mathcal{H}_S}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk})$.

Figure 2: The **Setup*** algorithm for the routing scheme.

Proof. One can create a simple reduction such that if for any $b \in \{0,1\}$ the aforementioned joint distributions of **Setup** and **Setup*** could be distinguished with some noticeable probability, then, the reduction can distinguish the following two distributions with noticeable probability and thus break the obliviousness of the routing network.

- $\{(\{\mathsf{rte}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{rte}_u\}_{u\in\mathcal{H}_S}) : (\mathsf{rte}_1,\ldots,\mathsf{rte}_n) \leftarrow \mathsf{AssignRoutes}(1^\lambda, n, \pi^{(b)})\}$
- $\left\{\left(\{\mathsf{rte}_u\}_{u\in\mathcal{K}_S}, \left\{\mathsf{rte}_u^{(b)}\right\}_{u\in\mathcal{H}_S}\right) : \left(\{\mathsf{rte}_u\}_{u\in\mathcal{K}_S}, \left\{\mathsf{rte}_u^{(0)}, \mathsf{rte}_u^{(1)}\right\}_{u\in\mathcal{H}_S}\right)\right.$
$\left.\leftarrow \mathsf{AssignRoutes}^*(1^\lambda, n, \pi^{(0)}, \pi^{(1)}, \mathcal{K}_S)\right\}$

□

Theorem 5.2. *Suppose that* PRF *is a polynomially secure puncturable PRF,* Sig *is a polynomially secure deterministic SSU signature scheme, and* iO *is a polynomially secure indistinguishability obfuscation scheme. Then, our NIAR construction in Section 2.5 satisfies selective security as defined in Section 2.1.*

In our paper, we instantiate the polynoially secure deterministic SSU signature scheme assuming the existence of a subexponentially secure indistinguishability obfuscation scheme and one-way permutations (Theorem 1.2). Consequently, we obtain the following corollary.

Corollary 5.3 (Restatement of Theorem 1.1.)**.** *Let λ be security parameter. Let $n = n(\lambda)$ be the number of senders/receivers. Then, assuming the existence of subexponentially-secure indistinguishability obfuscator and one-way permutations, there exists a NIAR scheme that satisfies selectively receiver-insider security. Further, the asymptotical performance bounds are as follows:*

1. *the token size and router computation per time step is $\widetilde{O}_\lambda(n)$ where $\widetilde{O}_\lambda(\cdot)$ hides $\mathsf{poly}(\lambda, \log n)$ factors for some fixed $\mathsf{poly}(\cdot)$;*
2. *the per-sender communication and encryption time per bit of the message is $\widetilde{O}_\lambda(1)$;*
3. *each sender key is of length $\widetilde{O}_\lambda(1)$, each receiver key is of length $O_\lambda(1)$.*

5.1 The Hybrids

We prove computational indistinguishability of hybrid experiments $\mathsf{SelSingleCh}^{0,\mathcal{A}}(1^\lambda, \mathsf{tlen}, \mathsf{len})$ and $\mathsf{SelSingleCh}^{1,\mathcal{A}}(1^\lambda, \mathsf{tlen}, \mathsf{len})$ via a sequence of hybrids, which we describe below. In each hybrid, the challenger plays a game with a PPT adversary $\mathcal{A}$. The first hybrid $\mathsf{Hyb}_0^{(0)}$ is identical to $\mathsf{SelSingleCh}^{0,\mathcal{A}}(1^\lambda, \mathsf{tlen}, \mathsf{len})$, and the last hybrid $\mathsf{Hyb}_0^{(1)}$ is identical to $\mathsf{SelSingleCh}^{1,\mathcal{A}}(1^\lambda, \mathsf{tlen}, \mathsf{len})$. Proving indistinguishability of the worlds thus reduces to proving indistinguishability between each adjacent pair of hybrids.

The hybrid sequence we follow is $\mathsf{Hyb}_0^{(0)} \to \ldots \to \mathsf{Hyb}_{13}^{(0)} \to \mathsf{Hyb}_{13}^{(1)} \to \ldots \to \mathsf{Hyb}_0^{(1)}$ and their descriptions are as follows for $b \in \{0,1\}$. In the middle, there is a sequence of $8(L-1)$ hybrids $\{\mathsf{Hyb}_{6,\ell,i}^{(0)}\}_{\ell\in[L-1], i\in[8]}$, with nine hybrids per layer, which we refer to as the "inner hybrids."

Hybrid $\mathsf{Hyb}_0^{(b)}$: In this hybrid, the challenger plays the game $\mathsf{SelSingleCh}^{b,\mathcal{A}}(1^\lambda, \mathsf{tlen}, \mathsf{len})$ with $\mathcal{A}$.

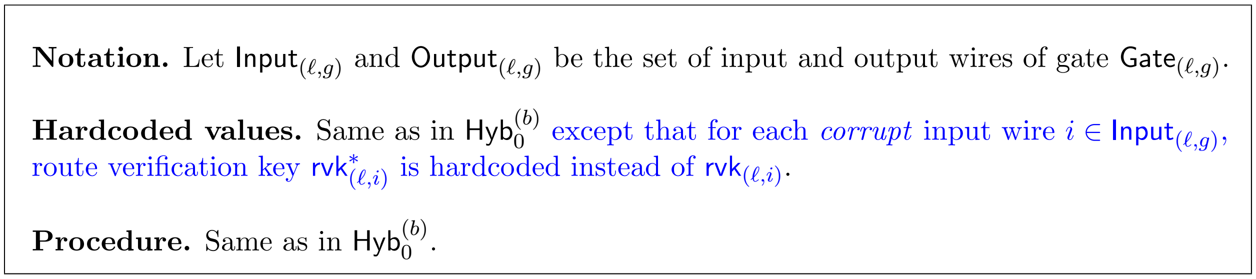
Hybrid $\mathsf{Hyb}_1^{(b)}$: This hybrid is identical to $\mathsf{Hyb}_0^{(b)}$ except that for all the *corrupt* wires, the challenger punctures the route signing keys at the corresponding routes and uses these punctured keys to generate the route signatures. More specifically, for each wire $(\ell, i = j_\ell)$, where $j_\ell \in \overline{\mathsf{rte}}_u^*$ for some sender $u \in \mathcal{K}_S$, we compute the route signatures as follows:

$$\begin{aligned}(\mathsf{rsk}_{(\ell,i)}, \mathsf{rvk}_{(\ell,i)}) &\leftarrow \mathsf{Sig}.\mathbf{Setup}(1^\lambda, \mathsf{tlen}, L \cdot \log(2n)) \ , \\ \mathsf{rsk}'_{(\ell,i)} &\leftarrow \mathsf{Sig}.\mathbf{Puncture}(\mathsf{rsk}_{(\ell,i)}, 1, \overline{\mathsf{rte}}_u^*) \ , \\ \mathsf{rsig}_\ell^* &= \mathsf{Sig}.\mathbf{PSign}(\mathsf{rsk}'_{(\ell,i)}, 1, \overline{\mathsf{rte}}_u^*) \ .\end{aligned}$$

Hybrid $\mathsf{Hyb}_2^{(b)}$: This hybrid is identical to $\mathsf{Hyb}_1^{(b)}$ except that the challenger uses the simulated setup to generate route signing/verification keys for all the *corrupt* wires. More specifically, for each wire $(\ell, i = j_\ell)$, where $j_\ell \in \overline{\mathsf{rte}}_u^*$ for some sender $u \in \mathcal{K}_S$, we compute the route signatures as follows:

$$\begin{aligned}(\mathsf{rsk}^*_{(\ell,i)}, \mathsf{rvk}^*_{(\ell,i)}) &\leftarrow \mathsf{Sig}.\mathbf{SimSetup}(1^\lambda, \mathsf{tlen}, L \cdot \log(2n), 1, \overline{\mathsf{rte}}_u^*) \ , \\ \mathsf{rsig}_\ell^* &= \mathsf{Sig}.\mathbf{PSign}(\mathsf{rsk}^*_{(\ell,i)}, 1, \overline{\mathsf{rte}}_u^*).\end{aligned}$$

Then, we replace the gates $\mathsf{Gate}_{(\ell,g)}$ for all $\ell \in [L-1], g \in [G]$ as described in Figure 3.



Notation. Let $\mathsf{Input}_{(\ell,g)}$ and $\mathsf{Output}_{(\ell,g)}$ be the set of input and output wires of gate $\mathsf{Gate}_{(\ell,g)}$.

Hardcoded values. Same as in $\mathsf{Hyb}_0^{(b)}$ except that for each *corrupt* input wire $i \in \mathsf{Input}_{(\ell,g)}$, route verification key $\mathsf{rvk}^*_{(\ell,i)}$ is hardcoded instead of $\mathsf{rvk}_{(\ell,i)}$.

Procedure. Same as in $\mathsf{Hyb}_0^{(b)}$.

Figure 3: The circuit $\mathsf{Gate}_{(\ell,g)}$ in hybrid experiment $\mathsf{Hyb}_2^{(b)}$.

Notation. Let $\mathsf{Input}_{(\ell,g)}$ and $\mathsf{Output}_{(\ell,g)}$ be the set of input and output wires of gate $\mathsf{Gate}_{(\ell,g)}$.

Hardcoded values. Same as in $\mathsf{Hyb}_2^{(b)}$. Additionally, for each *corrupt* input wire $i \in \mathsf{Input}_{(\ell,g)}$, suppose it is on expected route $\mathsf{rte}_u^* = (\overline{\mathsf{rte}}_u^* = (j_1^*, \ldots, j_L^*), (\mathsf{rsig}_1^*, \ldots, \mathsf{rsig}_L^*))$ for some sender $u \in [n]$. Then, hardcode (i, rte_u^*). **Procedure.**

- Step 1: For each input wire $i \in \mathsf{Input}_{(\ell,g)}$, if it is a *filler/honest* wire, then, compute as in $\mathsf{Hyb}_2^{(b)}$. Else:
  - Step (a) is same as in $\mathsf{Hyb}_2^{(b)}$.
  - Step (b): **Decrypt and authenticate the message/route:**
    - Steps i, ii, iii are same as in $\mathsf{Hyb}_2^{(b)}$.
    - Step iv: Parse rte as $((j_1, \ldots, j_L), (\mathsf{rsig}_1, \ldots, \mathsf{rsig}_L))$ and perform the following checks to authenticate the route rte: If $\mathsf{rte} \neq \overline{\mathsf{rte}}_u^*$, abort. If $\mathsf{rsig}_\ell \neq \mathsf{rsig}_\ell^*$, abort.
  - Step (c): **Prepare the output ciphertext $\mathsf{CT}_{(\ell+1,j_{\ell+1})}$:**
    - Step i: Let $j = j_{\ell+1}^*$. If $\mathsf{CT}_{(\ell+1,j)}$ has already been computed, then abort.
    - Step ii: If $\ell < L-1$, compute $\mathsf{msig}' = \mathsf{Sig}.\mathbf{Sign}(\mathsf{msk}_{(\ell+1,j)}, t, (x, \mathsf{rte}_u^*))$ and $\mathsf{CT}_{(\ell+1,j)} = (x, \mathsf{rte}_u^*, \mathsf{msig}') \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,j)}, t)$.
    - Step iii is same as in $\mathsf{Hyb}_2^{(b)}$.
- Steps 2 and 3 are same as in $\mathsf{Hyb}_2^{(b)}$.

Figure 4: The circuit $\mathsf{Gate}_{(\ell,g)}$ in hybrid experiment $\mathsf{Hyb}_3^{(b)}$.

Hybrid $\mathsf{Hyb}_3^{(b)}$: In this hybrid, for each *corrupt* wire, we perform the route authentication checks by comparing with hardcoded routes and corresponding signatures. Specifically, in each gate we hardcode the relevant routes sampled by Setup along with the route signature as computed in hybrid $\mathsf{Hyb}_2^{(b)}$. Then, we replace the gates $\mathsf{Gate}_{(\ell,g)}$ for all $\ell \in [L-1], g \in [G]$ as described in Figure 4.

Hybrid $\mathsf{Hyb}_4^{(b)}$: This hybrid is identical to $\mathsf{Hyb}_3^{(b)}$, except that the challenger punctures the message signing keys for all *filler/honest* wires $(1, i)$ in the first layer at the challenge round t^* and challenge plaintext $\tilde{x}^* = (x_{u,t^*}^{(b)}, \mathsf{rte}_u^{(b)})$ (or $\tilde{x}^* = (\bot_{\mathsf{filler}}, \bot_{\mathsf{filler}})$ in case of *filler*). That is, for each such wire $(1, i)$,

$$(\mathsf{msk}_{(1,i)}, \mathsf{mvk}_{(1,i)}) \leftarrow \mathsf{Sig}.\mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{tlen} + \mathsf{len} + n \cdot 2L \cdot \log(2n)),$$
$$\mathsf{msk}'_{(1,i)} \leftarrow \mathsf{Sig}.\mathbf{Puncture}(\mathsf{msk}_{(1,i)}, t^*, \tilde{x}^*).$$

Then, for each such wire $(1, i)$ in the first layer, whenever the challenger has to compute message signature for any round, it computes the message signatures for the first layer as follows:

$$\mathsf{msig}_{(1,i)} = \mathsf{Sig}.\mathbf{PSign}(\mathsf{msk}'_{(1,i)}, \cdot, \cdot).$$

Hybrid $\mathsf{Hyb}_5^{(b)}$: This hybrid is identical to $\mathsf{Hyb}_4^{(b)}$, except that the challenger uses the simulated setup to puncture message signing/verification keys for all *filler/honest* wires $(1, i)$ in the first layer at the challenge round t^* and challenge plaintext $\tilde{x}^* = (x^{(b)}_{u,t^*}, \mathsf{rte}^{(b)}_u)$ (or $\tilde{x}^* = (\perp_{\mathsf{filler}}, \perp_{\mathsf{filler}})$ in case of *filler*). That is, for each such wire $(1, i)$,

$$(\mathsf{msk}^*_{(1,i)}, \mathsf{mvk}^*_{(1,i)}) \leftarrow \mathsf{Sig}.\mathbf{SimSetup}(1^\lambda, \mathsf{tlen}, \mathsf{tlen} + \mathsf{len} + n \cdot 2L \cdot \log(2n), t^*, \tilde{x}^*).$$

Then, the challenger replaces the gates $\mathsf{Gate}_{(1,g)}$ for all $g \in [G]$ as described in Figure 5.

Notation. Let $\mathsf{Input}_{(1,g)}$ and $\mathsf{Output}_{(1,g)}$ be the set of input and output wires of gate $\mathsf{Gate}_{(1,g)}$.

Hardcoded values. Same as in $\mathsf{Hyb}^{(b)}_3$ except that for each *filler/honest* wire $i \in \mathsf{Input}_{(1,g)}$, the message verification key $\mathsf{mvk}^*_{(1,i)}$ is hardcoded instead of $\mathsf{mvk}_{(1,i)}$.

Procedure. Same as in $\mathsf{Hyb}^{(b)}_3$.

Figure 5: The circuit $\mathsf{Gate}_{(\ell,g)}$ in hybrid experiment $\mathsf{Hyb}^{(b)}_5$.

Hybrid $\mathsf{Hyb}^{(b)}_{6,\ell,1}$ for each $\ell \in [L-1]$: This hybrid is identical to $\mathsf{Hyb}^{(b)}_{6,\ell-1,5}$ (or $\mathsf{Hyb}^{(b)}_5$ if $\ell = 1$), except that for all the gates in this layer $\mathsf{Gate}_{(\ell,g)}$ for all $g \in [G]$, for all *filler/honest* wires (ℓ, i) in the input layer ℓ, the challenger hardcodes the expected challenge message $x^*_{(\ell,i)} = x^{(b)}_{u,t^*}$ (or $x^*_{(\ell,1)} = \perp_{\mathsf{filler}}$ in case of *filler* wire), the expected route $\mathsf{rte}^* = \mathsf{rte}^*_u$ (or $\mathsf{rte}^* = \perp_{\mathsf{filler}}$ in case of *filler* wire) and the message signature $\mathsf{msig}^*_{(\ell,i)} = \mathsf{Sig}.\mathbf{PSign}(\mathsf{msk}^*_{(\ell,i)}, t^*, (x^*, \mathsf{rte}^*))$ for the challenge round t^* and compares the message, route and message signature in the decrypted plaintext agaist the respective hardcoded challenge message, expected route and message signature instead of checking via $\mathsf{Sig}.\mathbf{Vf}$. Subsequently, for the outgoing wires, it uses $x^*_{(\ell,i)}$ and rte^* for computing the outgoing ciphertexts in layer $\ell + 1$. More formally, for all $g \in [G]$, the gates $\mathsf{Gate}_{(\ell,g)}$ are changed as described in Figure 6.

Notation. Let $\mathsf{Input}_{(\ell,g)}$ and $\mathsf{Output}_{(\ell,g)}$ be the set of input and output wires of gate $\mathsf{Gate}_{(\ell,g)}$.

Hardcoded values. Same as in $\mathsf{Hyb}^{(b)}_{6,\ell-1,5}$ (or $\mathsf{Hyb}^{(b)}_{5}$ if $\ell = 1$). Additionally, round t^* is hardcoded and for each *filler/honest* input wire $i \in \mathsf{Input}_{(\ell,g)}$, the expected challenge message $x^*_{(\ell,i)}$, expected route rte^* and signature $\mathsf{msig}^*_{(\ell,i)}$ are hardcoded.

Procedure. $\mathsf{Gate}_{(\ell,g)}$ takes as input a round t and a set of ciphertexts $\{\mathsf{CT}_{(\ell,i)} : i \in \mathsf{Input}_{(\ell,g)}\}$ corresponding to the input wires. Depending on the layer ℓ, it computes as follows.

- Step 1: For each input wire $i \in \mathsf{Input}_{(\ell,g)}$, if $t \neq t^*$ or wire i is *corrupt*, then, compute as in $\mathsf{Hyb}^{(b)}_{6,\ell-1,5}$ (or $\mathsf{Hyb}^{(b)}_{5}$ if $\ell = 1$). Else:
  - Step (a) is same as in $\mathsf{Hyb}^{(b)}_{6,\ell-1,5}$ (or $\mathsf{Hyb}^{(b)}_{5}$ if $\ell = 1$).
  - Step (b): **Decrypt and authenticate the message/route:**
    - Step i: compute the plaintext $(x, \mathsf{rte}, \mathsf{msig}) = \mathsf{CT}_{(\ell,i)} \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell,i)}, t^*)$.
    - Step ii: If $x \neq x^*_{(\ell,i)}$ or $\mathsf{rte} \neq \mathsf{rte}^*$ or $\mathsf{msig} \neq \mathsf{msig}^*_{(\ell,i)}$, then abort.
    - Steps iii and iv are same as in $\mathsf{Hyb}^{(b)}_{6,\ell-1,5}$ (or $\mathsf{Hyb}^{(b)}_{5}$ if $\ell = 1$).
  - Step (c): **Prepare the output ciphertext $\mathsf{CT}_{(\ell+1,j_{\ell+1})}$:**
    - Step i is same as in $\mathsf{Hyb}^{(b)}_{6,\ell-1,5}$ (or $\mathsf{Hyb}^{(b)}_{5}$ if $\ell = 1$).
    - Step ii: If $\ell < L-1$, compute $\mathsf{msig}' = \mathsf{Sig}.\mathbf{Sign}(\mathsf{msk}_{(\ell+1,j)}, t^*, (x^*_{(\ell,i)}, \mathsf{rte}^*))$ and $\mathsf{CT}^*_{(\ell+1,j)} = (x^*_{(\ell,i)}, \mathsf{rte}^*, \mathsf{msig}') \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,j)}, t)$.
    - Step iii: If $\ell = L-1$, compute $\mathsf{CT}^*_{(L,j)} \leftarrow (x^*_{(\ell,i)}, \perp, \perp) \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(L,j)}, t)$.
- Step 2: For each $j \in \mathsf{Output}_{(\ell,g)}$ such that $\mathsf{CT}_{(\ell+1,j)}$ has not been computed yet, compute filler ciphertexts as in $\mathsf{Hyb}^{(b)}_{6,\ell-1,5}$ (or $\mathsf{Hyb}^{(b)}_{5}$ if $\ell = 1$) if $t \neq t^*$. Else:
  - Step (a): Set $x = \perp_{\mathsf{filler}}$ and $\mathsf{rte} = \perp_{\mathsf{filler}}$.
  - Steps (b) and (c) are same as in $\mathsf{Hyb}^{(b)}_{6,\ell-1,5}$ (or $\mathsf{Hyb}^{(b)}_{5}$ if $\ell = 1$).
- Step 3 is same as in $\mathsf{Hyb}^{(b)}_{6,\ell-1,5}$ (or $\mathsf{Hyb}^{(b)}_{5}$ if $\ell = 1$).

Figure 6: The circuit $\mathsf{Gate}_{(\ell,g)}$ in hybrid experiment $\mathsf{Hyb}^{(b)}_{6,\ell,1}$.

Hybrid $\mathsf{Hyb}^{(b)}_{6,\ell,2}$ for each $\ell \in [L-1]$: This hybrid is identical to $\mathsf{Hyb}^{(b)}_{6,\ell,1}$, except that now, when generating all circuits for layer ℓ, the challenger punctures the hardcoded decryption keys and hardcodes the expected input ciphertexts and corresponding plaintexts for all *filler/honest* input wires for the challenge round t^*. In other words, for each *filler/honest* input wire (ℓ, i) that is on the route rte^*, do the following.

- The challenger hardcodes the expected input ciphertext $\overline{\mathsf{CT}}^*_{(\ell,i)} = (x^*_{(\ell,i)}, \mathsf{rte}^*, \mathsf{msig}^*_{(\ell,i)}) \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell,i)}, t^*)$ and compares the input ciphertext with hardcoded ciphertext instead of decrypting and performing subsequent checks.

- In addition, the challenger hardcodes the punctured PRF key $k_{(\ell,i)}$ at challenge round t^*: $k^*_{(\ell,i)} \leftarrow \mathsf{PRF}.\mathbf{Puncture}(k_{(\ell,i)}, t^*)$.

Formally, the behavior of gate $\mathsf{Gate}_{(\ell,g)}$ for all $g \in [G]$ as described in Figure 7.

Notation. Let $\mathsf{Input}_{(\ell,g)}$ and $\mathsf{Output}_{(\ell,g)}$ be the set of input and output wires of gate $\mathsf{Gate}_{(\ell,g)}$.

Hardcoded values. Same as in $\mathsf{Hyb}^{(b)}_{6,\ell,1}$, except that for each *filler/honest* wire $i \in \mathsf{Input}_{(\ell,g)}$, $\overline{\mathsf{CT}}^*_{(\ell,i)}$ is hardcoded along with the corresponding challenge message $x_{(\ell,i)}$ and route rte^*, and $\mathsf{msig}^*_{(\ell,i)}$ is not hardcoded anymore. In addition, the punctured PRF key $k^*_{(\ell,i)}$ is hardcoded instead of $k_{(\ell,i)}$, and is used for decrypting on wire (ℓ, i) during all rounds $t \neq t^*$.

Procedure. $\mathsf{Gate}_{(\ell,g)}$ takes as input a round t and a set of ciphertexts $\{\mathsf{CT}_{(\ell,i)} : i \in \mathsf{Input}_{(\ell,g)}\}$ corresponding to the input wires. Depending on the layer ℓ, it computes as follows.

- Step 1: For each input wire $i \in \mathsf{Input}_{(\ell,g)}$, if $t \neq t^*$ or wire i is *corrupt*, then, compute as in $\mathsf{Hyb}^{(b)}_{6,\ell,1}$. Else:
  - Steps (a) and (c) are same as in $\mathsf{Hyb}^{(b)}_{6,\ell,1}$.
  - Step (b): if $\mathsf{CT}^*_{(\ell,i)} \neq \overline{\mathsf{CT}}^*_{(\ell,i)}$, then abort. If wire i is filler, go to the next i.
- Steps 2 and 3 are same as in $\mathsf{Hyb}^{(b)}_{6,\ell,1}$.

Figure 7: The circuit $\mathsf{Gate}_{(\ell,g)}$ in hybrid experiment $\mathsf{Hyb}^{(b)}_{6,\ell,2}$.

Hybrid $\mathsf{Hyb}^{(b)}_{6,\ell,3}$ for each $\ell \in [L-1]$: This hybrid is identical to $\mathsf{Hyb}^{(b)}_{6,\ell,2}$, except that the challenger punctures the message signing keys for all *filler/honest* wires $(\ell+1, i)$ on the path $\mathsf{rte}^* = \mathsf{rte}^*_u$ (or $\mathsf{rte}^* = \perp_{\mathsf{filler}}$ in case of *filler* wire) at the challenge round t^* and challenge message $x^*_{(\ell+1,i)} = x^{(b)}_{u,t^*}$ (or $x^*_{(\ell+1,i)} = \perp_{\mathsf{filler}}$ in case of *filler* wire). That is, for each such *honest* wire $(\ell+1, i)$,

$$(\mathsf{msk}_{(\ell+1,i)}, \mathsf{mvk}_{(\ell+1,i)}) \leftarrow \mathsf{Sig}.\mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{tlen} + \mathsf{len} + n \cdot 2L \cdot \log(2n)),$$
$$\mathsf{msk}'_{(\ell+1,i)} \leftarrow \mathsf{Sig}.\mathbf{Puncture}(\mathsf{msk}_{(\ell+1,i)}, t^*, (x^*_{(\ell+1,i)}, \mathsf{rte}^*_u)).$$

Then, the behavior of gate $\mathsf{Gate}_{(\ell,g)}$ for all $g \in [G]$ is changed as described in Figure 8.

Hybrid $\mathsf{Hyb}^{(b)}_{6,\ell,4}$ for each $\ell \in [L-1]$: This hybrid is identical to $\mathsf{Hyb}^{(b)}_{6,\ell,3}$, except that the challenger uses the simulated setup to puncture message signing/verification keys for all *filler/honest* wires $(\ell+1, i)$ on the path rte^* at the challenge round t^* and challenge message $x^*_{(\ell,i)} = x^{(b)}_{u,t^*}$ (or $x^*_{(\ell,i)} = \perp_{\mathsf{filler}}$ in case of *filler* wire). That is, for each such wire $(1, i)$,

$$(\mathsf{msk}^*_{(\ell+1,i)}, \mathsf{mvk}^*_{(\ell+1,i)}) \leftarrow \mathsf{Sig}.\mathbf{SimSetup}(1^\lambda, \mathsf{tlen}, \mathsf{tlen} + \mathsf{len} + n \cdot 2L \cdot \log(2n), t^*, (x^*_{(\ell,i)}, \mathsf{rte}^*)).$$

Then, the challenger replaces the gates $\mathsf{Gate}_{(\ell,g)}$ and $\mathsf{Gate}_{(\ell+1,g)}$ for all $g \in [G]$ as described in Figure 9 and Figure 10.

Notation. Let $\mathsf{Input}_{(\ell,g)}$ and $\mathsf{Output}_{(\ell,g)}$ be the set of input and output wires of gate $\mathsf{Gate}_{(\ell,g)}$.

Hardcoded values. Same as in $\mathsf{Hyb}^{(b)}_{6,\ell,2}$, except that for each *filler/honest* wire $i \in \mathsf{Output}_{(\ell,g)}$ in the output layer $\ell+1$, the punctured message signing key $\mathsf{msk}'_{(\ell+1,i)}$ is hardcoded instead of $\mathsf{msk}_{(\ell+1,i)}$.

Procedure.

- Step 1: For each input wire $i \in \mathsf{Input}_{(\ell,g)}$, if $t \neq t^*$ or wire i is *corrupt*, compute as in $\mathsf{Hyb}^{(b)}_{6,\ell,2}$. Else:
  - Steps (a), (b) are same as in $\mathsf{Hyb}^{(b)}_{6,\ell,2}$.
  - Step (c): **Prepare the output ciphertext** $\mathsf{CT}_{(\ell+1,j_{\ell+1})}$**:**
    - Steps i and iii are same as in $\mathsf{Hyb}^{(b)}_{6,\ell,2}$.
    - Step ii: If $\ell < L-1$, compute $\mathsf{msig}' = \mathsf{Sig}.\mathbf{PSign}(\mathsf{msk}'_{(\ell+1,j)}, t^*, (x^{(b)}_{u,t^*}, \mathsf{rte}^*_u))$ and $\mathsf{CT}^*_{(\ell+1,j)} = (x^{(b)}_{u,t^*}, \mathsf{rte}^*_u, \mathsf{msig}') \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,j)}, t)$.
- Step 2: For each $j \in \mathsf{Output}_{(\ell,g)}$ such that $\mathsf{CT}_{(\ell+1,j)}$ has not been computed yet, compute filler ciphertexts as in $\mathsf{Hyb}^{(b)}_{6,\ell,2}$ if $t \neq t^*$. Else:
  - Step (a): Set $x = \bot_{\mathsf{filler}}$ and $\mathsf{rte} = \bot_{\mathsf{filler}}$.
  - Step (b): Compute $\mathsf{msig}' = \mathsf{Sig}.\mathbf{PSign}(\mathsf{msk}'_{(\ell+1,j)}, t^*, (x, \mathsf{rte}))$.
  - Step (c): Compute $\mathsf{CT}^*_{(\ell+1,j)} \leftarrow (x, \mathsf{rte}, \mathsf{msig}') \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,j)}, t^*)$.
- Step 3 is same as in $\mathsf{Hyb}^{(b)}_{6,\ell,2}$.

Figure 8: The circuit $\mathsf{Gate}_{(\ell,g)}$ in hybrid experiment $\mathsf{Hyb}^{(b)}_{6,\ell,3}$.

Notation. Let $\mathsf{Input}_{(\ell,g)}$ and $\mathsf{Output}_{(\ell,g)}$ be the set of input and output wires of gate $\mathsf{Gate}_{(\ell,g)}$.

Hardcoded values. Same as in $\mathsf{Hyb}^{(b)}_{6,\ell,3}$ except that for each *filler/honest* wire $i \in \mathsf{Output}_{(\ell,g)}$, message signing key $\mathsf{msk}^*_{(\ell+1,i)}$ is hardcoded instead of $\mathsf{msk}'_{(\ell+1,i)}$.

Procedure. Same as in $\mathsf{Hyb}^{(b)}_{6,\ell,3}$.

Figure 9: The circuit $\mathsf{Gate}_{(\ell,g)}$ in hybrid experiment $\mathsf{Hyb}^{(b)}_{6,\ell,4}$.

Notation. Let $\mathsf{Input}_{(\ell+1,g)}$ and $\mathsf{Output}_{(\ell+1,g)}$ be the set of input and output wires of gate $\mathsf{Gate}_{(\ell+1,g)}$.

Hardcoded values. Same as in $\mathsf{Hyb}^{(b)}_{6,\ell,3}$ except that for each *filler/honest* wire $i \in \mathsf{Input}_{(\ell+1,g)}$, message verification key $\mathsf{mvk}^*_{(\ell+1,i)}$ is hardcoded instead of $\mathsf{mvk}_{(\ell+1,i)}$.

Procedure. Same as in $\mathsf{Hyb}^{(b)}_{6,\ell,3}$.

Figure 10: The circuit $\mathsf{Gate}_{(\ell+1,g)}$ in hybrid experiment $\mathsf{Hyb}^{(b)}_{6,\ell,4}$.

Hybrid $\mathsf{Hyb}^{(b)}_{6,\ell,5}$ for each $\ell \in [L-1]$: This hybrid is identical to $\mathsf{Hyb}^{(b)}_{6,\ell,4}$, except that now, when generating all circuits for layer ℓ, for all *filler/honest* wires $(\ell+1, i)$ on the path rte^* the challenger hardcodes the expected output ciphertexts for the challenge round t^*: $\overline{\mathsf{CT}}^*_{(\ell+1,i)} = (x^{(b)}_{u,t^*}, \mathsf{rte}^*_u, \mathsf{msig}^*_{(\ell+1,i)}) \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,i)}, t^*)$ if $\ell < L-1$, else $\overline{\mathsf{CT}}^*_{(L,i)} = (x^{(b)}_{u,t^*}, \perp, \perp) \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(L,i)}, t^*)$. Then, instead of dynamically computing the output ciphertext inside the gate, the challenger simply uses the hardcoded ciphertext. In addition, the challenger hardcodes punctured keys $k^*_{(\ell+1,i)} \leftarrow \mathsf{PRF}.\mathbf{Puncture}(k_{(\ell,i)}, t^*)$ for the output wires into $\mathsf{Gate}_{(\ell,g)}$.

At this point, since both honest and filler wires are dealt with by comparing the inputs to fixed ciphertexts, and outputting fixed ciphertexts, the gate does not need to know which wires are honest and which are fillers during round t^*. We change the flow of the program to reflect this.

Formally, the behavior of gate $\mathsf{Gate}_{(\ell,g)}$ for all $g \in [G]$ is as described in Figure 11.

Notation. Let $\mathsf{Input}_{(\ell,g)}$ and $\mathsf{Output}_{(\ell,g)}$ be the set of input and output wires of gate $\mathsf{Gate}_{(\ell,g)}$.

Hardcoded values. Same as $\mathsf{Hyb}^{(b)}_{6,\ell,4}$ except that for each *filler/honest* wire $i \in \mathsf{Output}_{(\ell,i)}$ in the output layer $\ell+1$, the punctured PRF key $k^*_{(\ell+1,i)}$ is hardcoded instead of $k_{(\ell+1,i)}$.

Procedure. $\mathsf{Gate}_{(\ell,g)}$ takes as input a round t and a set of ciphertexts $\{\mathsf{CT}_{(\ell,i)} : i \in \mathsf{Input}_{(\ell,g)}\}$ corresponding to the input wires. Depending on the layer ℓ, it computes as follows.

1. Step 1: For each input wire $i \in \mathsf{Input}_{(\ell,g)}$, if $t \neq t^*$ or wire i is *corrupt*, compute as in $\mathsf{Hyb}^{(b)}_{6,\ell,5}$. Else:
   (a) If $\ell = 1$ and $i \in F_g$, continue to next i. //This is a filler element and is ignored.
   (b) **Authenticate the message/route:** If $\mathsf{CT}^*_{(\ell,i)} \neq \overline{\mathsf{CT}}^*_{(\ell,i)}$, then abort.
2. For each $j \in \mathsf{Output}_{(\ell,g)}$ such that $\mathsf{CT}_{(\ell+1,j)}$ has not been computed yet, compute filler ciphertexts as in $\mathsf{Hyb}^{(b)}_{6,\ell,5}$ if $t \neq t^*$. Else compute *filler/honest* ciphertexts as $\mathsf{CT}^*_{(\ell+1,j)} = \overline{\mathsf{CT}}^*_{(\ell+1,j)}$.
3. Output $\{\mathsf{CT}_{(\ell+1,i)} : i \in \mathsf{Output}_{(\ell,g)}\}$.

Figure 11: The circuit $\mathsf{Gate}_{(\ell,g)}$ in hybrid experiment $\mathsf{Hyb}^{(b)}_{6,\ell,5}$.

Hybrid $\mathsf{Hyb}^{(b)}_7$: This hybrid is same as $\mathsf{Hyb}^{(b)}_{6,L-1,5}$ except that the challenger changes the hardcoded ciphertexts as follows: In gates $\mathsf{Gate}_{(L-1,g)}$ for all $g \in [G]$, for all *honest* output wires $i \in \mathsf{Output}_{(L-1,g)}$, if the corresponding receiver is *corrupt*, then, do not change anything. Else, if it is an *honest* output wire whose corresponding receiver is *honest*, or if it is *filler* wire $i \in \mathsf{Output}_{(L-1,g)}$, or if it *filler/honest* wire i in any $\mathsf{Gate}_{(\ell,g)}$ for any $\ell < L-1$ and for any $g \in [G]$, then, change the hardcoded ciphertext to be encryption of $\perp_{\mathsf{filler}}$. That is, for wire (ℓ, i), set $\overline{\mathsf{CT}}^*_{(\ell,i)} = \mathit{random}$.

Hybrid $\mathsf{Hyb}^{(b)}_8$: This hybrid is same as $\mathsf{Hyb}^{(b)}_7$ except that the challenger unpunctures all the message verification keys that were previously puncutured: for all the message signing/verification keys for all *filler/honest* wires (ℓ, i) on the path rte^* in all the layers, the challenger uses setup and puncture algorithms instead of simulated setup at the challenge round t^* and challenge message $x^*_{(L,i)} = x^{(b)}_{u,t^*}$ (or $x^*_{(L,i)} = \perp_{\mathsf{filler}}$ in case of *filler* wire). That is, for each such wire (ℓ, i),

$$(\mathsf{msk}_{(\ell,i)}, \mathsf{mvk}_{(\ell,i)}) \leftarrow \mathsf{Sig}.\mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{tlen} + \mathsf{len} + n \cdot 2L \cdot \log(2n)),$$
$$\mathsf{msk}'_{(\ell,i)} \leftarrow \mathsf{Sig}.\mathbf{Puncture}(\mathsf{msk}_{(\ell,i)}, t^*, (x^*_{(L,i)}, \mathsf{rte}^*)).$$

Then, for each such wire (ℓ, i), whenever the challenger has to compute message signature for any round for a *filler/honest* wire, it computes as: $\mathsf{msig}_{(\ell,i)} = \mathsf{Sig}.\mathbf{PSign}(\mathsf{msk}'_{(\ell,i)}, \cdot, \cdot)$.

Hybrid $\mathsf{Hyb}^{(b)}_9$: This hybrid is same as $\mathsf{Hyb}^{(b)}_8$ except that the challenger unpunctures all the message signing keys that were previously puncutured: for all the message signing/verification keys for all *filler/honest* wires (ℓ, i) on the path rte^* in all the layers, the challenger uses setup algorithm only and does not puncture anymore at the challenge round t^* and challenge message $x^*_{(L,i)}$. That is, for each such wire (ℓ, i),

$$(\mathsf{msk}_{(\ell,i)}, \mathsf{mvk}_{(\ell,i)}) \leftarrow \mathsf{Sig}.\mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{tlen} + \mathsf{len} + n \cdot 2L \cdot \log(2n)).$$

Then, for each such wire (ℓ, i), whenever the challenger has to compute message signature for any round for a *filler/honest* wire, it computes as: $\mathsf{msig}_{(\ell,i)} = \mathsf{Sig}.\mathbf{Sign}(\mathsf{msk}_{(\ell,i)}, \cdot, \cdot)$.

To summarize, at this point the circuit $\mathsf{Gate}_{(\ell,g)}$ for all $\ell \in [L-1]$ and $g \in [G]$ is as in Figure 12 and Figure 13.

Notation. Let $\mathsf{Input}_{(\ell,g)}$ and $\mathsf{Output}_{(\ell,g)}$ be the set of input and output wires of gate $\mathsf{Gate}_{(\ell,g)}$.

Hardcoded values. $\mathsf{Gate}_{(\ell,g)}$ has hardcoded the following values:

- For each wire $i \in \mathsf{Input}_{(\ell,g)}$ in layer ℓ:
  - if *corrupt*: the PRF key $k_{(\ell,i)}$, the message verification key $\mathsf{mvk}_{(\ell,i)}$, the route verification key $\mathsf{rvk}^*_{(\ell,i)}$, the expected route rte^*_u.
  - if *filler/honest*: the PRF key $k^*_{(\ell,i)}$, the message verification key $\mathsf{mvk}_{(\ell,i)}$, the route verification key $\mathsf{rvk}_{(\ell,i)}$, the expected challenge ciphertext $\overline{\mathsf{CT}}^*_{(\ell,i)}$.
- For each wire $i \in \mathsf{Output}_{(\ell,g)}$ in layer $\ell+1$:
  - if *corrupt*: the PRF key $k_{(\ell+1,i)}$, the message signing key $\mathsf{msk}_{(\ell+1,i)}$.
  - if *filler/honest*: the PRF key $k^*_{(\ell+1,i)}$, the message signing key $\mathsf{msk}_{(\ell+1,i)}$, the expected challenge ciphertext $\overline{\mathsf{CT}}^*_{(\ell+1,i)}$.
- If $\ell = 1$, the set F_g which contains indices $i \in \mathsf{Input}_{(\ell,g)}$ such that $i \notin \{\pi_s(2k-1) : k \in [n]\}$.
- The challenge round t^*.

Figure 12: The circuit $\mathsf{Gate}_{(\ell,g)}$ in hybrid experiment $\mathsf{Hyb}_9^{(b)}$: notation and hardcoded values.

Procedure. $\mathsf{Gate}_{(\ell,g)}$ takes as input a round t and a set of ciphertexts $\{\mathsf{CT}_{(\ell,i)} : i \in \mathsf{Input}_{(\ell,g)}\}$ corresponding to the input wires. Depending on the layer ℓ, it computes as follows.

1. For each input wire $i \in \mathsf{Input}_{(\ell,g)}$:
   (a) If $\ell = 1$ and $i \in F_g$, continue to next i. //This is a *filler* wire and is ignored.
   (b) **Decrypt and authenticate the message/route:**
   If it is a *filler/honest* wire and $t = t^*$: If $\mathsf{CT}^*_{(\ell,i)} \neq \overline{\mathsf{CT}}^*_{(\ell,i)}$, then abort.
   If it is a (*corrupt* wire) or (*filler/honest* wire and $t \neq t^*$):
      i. Compute the plaintext $(x, \mathsf{rte}, \mathsf{msig}) = \mathsf{CT}_{(\ell,i)} \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell,i)}, t)$.
      ii. If msig is not a valid signature of (x, rte) w.r.t. $\mathsf{mvk}_{(\ell,i)}$ and round t, then abort.
      iii. If $x = \bot_{\mathsf{filler}}$ and $\mathsf{rte} = \bot_{\mathsf{filler}}$, go to the next i. // This is a *filler* wire and is ignored.
      iv. Parse rte as $((j_1, \ldots, j_L), (\mathsf{rsig}_1, \ldots, \mathsf{rsig}_L))$ and perform the following checks to authenticate the route rte:
      If it is a *corrupt* wire: If $\mathsf{rte} \neq \overline{\mathsf{rte}}^*_u$, abort. If $\mathsf{rsig}_\ell \neq \mathsf{rsig}^*_\ell$, abort.
      If it is an *honest* wire and $t \neq t^*$: Parse rte as $((j_1, \ldots, j_L), (\mathsf{rsig}_1, \ldots, \mathsf{rsig}_L))$ and perform the following checks to authenticate the route rte: If $j_\ell \neq i$ or $j_{\ell+1} \notin \mathsf{Output}_{(\ell,g)}$, then abort. If rsig_ℓ is not valid signature of $(j_1, \ldots, j_L)$ w.r.t. $\mathsf{rvk}_{(\ell,i)}$, then abort.
   (c) **Prepare the output ciphertext $\mathsf{CT}_{(\ell+1,j_{\ell+1})}$:**
   If it is a *corrupt* wire:
      i. Let $j = j^*_{\ell+1}$. If $\mathsf{CT}_{(\ell+1,j_{\ell+1})}$ has already been computed, then abort.
      ii. If $\ell < L-1$, compute $\mathsf{msig}' = \mathsf{Sig}.\mathbf{Sign}(\mathsf{msk}_{(\ell+1,j)}, t, (x, \mathsf{rte}^*_u))$ and $\mathsf{CT}_{(\ell+1,j)} = (x, \mathsf{rte}^*_u, \mathsf{msig}') \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,j)}, t)$.
      iii. If $\ell = L-1$ (output layer), compute $\mathsf{CT}_{(L,j)} \leftarrow (x, \bot, \bot) \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(L,j)}, t)$.
   If it is an *honest* wire and $t \neq t^*$:
      i. Let $j = j_{\ell+1}$. If $\mathsf{CT}_{(\ell+1,j_{\ell+1})}$ has already been computed, then abort.
      ii. If $\ell < L-1$ (intermediate layer), compute $\mathsf{msig}' = \mathsf{Sig}.\mathbf{Sign}(\mathsf{msk}_{(\ell+1,j)}, t, (x, \mathsf{rte}))$ and $\mathsf{CT}_{(\ell+1,j)} \leftarrow (x, \mathsf{rte}, \mathsf{msig}') \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,j)}, t)$.
      iii. If $\ell = L-1$ (output layer), compute $\mathsf{CT}_{(L,j)} \leftarrow (x, \bot, \bot) \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(L,j)}, t)$.
2. For each $j \in \mathsf{Output}_{(\ell,g)}$ such that $\mathsf{CT}_{(\ell+1,j)}$ has not been computed yet:
   If $t = t^*$, compute *filler/honest* ciphertexts as $\mathsf{CT}^*_{(\ell+1,j)} = \overline{\mathsf{CT}}^*_{(\ell+1,j)}$.
   If $t \neq t^*$, compute *filler* ciphertexts:
   (a) Set $x = \mathsf{rte} = \bot_{\mathsf{filler}}$.
   (b) Compute $\mathsf{msig}' = \mathsf{Sig}.\mathbf{Sign}(\mathsf{msk}_{(\ell+1,j)}, t, (x, \mathsf{rte}))$.
   (c) Compute $\mathsf{CT}_{(\ell+1,j)} \leftarrow (x, \mathsf{rte}, \mathsf{msig}') \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,j)}, t)$.
3. Output $\{\mathsf{CT}_{(\ell+1,i)} : i \in \mathsf{Output}_{(\ell,g)}\}$.

Figure 13: The circuit $\mathsf{Gate}_{(\ell,g)}$ in hybrid experiment $\mathsf{Hyb}^{(b)}_9$: procedure.

5.2 Proofs of Indistinguishability

Let the number of corrupt senders be $\theta = |\mathcal{K}_S| \leq n$.

Claim 5.4. *For $b \in \{0,1\}$, adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$ are identical.*

Proof. The difference between $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$ is that the for all the *corrupt* wires, the route signing keys are unpunctured in the former and punctured in the latter hybrid experiment. While no route signing keys are in the view of the adversary, the adversary does get route signatures for corrupt senders and these are computed differently across two hybrids. In the former, they are computed using $\mathsf{Sig}.\mathbf{Sign}$ algorithm and in the latter they are computed using $\mathsf{Sig}.\mathbf{PSign}$ algorithm. It follows from the correctness of the SSU signature scheme as defined in Section 4 that the input/output behaviour of these two algorithms is identical for all constrained points. As these signatures are generated by the challenger for some constrained points, hence, adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$ are identical. □

Claim 5.5. *For $b \in \{0,1\}$, assuming the SSU signature scheme satisfies computational indistinguishability of simulated setup, adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_2^{(b)}$ are computationally indistinguishable.*

Proof. The difference between hybrids $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_2^{(b)}$ is that in hybrid $\mathsf{Hyb}_1^{(b)}$, for all the *corrupt* wires, the route signing/verification key pair $(\mathsf{rsk}'_{(\ell,i)}, \mathsf{rvk}_{(\ell,i)})$ is used where the signing key is punctured and the verification key is unpunctured. Whereas in hybrid $\mathsf{Hyb}_2^{(b)}$, for all the *corrupt* wires, the route signing/verification key pair $(\mathsf{rsk}^*_{(\ell,i)}, \mathsf{rvk}^*_{(\ell,i)})$ is used where both the keys are punctured and are generated using simulated setup. There are θ *corrupt* wires in each layer $\ell = 1, \ldots, L$. For the sake of simplicity, call these total of $L \cdot \theta$ wires to be $w_1, \ldots, w_{L\cdot\theta}$. We will show that the adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_2^{(b)}$ are computationally indistinguishable via a sequence of hybird $\mathsf{H}_{1,0}^{(b)}, \ldots, \mathsf{H}_{1,L\cdot\theta}^{(b)}$ where in $\mathsf{H}_{1,i}$, for wire w_j, the route signing/verification key pair $(\mathsf{rsk}^*_{w_j}, \mathsf{rvk}^*_{w_j})$ is used if $j \leq i$, else the pair $(\mathsf{rsk}'_{w_j}, \mathsf{rvk}_{w_j})$ is used. With this sequence, observe that $\mathsf{H}_{1,0}^{(b)}$ is identical to $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{H}_{1,L\cdot\theta}^{(b)}$ is identical to $\mathsf{Hyb}_2^{(b)}$. We will show that $\mathsf{H}_{1,i-1}^{(b)}$ and $\mathsf{H}_{1,i}^{(b)}$ are computationally indistinguishable for all $i \in [L \cdot \theta]$ and then, by triangle inequality it follows that $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_2^{(b)}$ are computationally indistinguishable.

All that remains to show now is that for all $i \in [L \cdot \theta]$, $\mathsf{H}_{1,i-1}^{(b)}$ and $\mathsf{H}_{1,i}^{(b)}$ are computationally indistinguishable. Let wire w_i be on the route $\overline{\mathsf{rte}}^*_u$ for some $u \in \mathcal{K}_S$. Observe that the difference between the two hybrids is that for wire w_i in $\mathsf{H}_{1,i-1}^{(b)}$, the route signing/verification key pair $(\mathsf{rsk}'_{w_i}, \mathsf{rvk}_{w_i})$ is used, whereas in $\mathsf{H}_{1,i}^{(b)}$, the route signing/verification key pair $(\mathsf{rsk}^*_{w_i}, \mathsf{rvk}^*_{w_i})$ is used. Then, we can create a simple reduction from the computational indistinguishability of the two hybrids to the computational indistinguishability of simulated setup of SSU signature scheme which states that the following two distibutions are computationally indistinguishable.

1. Let $(\mathsf{rsk}_{w_i}, \mathsf{rvk}_{w_i}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len})$, $\mathsf{rsk}'_{w_i} \leftarrow \mathbf{Puncture}(\mathsf{rsk}, 1, \overline{\mathsf{rte}}^*_u)$, and output $(\mathsf{rsk}'_{w_i}, \mathsf{rvk}_{w_i})$.
2. Let $(\mathsf{rsk}^*_{w_i}, \mathsf{rvk}^*_{w_i}) \leftarrow \mathbf{SimSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, 1, \overline{\mathsf{rte}}^*_u)$, and output $(\mathsf{rsk}^*_{w_i}, \mathsf{rvk}^*_{w_i})$.

□

Remark 5.6. *Observe that in $\mathsf{H}_{1,i-1}^{(b)}$ and $\mathsf{H}_{1,i}^{(b)}$ both, the adversary's view never contains rsk'_{w_i} or $\mathsf{rsk}^*_{w_i}$, but only route signatures computed using these signing keys. Hence, while the adversary $\mathcal{A}$'s views in the two hybrids can be simulated using these keys, a weaker form would suffice where the* *adversary only has oracle access to the signing keys. But, we still use this stronger form to keep the signature scheme same for message as well as route signatures and as we will see later , making such a switch for message signing keys indeed requires this stronger form where the adversary has access to some punctured message signing keys. Looking ahead this would be required because unlike route signing keys, message signing keys are hardcoded in the obfuscated gates provided to the adversary $\mathcal{A}$.*

Claim 5.7. *For $b \in \{0,1\}$, assuming that the SSU signature scheme is deterministic and statistically unforgeable and the indistinguishability obfuscation scheme is secure, adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_2^{(b)}$ and $\mathsf{Hyb}_3^{(b)}$ are computationally indistinguishable.*

Proof. The difference between the hybrids $\mathsf{Hyb}_2^{(b)}$ and $\mathsf{Hyb}_3^{(b)}$ is in the way route authentication checks are performed for all *corrupt* wires in all the obfuscated gates. In particular, for all gates $\mathsf{Gate}_{(\ell,g)}$ for $\ell \in [L-1]$ and $g \in [G]$, after decryption of incoming ciphertext on a *corrupt* wire (ℓ, i) and message authentication of the plaintext (containing route information $\mathsf{rte}_u = (\overline{\mathsf{rte}}_u = (j_1, \ldots, j_L), (\mathsf{rsig}_1, \ldots, \mathsf{rsig}_L))$ for some user $u \in \mathcal{K}_S$), the route authentication in hybrid $\mathsf{Hyb}_2^{(b)}$ is performed by checking

$$j_\ell = i, j_{\ell+1} \in \mathsf{Output}_{(\ell,i)}, \mathsf{Sig}.\mathbf{Vf}(\mathsf{rvk}^*_{(\ell,i)}, \overline{\mathsf{rte}}_u, 1, \mathsf{rsig}_\ell) = 1. \tag{1}$$

If any of these checks fail, the circuit aborts further computation. On the other hand, the route authentication in hybrid $\mathsf{Hyb}_3^{(b)}$ is performed by checking

$$\overline{\mathsf{rte}}_u = \overline{\mathsf{rte}}^*_u, \mathsf{rsig}_\ell = \mathsf{rsig}^*_\ell, \tag{2}$$

where the expected route for this wire $\mathsf{rte}^*_u = (\overline{\mathsf{rte}}^*_u = (j^*_1, \ldots, j^*_L), (\mathsf{rsig}^*_1, \ldots, \mathsf{rsig}^*_L))$ is hardcoded in the gate circuit. If we can argue that for all gates $\mathsf{Gate}_{(\ell,g)}$ for $\ell \in [L-1]$ and $g \in [G]$, the gate circuit in the two hybrid experiments have identical input/output behaviour, then, the computational indistinguishability of the adversary $\mathcal{A}$'s views in the two hybrids follows from the security of the indistinguishability obfuscation scheme.

All that remains to be shown is that for any gate $\mathsf{Gate}_{(\ell,g)}$, the input/output behaviour of the circuits in the two hybrids is indeed identical. In other words, we want to show that it is equivalent to check either Equation (1) or Equation (2). If Equation (2) is satisfied, then, it is straighforward to observe that Equation (1) is also satisfied. It is non-trivial to see that whenever Equation (1) is satisfied, then, Equation (2) is also satisfied. To see this, notice that $\mathsf{rvk}^*_{(\ell,i)}$ is a punctured route verification key and satisfies statistical unforgeability at round 1 and value $(j^*_1, \ldots, j^*_L)$ as defined in Definition 2.2. Hence, the signature verification algorithm will only accept signature rsig^*_ℓ for $(j^*_1, \ldots, j^*_L)$ and no other signature for this or any other route. Then, it follows that Equation (2) is also satisfied. □

Claim 5.8. *For $b \in \{0,1\}$, adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_3^{(b)}$ and $\mathsf{Hyb}_4^{(b)}$ are identical.*

Proof. The difference between $\mathsf{Hyb}_3^{(b)}$ and $\mathsf{Hyb}_4^{(b)}$ is that the for all the *filler/honest* wires $(1, i)$, the message signing keys are unpunctured in the former and punctured in the latter hybrid experiment at the challenge rount t^* and the respective challenge plaintexts $\tilde{x}^* = (x^{(b)}_{u,t^*}, \mathsf{rte}^{(b)}_u)$ in case of *honest* wire for some $u \in \mathcal{H}_S$ or $\tilde{x}^* = (\perp_{\mathsf{filler}}, \perp_{\mathsf{filler}})$ in case of *filler* wire. But, as none of these message signing keys are in the view of the adversary, hence, it follows that the adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_3^{(b)}$ and $\mathsf{Hyb}_4^{(b)}$ are identical. □

Claim 5.9. *For $b \in \{0,1\}$, assuming the SSU signature scheme satisfies computational indistinguishability of simulated setup, adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_4^{(b)}$ and $\mathsf{Hyb}_5^{(b)}$ are computationally indistinguishable.*

Proof. The difference between hybrids $\mathsf{Hyb}_4^{(b)}$ and $\mathsf{Hyb}_5^{(b)}$ is that in hybrid $\mathsf{Hyb}_5^{(b)}$, for all the *filler/honest* wires $(1,i)$, the message signing/verification key pair $(\mathsf{msk}'_{(1,i)}, \mathsf{mvk}_{(1,i)})$ is used where the signing key is punctured and the verification key is unpunctured. Whereas in hybrid $\mathsf{Hyb}_5^{(b)}$, for all the *filler/honest* wires, the message signing/verification key pair $(\mathsf{msk}^*_{(1,i)}, \mathsf{mvk}^*_{(1,i)})$ is used where both the keys are punctured and are generated using simulated setup. Suppose that there are $\eta < 2n$ *filler/honest* wires in the first layer $\ell = 1$. For the sake of simplicity, call these η wires to be $w_1, \dots, w_\eta$. We will show that the adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_4^{(b)}$ and $\mathsf{Hyb}_5^{(b)}$ are computationally indistinguishable via a sequence of hybird $\mathsf{H}_{4,0}^{(b)}, \dots, \mathsf{H}_{4,\eta}^{(b)}$ where in $\mathsf{H}_{4,i}$, for wire w_j, the message signing/verification key pair $(\mathsf{msk}^*_{w_j}, \mathsf{mvk}^*_{w_j})$ is used if $j \le i$, else the pair $(\mathsf{msk}'_{w_j}, \mathsf{mvk}_{w_j})$ is used. With this sequence, observe that $\mathsf{H}_{4,0}^{(b)}$ is identical to $\mathsf{Hyb}_4^{(b)}$ and $\mathsf{H}_{4,\eta}^{(b)}$ is identical to $\mathsf{Hyb}_5^{(b)}$. We will show that $\mathsf{H}_{4,i-1}^{(b)}$ and $\mathsf{H}_{4,i}^{(b)}$ are computationally indistinguishable for all $i \in [\eta]$ and then, by triangle inequality it follows that $\mathsf{Hyb}_4^{(b)}$ and $\mathsf{Hyb}_5^{(b)}$ are computationally indistinguishable.

All that remains to show now is that for all $i \in [\eta]$, $\mathsf{H}_{4,i-1}^{(b)}$ and $\mathsf{H}_{4,i}^{(b)}$ are computationally indistinguishable. Observe that the difference between the two hybrids is the treatment of message signing/verification key pair for wire w_i. If wire w_i is an *honest* wire, then, the challenge plaintext is $\tilde{x}^* = (x_{u,t^*}^{(b)}, \mathsf{rte}_u^{(b)})$ for some user $u \in \mathcal{H}_S$. Else, if it is a *filler* wire, then, the challenge plaintext is $\tilde{x}^* = (\perp_{\mathsf{filler}}, \perp_{\mathsf{filler}})$. In $\mathsf{H}_{4,i-1}^{(b)}$, the message signing/verification key pair $(\mathsf{msk}'_{w_i}, \mathsf{mvk}_{w_i})$ is used, whereas in $\mathsf{H}_{4,i}^{(b)}$, the message signing/verification key pair $(\mathsf{msk}^*_{w_i}, \mathsf{mvk}^*_{w_i})$ is used. In both the above hybrids the puncturing is done at challenge round t^* and challenge plaintext $\tilde{x}^*$. We can create a simple reduction from the computational indistinguishability of the two hybrids to the computational indistinguishability of simulated setup of SSU signature scheme which states that the following two distibutions are computationally indistinguishable.

1. Let $(\mathsf{msk}_{w_i}, \mathsf{mvk}_{w_i}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{tlen}{+}\mathsf{len}{+}n{\cdot}2L{\cdot}\log(2n))$, $\mathsf{msk}'_{w_i} \leftarrow \mathbf{Puncture}(\mathsf{msk}_{w_i}, t^*, \tilde{x}^*)$, and output $(\mathsf{rsk}'_{w_i}, \mathsf{rvk}_{w_i})$.
2. Let $(\mathsf{msk}^*_{w_i}, \mathsf{mvk}^*_{w_i}) \leftarrow \mathbf{SimSetup}(1^\lambda, \mathsf{tlen}, \mathsf{tlen}{+}\mathsf{len}{+}n{\cdot}2L{\cdot}\log(2n), t^*, \tilde{x}^*)$, and output $(\mathsf{msk}^*_{w_i}, \mathsf{mvk}^*_{w_i})$.

□

Claim 5.10. *For $b \in \{0,1\}$ and $\ell \in [L-1]$, assuming the SSU signature scheme is a deterministic signature scheme and is statistically unforgeable at the simulated point and that the indistinguishability obfuscation scheme is secure, adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_{6,\ell-1,5}^{(b)}$ (or $\mathsf{Hyb}_5^{(b)}$ if $\ell = 1$) and $\mathsf{Hyb}_{6,\ell,1}^{(b)}$ are computationally indistinguishable.*

Proof. For the sake of simplicity, we define $\mathsf{Hyb}_{6,0,5}^{(b)} = \mathsf{Hyb}_5^{(b)}$ in this proof.

The only difference between $\mathsf{Hyb}_{6,\ell,1}^{(b)}$ and $\mathsf{Hyb}_{6,\ell-1,5}^{(b)}$ is in the way that the message signatures $\mathsf{msig}_{(\ell,i)}$ for all *filler/honest* wires $i \in \mathsf{Input}_{(\ell,g)}$ are verified inside of each obfuscated program $\overline{\mathsf{Gate}}_{(\ell,g)}$, $g \in [G]$ during round t^*. In $\mathsf{Hyb}_{6,\ell-1,5}^{(b)}$, this is done by using the verification algorithm $\mathsf{Sig}.\mathbf{Vf}$, whereas in $\mathsf{Hyb}_{6,\ell,1}^{(b)}$ this is done by checking $\mathsf{msig}_{(\ell,i)}$ is equal to to the hardcoded signature $\mathsf{msig}^*_{(\ell,i)}$, by checking that the signed message x is equal to the hardcoded message $x^*_{(\ell,i)}$. and by checking that the route rte is equal to the hardcoded route rte^*.

We prove indistinguishability via a sequence of subhybrids Hyb'_α, where $\alpha \in \{1, \ldots, 2n - \theta + 1\}$.

We define Hyb'_α to produce programs $\{\overline{\mathsf{Gate}_{(\ell,g)}}\}_{g\in[G]}$ which verify all message signatures $\mathsf{msig}_{(\ell,i)}$ as in $\mathsf{Hyb}^{(b)}_{6,\ell,1}$ when $i < \alpha$, and as in $\mathsf{Hyb}^{(b)}_{6,\ell-1,5}$ when $i \geq \alpha$. It is clear that $\mathsf{Hyb}'_1 = \mathsf{Hyb}^{(b)}_{6,\ell-1,5}$ and that $\mathsf{Hyb}'_{2n-\theta+1} = \mathsf{Hyb}^{(b)}_{6,\ell,1}$. Proving the claim thus reduces to proving indistinguishability between $\mathsf{Hyb}'_{\alpha-1}$ and Hyb'_α for all α. Note that the only difference between Hyb'_α and $\mathsf{Hyb}'_{\alpha-1}$ is in the behavior of a single $\overline{\mathsf{Gate}_{(\ell,g)}}$ for g such that $\alpha \in \mathsf{Input}_{(\ell,g)}$. Provided that the circuits obfuscated in $\mathsf{Hyb}'_{\alpha-1}$ and Hyb'_α to produce $\overline{\mathsf{Gate}_{(\ell,g)}}$ are functionally equivalent, a simple reduction to security of the obfuscation scheme shows that $\mathsf{Hyb}'_{\alpha-1}$ and Hyb'_α are computationally indistinguishable. Thus we have reduced the claim to showing functional equivalence of the two circuits in question.

To show functional equivalence, we observe that the only possible point at which the circuit outputs could differ is where $\mathsf{CT}_{(\ell,\alpha)}$ is an encryption of some message $(x_{(\ell,\alpha)}, \mathsf{rte}, \mathsf{msig}_{(\ell,\alpha)})$ with respect to round t^* which was not generated honestly. It is clear that in Hyb'_α, because the *filler/honest* message $x^*_{(\ell,i)}$, route rte^* and signature $\mathsf{msig}_{(\ell,\alpha)}$ are hardcoded, $\mathsf{Gate}_{(\ell,g)}$ rejects all dishonestly generated ciphertexts. Because of statistical unforgeability of the SSU signature scheme at point $(t^*, x^*_{(\ell,i)})$ with respect to the keypair $(\mathsf{msk}^*_{(\ell,\alpha)}, \mathsf{mvk}^*_{(\ell,\alpha)})$ which was generated by $\mathsf{Sig}.\mathbf{SimSetup}$, the only message that is accepted by $\mathsf{Sig}.\mathbf{Vf}$ is $(x^*_{(\ell,i)}, \mathsf{rte}^*)$, and by the fact that the scheme is a determistic signature scheme, the only accepted signature is $\mathsf{msig}^*_{(\ell,\alpha)}$. Thus, in $\mathsf{Hyb}'_{\alpha-1}$ at round t^*, $\mathsf{Gate}_{(\ell,g)}$ also rejects all inputs where $\mathsf{CT}_{(\ell,\alpha)}$ encrypts a message which was dishonestly generated. It follows that the circuits have identical behavior with respect to $\mathsf{CT}_{(\ell,\alpha)}$, and thus are functionally equivalent.

□

Claim 5.11. *For $b \in \{0,1\}$ and $\ell \in [L-1]$, assuming correctness of the punctured PRF scheme and the indistinguishability obfuscation scheme is secure, adversary $\mathcal{A}$'s views in $\mathsf{Hyb}^{(b)}_{6,\ell,1}$ and $\mathsf{Hyb}^{(b)}_{6,\ell,2}$ are computationally indistinguishable.*

Proof. The only difference between $\mathsf{Hyb}^{(b)}_{6,\ell,2}$ and $\mathsf{Hyb}^{(b)}_{6,\ell,1}$ is that in the obfuscated programs $\{\overline{\mathsf{Gate}_{(\ell,g)}}\}_g$ of $\mathsf{Hyb}^{(b)}_{6,\ell,2}$, the obfuscated programs honest/filler wire keys are punctured at t^*, and during round t^* honest/filler wire ciphertexts $\mathsf{CT}_{(\ell,i)}$ are not decrypted directly, and instead are compared with a fixed value $\overline{\mathsf{CT}}^*_{(\ell,i)}$, whose corresponding decryption is also hardcoded and used for the rest of the procedure.

We prove indistinguishability via a sequence of subhybrids Hyb'_α, where $\alpha \in \{1, \ldots, 2n - \theta + 1\}$.

We define Hyb'_α to produce programs $\{\overline{\mathsf{Gate}_{(\ell,g)}}\}_{g\in[G]}$ which are identical to those in $\mathsf{Hyb}^{(b)}_{6,\ell,1}$, except for the following differences:

- For all $i < \alpha$, hardcode punctured key $k^*_{(\ell,i)}$. Treat input honest/filler wire ciphertexts $\mathsf{CT}_{(\ell,i)}$ in the same way as in $\mathsf{Hyb}^{(b)}_{6,\ell,2}$ (i.e., do not decrypt directly, instead check whether $\mathsf{CT}_{(\ell,i)} = \overline{\mathsf{CT}}^*_{(\ell,i)}$, and if so, use corresponding hardcoded plaintext in the rest of the procedure.
- For all $i \geq \alpha$, hardcode non-punctured key $k_{(\ell,i)}$. Treat input honest/filler wire ciphertexts $\mathsf{CT}_{(\ell,i)}$ in the same way as in $\mathsf{Hyb}^{(b)}_{6,\ell,1}$ (i.e., decrypt directly and proceed as normal).

It is clear that $\mathsf{Hyb}'_1 = \mathsf{Hyb}^{(b)}_{6,\ell,1}$ and that $\mathsf{Hyb}'_{2n-\theta+1} = \mathsf{Hyb}^{(b)}_{6,\ell,2}$. Proving the claim thus reduces to proving indistinguishability between $\mathsf{Hyb}'_{\alpha-1}$ and Hyb'_α for all α. Note that the only difference between Hyb'_α and $\mathsf{Hyb}'_{\alpha-1}$ is in the circuit $\mathsf{Gate}_{(\ell,g)}$ which is used to generate a single obfuscated program $\overline{\mathsf{Gate}_{(\ell,g)}}$ for g such that $\alpha \in \mathsf{Input}_{(\ell,g)}$. Provided that the circuits obfuscated in $\mathsf{Hyb}'_{\alpha-1}$

and Hyb'_α to produce $\overline{\mathsf{Gate}}_{(\ell,g)}$ are functionally equivalent, a simple reduction to security of the obfuscation scheme shows that $\mathsf{Hyb}'_{\alpha-1}$ and Hyb'_α are computationally indistinguishable. Thus we have reduced the claim to showing functional equivalence of the two circuits in question.

To show functional equivalence, note that the behavior of $\mathsf{Gate}_{(\ell,g)}$ only differs across $\mathsf{Hyb}'_{\alpha-1}$ and Hyb'_α in terms of the behavior for the input wire $(\ell, \alpha - 1)$. We focus on this wire. In $\mathsf{Hyb}'_{\alpha-1}$, $\mathsf{Gate}_{(\ell,g)}$ only accepts exactly one value for $\mathsf{CT}_{(\ell,\alpha-1)}$ during round t^*. This is because $\mathsf{Gate}_{(\ell,g)}$ decrypts $\mathsf{CT}_{(\ell,\alpha-1)}$ and then compares the decrypted plaintext to fixed hardcoded values, and aborts if they are unequal. Since decryption is deterministic, only one such ciphertext $\mathsf{CT}_{(\ell,\alpha-1)}$ does not cause an abort. Since in Hyb'_α $\mathsf{Gate}_{(\ell,g)}$ hardcodes this exact ciphertext and the corresponding plaintext, the behavior with respect to round t^* is identical. Because of correctness of the punctured PRF key at all non-punctured points $t \neq t^*$, the behavior of $\mathsf{Gate}_{(\ell,g)}$ between $\mathsf{Hyb}'_{\alpha-1}$ and Hyb'_α across all other rounds are also identical. Thus, $\mathsf{Gate}_{(\ell,g)}$ is functionally equivalent across these two subhybrids.

□

Claim 5.12. *For $b \in \{0,1\}$ and $\ell \in [L-1]$, assuming correctness of the SSU signature scheme, and assuming the indistinguishability obfuscation scheme is secure, adversary $\mathcal{A}$'s views in $\mathsf{Hyb}^{(b)}_{6,\ell,2}$ and $\mathsf{Hyb}^{(b)}_{6,\ell,3}$ are computationally indistinguishable.*

Proof. The only difference between $\mathsf{Hyb}^{(b)}_{6,\ell,3}$ and $\mathsf{Hyb}^{(b)}_{6,\ell,2}$ is that in $\mathsf{Hyb}^{(b)}_{6,\ell,3}$, for honest/filler output wires $(\ell+1, i)$, the challenger hardcodes punctured message signing keys $\mathsf{msk}'_{(\ell+1,i)}$ instead of unpunctured ones. The obfuscated gates then use the punctured signing algorithm $\mathsf{Sig}.\mathbf{PSign}$ when signing outgoing messages at layer $\ell+1$.

We prove indistinguishability via a sequence of subhybrids Hyb'_α, where $\alpha \in \{1, \ldots, 2n-\theta+1\}$.

We define Hyb'_α to produce programs $\{\overline{\mathsf{Gate}}_{(\ell,g)}\}_{g\in[G]}$ which are identical to those in $\mathsf{Hyb}^{(b)}_{6,\ell,2}$, except for the following differences:

- For all $i < \alpha$, hardcode punctured signing key $\mathsf{msk}'_{(\ell+1,i)}$, and sign messages for output wire $(\ell+1, i)$ using $\mathsf{Sig}.\mathbf{PSign}$.
- For all $i \geq \alpha$, hardcode non-punctured signing key $\mathsf{msk}_{(\ell+1,i)}$, and sign messages for output wire $(\ell+1, i)$ using $\mathsf{Sig}.\mathbf{Sign}$.

It is clear that $\mathsf{Hyb}'_1 = \mathsf{Hyb}^{(b)}_{6,\ell,2}$ and that $\mathsf{Hyb}'_{2n-\theta+1} = \mathsf{Hyb}^{(b)}_{6,\ell,3}$. Proving the claim thus reduces to proving indistinguishability between $\mathsf{Hyb}'_{\alpha-1}$ and Hyb'_α for all α. Note that the only difference between Hyb'_α and $\mathsf{Hyb}'_{\alpha-1}$ is in the circuit $\mathsf{Gate}_{(\ell,g)}$ which is used to generate a single obfuscated program $\overline{\mathsf{Gate}}_{(\ell,g)}$ for g such that $\alpha \in \mathsf{Output}_{(\ell,g)}$. Provided that the circuits obfuscated in $\mathsf{Hyb}'_{\alpha-1}$ and Hyb'_α to produce $\overline{\mathsf{Gate}}_{(\ell,g)}$ are functionally equivalent, a simple reduction to security of the obfuscation scheme shows that $\mathsf{Hyb}'_{\alpha-1}$ and Hyb'_α are computationally indistinguishable. Thus we have reduced the claim to showing functional equivalence of the two circuits in question. Functional equivalence follows directly from correctness of the SSU signature scheme and the observation that no message is ever signed with respect to round t^* except for the exact message which was punctured.

□

Claim 5.13. *For $b \in \{0,1\}$ and $\ell \in [L-1]$, assuming the SSU signature scheme satisfies computational indistinguishability of simulated setup, adversary $\mathcal{A}$'s views in $\mathsf{Hyb}^{(b)}_{6,\ell,3}$ and $\mathsf{Hyb}^{(b)}_{6,\ell,4}$ are computationally indistinguishable.*

Proof. The only difference between $\mathsf{Hyb}^{(b)}_{6,\ell,4}$ and $\mathsf{Hyb}^{(b)}_{6,\ell,3}$ is that in $\mathsf{Hyb}^{(b)}_{6,\ell,4}$, the message signing and verification keys for wires $(\ell+1, i)$, $i \in [2n]$ are all generated using $\mathsf{Sig}.\mathbf{SimSetup}$, whereas in $\mathsf{Hyb}^{(b)}_{6,\ell,3}$ they are generated using $\mathsf{Sig}.\mathbf{Setup}$ and $\mathsf{Sig}.\mathbf{Puncture}$.

We prove indistinguishability via a sequence of subhybrids Hyb'_α, where $\alpha \in \{1, \ldots, 2n - \theta + 1\}$.

We define Hyb'_α to generate message keypairs $(\mathsf{msk}_{(\ell+1,i)}, \mathsf{mvk}_{(\ell+1,i)})$ using $\mathsf{Sig}.\mathbf{SimSetup}$ for all $i < \alpha$, and to generate $(\mathsf{msk}_{(\ell+1,i)}, \mathsf{mvk}_{(\ell+1,i)})$ using $\mathsf{Sig}.\mathbf{Setup}$ and $\mathsf{Sig}.\mathbf{Puncture}$ for all $i \geq \alpha$. It is clear that $\mathsf{Hyb}'_1 = \mathsf{Hyb}^{(b)}_{6,\ell,3}$ and that $\mathsf{Hyb}'_{2n-\theta+1} = \mathsf{Hyb}^{(b)}_{6,\ell,4}$. Proving the claim thus reduces to proving indistinguishability between $\mathsf{Hyb}'_{\alpha-1}$ and Hyb'_α for all α. Note that the only difference between Hyb'_α and $\mathsf{Hyb}'_{\alpha-1}$ is in the keypair $(\mathsf{msk}_{(\ell+1,\alpha-1)}, \mathsf{mvk}_{(\ell+1,\alpha)})$. As such, a simple reduction to computational indistinguishability of the simulated setup of the SSU signature scheme shows this indistinguishability. This proves the claim.

□

Claim 5.14. *For $b \in \{0,1\}$, assuming the indistinguishability obfuscation scheme is secure, adversary $\mathcal{A}$'s views in $\mathsf{Hyb}^{(b)}_{6,\ell,4}$ and $\mathsf{Hyb}^{(b)}_{6,\ell,5}$ are computationally indistinguishable.*

Proof. The only difference between $\mathsf{Hyb}^{(b)}_{6,\ell,5}$ and $\mathsf{Hyb}^{(b)}_{6,\ell,4}$ is in the behavior of circuits $\{\mathsf{Gate}_{(\ell,g)}\}_{g\in[G]}$ used to generated the obfuscated programs $\{\overline{\mathsf{Gate}}_{(\ell,g)}\}_{g\in[G]}$. In $\mathsf{Hyb}^{(b)}_{6,\ell,5}$ during round t^*, for honest/filler output wires $(\ell+1, i)$, $\mathsf{Gate}_{(\ell,g)}$ sets $\mathsf{CT}_{(\ell+1,i)}$ to be a hardcoded value $\overline{\mathsf{CT}}^*_{(\ell+1,i)}$, where $\overline{\mathsf{CT}}^*_{(\ell+1,i)} = (x^{(b)}_{u,t^*}, \mathsf{rte}^*_u, \mathsf{msig}^*_{(\ell+1,i)}) \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,i)}, t^*)$ if $\ell < L-1$, else $\overline{\mathsf{CT}}^*_{(\ell+1,i)} = (x^{(b)}_{u,t^*}, \bot, \bot) \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(L,i)}, t^*)$, and outputs this value for wire $(\ell+1, i)$.

We prove indistinguishability via a sequence of subhybrids Hyb'_α, where $\alpha \in \{1, \ldots, 2n - \theta + 1\}$.

We define Hyb'_α to produce programs $\{\overline{\mathsf{Gate}}_{(\ell,g)}\}_{g\in[G]}$ which are identical to those in $\mathsf{Hyb}^{(b)}_{6,\ell,4}$, except for the following difference. During round t^*:

- For all $i < \alpha$ such that $(\ell+1, i)$ is a filler/honest wire, output $\mathsf{CT}_{(\ell+1,i)} = \overline{\mathsf{CT}}^*_{(\ell+1,i)}$ for this wire.
- For all $i \geq \alpha$, act as in $\mathsf{Hyb}^{(b)}_{6,\ell,4}$.

It is clear that $\mathsf{Hyb}'_1 = \mathsf{Hyb}^{(b)}_{6,\ell,4}$ and that $\mathsf{Hyb}'_{2n-\theta+1} = \mathsf{Hyb}^{(b)}_{6,\ell,5}$. Proving the claim thus reduces to proving indistinguishability between $\mathsf{Hyb}'_{\alpha-1}$ and Hyb'_α for all α. Note that the only difference between Hyb'_α and $\mathsf{Hyb}'_{\alpha-1}$ is in the circuit $\mathsf{Gate}_{(\ell,g)}$ which is used to generate a single obfuscated program $\overline{\mathsf{Gate}}_{(\ell,g)}$ for g such that $\alpha \in \mathsf{Output}_{(\ell,g)}$. Provided that the circuits obfuscated in $\mathsf{Hyb}'_{\alpha-1}$ and Hyb'_α to produce $\overline{\mathsf{Gate}}_{(\ell,g)}$ are functionally equivalent, a simple reduction to security of the obfuscation scheme shows that $\mathsf{Hyb}'_{\alpha-1}$ and Hyb'_α are computationally indistinguishable. Thus we have reduced the claim to showing functional equivalence of the two circuits in question.

To show functional equivalence, observe that the only wire where the two circuits could possibly differ is the output wire $(\ell+1, \alpha-1)$. Functional equivalence then follows directly from the following facts:

- No corrupt wire from layer ℓ could be re-routed to a *filler/honest* wire in layer $\ell+1$ because of the hardwired routes for *corrupt* wires in hybrid $\mathsf{Hyb}^{(b)}_3$ (Figure 4).
- During round t^*, the plaintext $(x, \mathsf{rte}, \mathsf{msig}')$ encrypted by $\mathsf{Gate}_{(\ell,g)}$ to form $\mathsf{CT}_{(\ell+1,\alpha-1)}$ in $\mathsf{Hyb}'_{\alpha-1}$ has values x and rte are fixed and exactly equal to the corresponding plaintext components of $\overline{\mathsf{CT}}^*_{(\ell+1,\alpha-1)}$ in Hyb'_α, along with the fact that the SSU signature scheme

produces deterministic signatures (which means that the signature msig' is also fixed and equal to the corresponding component of $\overline{\mathsf{CT}}^*_{(\ell+1,\alpha-1)}$).

□

Claim 5.15. *For* $b \in \{0,1\}$*, assuming pseudorandomness of the PRF at punctured points, adversary* $\mathcal{A}$*'s views in* $\mathsf{Hyb}^{(b)}_{6,L-1,5}$ *and* $\mathsf{Hyb}^{(b)}_7$ *are computationally indistinguishable.*

Proof. The difference between the two hybrids is in how are the hardcoded ciphertexts for the challenge round t^* for all the *filler/honest* wires in all the layers (with the exception of *honest* outpout wires directed towards corrupt receievers in the last layer) computed by the challenger. In $\mathsf{Hyb}^{(b)}_{6,L-1,5}$, the hardcoded ciphertext for wire (ℓ, i) is of the form $\overline{\mathsf{CT}}^*_{(\ell,i)} = (\tilde{x}^*, \mathsf{msig}^*_{(\ell,i)}) \oplus y^*$, where $\mathsf{msig}^*_{(\ell,i)} = \mathsf{Sig}.\mathbf{PSign}(\mathsf{msk}^*_{(\ell,i)}, t^*, \tilde{x}^*)$ and $y^* = \mathsf{PRF}.\mathbf{Eval}(k_{(\ell,i)}, t^*)$. If it is an *honest* wire corresponding to route $\mathsf{rte}^{(b)}_u$ for some $u \in \mathcal{H}_S$, then, $\tilde{x}^* = (x^{(b)}_{u,t^*}, \mathsf{rte}^{(b)}_u)$. Else, if it is a *filler* wire, then, $\tilde{x}^* = (\perp_{\mathsf{filler}}, \perp_{\mathsf{filler}})$. On the other hand, in $\mathsf{Hyb}^{(b)}_7$, for a *filler/honest* wire (ℓ, i), $\overline{\mathsf{CT}}^*_{(\ell,i)} = random$. To argue the computationally indistinguishability of adversary's view in these two hybrids, notice that the hardcoded PRF key $k^*_{(\ell,i)}$ is punctured at t^*, whereas y^* is the PRF evaluation at exactly this punctured point. So, one can use the pseudorandomness of the PRF at punctured points to show the computational indistinguishability.

For a *filler/honest* wire (ℓ, i), we want to change from $\overline{\mathsf{CT}}^*_{(\ell,i)} = (\tilde{x}^*, \mathsf{msig}^*_{(\ell,i)}) \oplus y^*$ to $\overline{\mathsf{CT}}^*_{(\ell,i)} = random$, where $y^* = \mathsf{PRF}.\mathbf{Eval}(k_{(\ell,i)}, t^*)$. We first invoke the psuedorandomness of PRF at punctured points to change to $y^* = random'$. Then, we can change from $\overline{\mathsf{CT}}^*_{(\ell,i)} = (\tilde{x}^*, \mathsf{msig}^*_{(\ell,i)}) \oplus random'$ to $\overline{\mathsf{CT}}^*_{(\ell,i)} = random$ as both are identically distributed. This is exactly what we want in $\mathsf{Hyb}^{(b)}_7$. □

Claim 5.16. *For* $b \in \{0,1\}$*, assuming the SSU signature scheme satisfies computational indistinguishability of simulated setup, adversary* $\mathcal{A}$*'s views in* $\mathsf{Hyb}^{(b)}_7$ *and* $\mathsf{Hyb}^{(b)}_8$ *are computationally indistinguishable.*

Proof. Similar to Claim 5.9, except that here the message verification keys are changed from punctured to unpunctured for all the *filler/honest* wires (ℓ, i) in all the layers. □

Claim 5.17. *For* $b \in \{0,1\}$*, assuming the indistinguishability obfuscation scheme is secure, adversary* $\mathcal{A}$*'s views in* $\mathsf{Hyb}^{(b)}_8$ *and* $\mathsf{Hyb}^{(b)}_9$ *are computationally indistinguishable.*

Proof. The difference between the two hybrids is that in $\mathsf{Hyb}^{(b)}_8$, for all the *filler/honest* wires (ℓ, i), punctured message signing key $\mathsf{msk}'_{(\ell,i)}$ is used, whereas in $\mathsf{Hyb}^{(b)}_9$, unpunctured message signing key $\mathsf{msk}_{(\ell,i)}$ is used. We will argue that for all the circuits, $\mathsf{Gate}_{(\ell,g)}$, the input/output behaviour is still the same. Hence, the indistinguishability of the two hybrids follows through a sequence of intermediate hybrid transitions where we switch the circuit for one gate at a time and the computational indistinguishability of any two consequtive intermediate hybrids can be shown through a simple reduction to the security of the indistinguishability obfuscation scheme.

All that remains to be shown is that all the circuits $\mathsf{Gate}_{(\ell,g)}$ in the two hybrids have identical input/output behaviour. Observe that the only points where the circuits in the two hybrids may differ are precisely the points that were punctured. $\mathsf{msk}_{(\ell,i)}$ can be used to sign even for challenge round t^* and non-challenge messages $x \neq x^*$ but $\mathsf{msk}'_{(\ell,i)}$ can't sign for these points. But observe that in $\mathsf{Hyb}^{(b)}_9$, $\mathsf{msk}_{(\ell,i)}$ is never used to sign messages for the challenge round t^* as the hardcoded ciphertexts are directly used instead. Hence, it follows that all the circuits $\mathsf{Gate}_{(\ell,g)}$ in the two hybrids indeed have identical input/output behaviour. □

Claim 5.18. *Adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_9^{(0)}$ and $\mathsf{Hyb}_9^{(1)}$ are identical.*

Proof. Recall the formal description of circuits $\mathsf{Gate}_{(\ell,g)}$ described in Figure 12 and Figure 13. It suffices to argue that $\mathsf{Hyb}_9^{(b)}$ and $\mathsf{Hyb}_9^{(1-b)}$ contain no information about the challenge bit b. Observe that the gate circuit had no information about challenge bit b in the real world description. So, the only new sources of information about it could be the changes done to the circuit that are highlighted in the figures. We will now argue that all of them have no information about the challenge bit b.

- For each *corrupt* wire $i \in \mathsf{Input}_{(\ell,i)}$, punctured route verification $\mathsf{rvk}^*_{(\ell,i)}$ is hardwired. These keys are punctured at routes rte^*_u for some $u \in \mathcal{K}_S$. Notice that in **Setup***, these routes are sampled before routes are sampled for honest users. Hence, it follows, that the punctured route verification keys for corrupt senders do not contain any information about the challenge bit b. For these wires, rte^* is also hardcoded. The admissibility criteria dictates that for each $u \in \mathcal{K}_S$, $\pi^{(0)}(u) = \pi^{(1)}(u)$. Hence, the hardcoded routes for corrupt senders also have no information about the challenge bit b.

- For each *filler/honest* wire $i \in \mathsf{Input}_{(\ell,i)}$, the punctured PRF key $k^*_{(\ell,i)}$ is hardwired. This key is punctured at the challenge round t^*, and hence has no information about the challenge bit b. For these wires, the expected challenge ciphertext $\overline{\mathsf{CT}}^*_{(\ell,i)}$ is also hardcoded. As all of these are uniformly random values, hence, it follows that they have no information about the challenge bit b.

- For each *filler/honest* wire $i \in \mathsf{Output}_{(\ell,i)}$, the punctured PRF key $k^*_{(\ell+1,i)}$ is hardwired. This key is punctured at the challenge round t^*, and hence has no information about the challenge bit b. For these wires, the expected challenge ciphertext $\overline{\mathsf{CT}}^*_{(\ell+1,i)}$ is also hardcoded. If $\ell \neq L-1$, all of these are uniformly random values, hence, it follows that they have no information about the challenge bit b. If $\ell = L-1$ and the wire's destination is not a *corrupt* receiver, then also all of these are uniformly random values. Hence, it follows that they have no information about the challenge bit b. If $\ell = L-1$ and the wire's destination is a *corrupt* receiver, then, the hardcoded value is the what the *corrupt* receiver would expect. The admissibility criteria dictates that if two different honest senders are sending some message to a corrupt receiver in the two worlds $b = 0$ and $b = 1$, then, the message values by the two honest senders should be the same. Hence, it follows that the same ciphertext value is hardcoded in the two worlds and consequently, it leaks no information about the challenge bit b.

- It is possible that a wire could be *filler* when $b = 0$ and *honest* when $b = 1$. Hence, it could have different treatment by the circuit procedure in the two worlds. But notice that in both hybrids $\mathsf{Hyb}_9^{(0)}$ and $\mathsf{Hyb}_9^{(1)}$, for the challenge round $t = t^*$, the treatment of filler and honest wires is exactly the same. Hence, it leaks no information about the challenge bit b.

From the above analysis, it follows that adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_9^{(0)}$ and $\mathsf{Hyb}_9^{(1)}$ are identical. □

Acknowledgements. This work is in part supported by DARPA SIEVE grant, NSF grants 2212746, 2128519, 2044679, and 1704788, a faculty award from JP Morgan, a Packard fellowship, and an ONR grant under the grant number N00014-22-1-2064-P00001.

References

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

[ACN$^+$20] Gilad Asharov, T.-H. Hubert Chan, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Bucket oblivious sort: An extremely simple oblivious sort. In *SOSA*, 2020.

[AKTZ17] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. Mcmix: Anonymous messaging via secure multiparty computation. In *Usenix Security*, 2017.

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

[BBGN19a] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. Differentially private summation with multi-message shuffling. *CoRR*, abs/1906.09116, 2019.

[BBGN19b] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. The privacy blanket of the shuffle model. In *CRYPTO*, 2019.

[BDGM20] Zvika Brakerski, Nico Dottling, Sanjam Garg, and Giulio Malavolta. Factoring and pairings are not necessary for io: Circular-secure lwe suffices. Cryptology ePrint Archive, Report 2020/1024, 2020.

[BEM$^+$17] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnés, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. *CoRR*, abs/1710.00901, 2017.

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

[BGI$^+$01a] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. In *Annual international cryptology conference*, pages 1–18. Springer, 2001.

[BGI$^+$01b] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *International workshop on public key cryptography*, pages 501–519. Springer, 2014.

[BHMS21] Benedikt Bünz, Yuncong Hu, Shin'ichiro Matsuo, and Elaine Shi. Non-interactive differentially anonymous router. Cryptology ePrint Archive, Paper 2021/1242, 2021. https://eprint.iacr.org/2021/1242.

[BKO22] Paul Bunn, Eyal Kushilevitz, and Rafail Ostrovsky. Anonymous permutation routing. Cryptology ePrint Archive, Paper 2022/1353, 2022. https://eprint.iacr.org/2022/1353.

[BSW16] Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 792–821, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

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

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

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

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

[CL05] Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In *CRYPTO*, 2005.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography*, pages 468–497, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[CSU$^+$19] Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling, 04 2019.

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

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

[DS18] Jean Paul Degabriele and Martijn Stam. Untagging tor: A formal treatment of onion encryption. In *EUROCRYPT*, 2018.

[EFM$^+$19] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In *SODA*, 2019.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2013.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pages 464–479. IEEE Computer Society, 1984.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

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

[GKLX22] S. Dov Gordon, Jonathan Katz, Mingyu Liang, and Jiayu Xu. Spreading the privacy blanket: Differentially oblivious shuffling for differential privacy. In *ACNS*, 2022.

[GL89] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, page 25–32, New York, NY, USA, 1989. Association for Computing Machinery.

[GP20] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. Cryptology ePrint Archive, Report 2020/1010, 2020.

[GPV19] Badih Ghazi, Rasmus Pagh, and Ameya Velingker. Scalable and differentially private distributed aggregation in the shuffled model. *CoRR*, abs/1906.08320, 2019.

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

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[HKLR05] Mohammad T. Hajiaghayi, Robert D. Kleinberg, Tom Leighton, and Harald Räcke. Oblivious routing on node-capacitated and directed graphs. In *SODA*, 2005.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, page 163–172, New York, NY, USA, 2015. Association for Computing Machinery.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021.

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015*, pages 121–145, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

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

[R0̈2] Harald Räcke. Minimizing congestion in general networks. In *FOCS*, 2002.

[RS21] Vijaya Ramachandran and Elaine Shi. Data oblivious algorithms for multicores. In Kunal Agrawal and Yossi Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 373–384. ACM, 2021.

[SA19] Nigel P. Smart and Younes Talibi Alaoui. Distributing any elliptic curve based protocol. In *IMACC*, 2019.

[SBS02] Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. P5: A protocol for scalable anonymous communication. In *IEEE Symposium on Security and Privacy*, 2002.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 475–484, 2014.

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

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

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

[WW20] Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious LWE sampling. Cryptology ePrint Archive, Report 2020/1042, 2020.

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

Appendix

A Constrained PRF Construction

We construct a constrained PRF (CPRF) family $F_s : \{0,1\}^{\mathsf{rlen}+\mathsf{len}} \to \{0,1\}^{2(\mathsf{rlen}+\mathsf{len})}$ as defined in Section 3.4 based on the Goldreich-Goldwasswer-Micali tree-based PRFs [GGM86].

Recall that the GGM construction uses a length-doubling pseudorandom generator $G : \{0,1\}^{\ell} \to \{0,1\}^{2\ell}$. Denote the two halves of the output of the PRG G as $G(z) = G_0(z)G_1(z)$. Then, the PRF with seed s on input z whose binary decomposition is $z = z_1 z_2 \dots z_n$ (where z_1 is the most significant bit and z_n is the least significant bit) is defined as $F_s(z) = G_{z_n}(\dots G_{z_2}(G_{z_1}(s)))$.

A.1 Construction

Let $G : \{0,1\}^{\mathsf{rlen}+\mathsf{len}} \to \{0,1\}^{2(\mathsf{rlen}+\mathsf{len})}$ be a length-doubling PRG. Then, our construction is as follows.

- $\mathsf{sk} \leftarrow \mathbf{Gen}(1^\lambda, \mathsf{tlen}, \mathsf{len})$: Choose a uniformly random seed $s \xleftarrow{\$} \{0,1\}^{\mathsf{rlen}+\mathsf{len}}$ and output $\mathsf{sk} = s$.
- $\sigma \leftarrow \mathbf{Eval}(\mathsf{sk}, t, x)$: Let $t = t_1 \dots t_{\mathsf{tlen}}$ and $x = x_1 \dots x_{\mathsf{len}}$ be the binary decompositions of t and x. Compute $\sigma \leftarrow G_{x_{\mathsf{len}}}(\dots G_{x_1}(G_{t_{\mathsf{tlen}}}(\dots G_{t_1}(s))))$ and output σ.
- $\mathsf{sk}^* \leftarrow \mathbf{Constr}(\mathsf{sk}, t^*, [y{:}] \cup \{x^*\})$: We consider two cases based on the value of y.
  - <u>If $y > 0$.</u> Let $t^* = t^*_1 \dots t^*_{\mathsf{tlen}}$, $y - 1 = y_1 \dots y_{\mathsf{len}}$, and $x = x_1 \dots x_{\mathsf{len}}$ be the binary decompositions of t, $y-1$, and x. Further, for a bit b, denote by $\overline{b} = b \oplus 1$ its compliment bit. Let $z = G_{t^*_{\mathsf{tlen}}}(\dots G_{t^*_1}(s))$. Let $S = \{k | y_k = 0\}$ be the set of indices k of $y-1$ where $y_k = 0$. Let its size be $|S| = \ell \le \mathsf{len}$ and denote these indices as $i_1, \dots, i_\ell$ in increasing order. Compute $\sigma^* \leftarrow \mathbf{Eval}(\mathsf{sk}, t^*, x^*)$,

$$\begin{aligned}
\mathsf{sk}_1 &= G_{\overline{t^*_1}}(s), \\
\mathsf{sk}_2 &= G_{\overline{t^*_2}}(G_{t^*_1}(s)), \\
&\vdots \\
\mathsf{sk}_{\mathsf{tlen}} &= G_{\overline{t^*_{\mathsf{tlen}}}}(G_{t^*_{\mathsf{tlen}}-1}(\dots G_{t^*_1}(s))), \\
\mathsf{sk}_{\mathsf{tlen}+1} &= G_{\overline{y_{i_1}}}(G_{y_{i_1}-1}(\dots G_{y_1}(z))), \\
\mathsf{sk}_{\mathsf{tlen}+2} &= G_{\overline{y_{i_2}}}(G_{y_{i_2}-1}(\dots G_{y_1}(z))), \\
&\vdots \\
\mathsf{sk}_{\mathsf{tlen}+\ell} &= G_{\overline{y_{i_\ell}}}(G_{y_{i_\ell}-1}(\dots G_{y_1}(z))),
\end{aligned}$$

    and output $sk^* = (t^*, [y{:}] \cup \{x^*\}, \sigma^*, \mathsf{sk}_1, \dots, \mathsf{sk}_{\mathsf{tlen}+\ell})$.
  - <u>Else if $y = 0$.</u> In this case, output $sk^* = (t^*, [y{:}] \cup \{x^*\}, \sigma^*, \mathsf{sk}_1, \dots, \mathsf{sk}_{\mathsf{tlen}}, z)$.
- $\sigma \leftarrow \mathbf{CEval}(\mathsf{sk}^*, t, x)$: The output varies based on the value of t, y, and x as follows:
  - <u>If $t = t^*$ and $x \ne x^*$.</u> Output $\perp$.

– Else if $t = t^*$ and $x = x^*$. Output σ^*.

– Else if $t = t^*$, $x \in [y{:}]$ and $y > 0$. Let $y - 1 = y_1 \dots y_{\mathsf{len}}$, and $x = x_1 \dots x_{\mathsf{len}}$ be the binary decompositions of $y - 1$ and x. Let k be the smallest index for which $x_k \neq y_k$. As $x > y$, it must be the case that $x_k = 1, y_k = 0$. Hence, $k \in S = \{i_1, \dots, i_\ell\}$. Let $k = i_j$. In other words, $x_k = \overline{y_{i_j}}$. Output $\sigma \leftarrow G_{x_{\mathsf{len}}}(\dots G_{x_{k+1}}(\mathsf{sk}_{\mathsf{tlen}+j}))$.

– Else if $t = t^*$, $x \in [y{:}]$ and $y = 0$. Let $x = x_1 \dots x_{\mathsf{len}}$ be the binary decompositions of and x. Output $\sigma \leftarrow G_{x_{\mathsf{len}}}(\dots G_{x_1}(z))$.

– Else if $t \neq t^*$. Let $t = t_1 \dots t_{\mathsf{tlen}}$, $t^* = t^*_1 \dots t^*_{\mathsf{tlen}}$, and $x = x_1 \dots x_{\mathsf{len}}$ be the binary decomposition of t, t^* and x. Let j be the smallest index for which $t_j \neq t^*_j$. In other words, $t_j = \overline{t^*_j}$. Output $\sigma \leftarrow G_{x_{\mathsf{len}}}(\dots G_{x_1}(G_{t_{\mathsf{tlen}}}(\dots G_{t_{j+1}}(\mathsf{sk}_j))))$.

A.2 Security Proof

Lemma A.1. *Suppose that PRGs are sub-exponentially secure. Then, the above CPRF scheme is sub-exponentially secure as defined in Section 3.4.*

Proof. We prove that experiments $\mathsf{ExptCPRF}^{\mathcal{A},0}$ and $\mathsf{ExptCPRF}^{\mathcal{A},1}$ are sub-exponentially close via a sequence for hybrid experiments $\mathsf{Hyb}^{\mathcal{A},0}$, $\mathsf{Hyb}^{\mathcal{A},1}$, $\mathsf{Hyb}^{\mathcal{A},2}$. The hybrid sequence is similar to the ones used in proving security of GGM tree-based PRFs [GGM86] and functional pseudorandom functions [BGI14]. The experiment $\mathsf{Hyb}^{\mathcal{A},b}$ parametrized by $b \in \{0, 1, 2\}$ is as follows.

- The challenger honestly runs the **Gen** algorithm to obtain $\mathsf{sk} = s$.
- The adversary $\mathcal{A}$ can make the following queries adaptively:
  – **Eval**: $\mathcal{A}$ submits a pair (t, x). If $b = 0$ or $b = 2$, the challenger responds honestly by starting with s as the root node of the GGM tree and traversing down the path corresponding to $t||x$. If $b = 1$, the challenger responds with a random bit-string of length $2(\mathsf{tlen} + \mathsf{len})$ in a consistent manner as follows. It traverses down the path corresponding to $t||x$ and for each node on the path, it checks if the node is marked. If it is marked, it means that this node was already traversed through in an earlier query, so the challenger does not sample a uniformly random bit-string for the node. Else, the challenger marks the node and samples a uniformly random bit-string of appropriate length ($\mathsf{tlen} + \mathsf{len}$ for intermediate nodes and $2(\mathsf{tlen} + \mathsf{len})$ for leaf node) for this node. The challenger responds with the uniformly random value set corresponding to the leaf node on this path.

  – **Constr**: $\mathcal{A}$ submits a tuple (t^*, x^*, y). If $b = 0$ or $b = 2$, the challenger responds honestly by starting with s as the root node of the GGM tree and computing all the intermediate and leaf values in the tree as described in the construction. If $b = 1$, the challenger responds in a uniformly random and consistent manner as follows. For computing $\sigma^*, z, \mathsf{sk}_1, \dots, \mathsf{sk}_{\mathsf{tlen}+\ell}$, the challenger traverses down the respective paths in the tree and computes the uniformly random values for each node on the path based on whether that node is marked or not as described earlier for **Eval** queries and sets the response accordingly.

  – **Challenge**: $\mathcal{A}$ submits a challenge pair $(\widetilde{t}, \widetilde{x})$ satisfying the admissibility criteria as defined in the security definition in Section 3.4. If $b = 0$, the challenger responds honestly by starting with s as the root node of the GGM tree and traversing down the path corresponding to $t^*||x^*$. If $b = 1$, the challenger responds with a unformly random value in a consistent manner based on whether the leaf node corresponding to $\widetilde{t}||\widetilde{x}$ is marked or not as described earlier for **Eval** queries. If $b = 2$, the challenger responds with a random bit-string of length $2(\mathsf{tlen} + \mathsf{len})$.

- $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$, the experiment outputs b'.

Observe that $\mathsf{Hyb}^{\mathcal{A},0}$ is same as $\mathsf{ExptCPRF}^{\mathcal{A},0}$ and $\mathsf{Hyb}^{\mathcal{A},2}$ is same as $\mathsf{ExptCPRF}^{\mathcal{A},1}$. To complete the proof, we show that the views of the adversary in experiments $\mathsf{Hyb}^{\mathcal{A},0}$ and $\mathsf{Hyb}^{\mathcal{A},1}$ are sub-exponentially close in Lemma A.2 and those in $\mathsf{Hyb}^{\mathcal{A},1}$ and $\mathsf{Hyb}^{\mathcal{A},2}$ are sub-exponentially close in Lemma A.3. □

Lemma A.2. *Suppose that PRGs are sub-exponentially secure. Then, there exists a constant $0 < \gamma < 1$ such that for any probabilistic polynomial-time (p.p.t.)* admissible *adversary $\mathcal{A}$, for any len that is polynomially bounded in λ, there exists λ_0 such that for any $\lambda > \lambda_0$,*

$$|\Pr[\mathsf{Hyb}^{\mathcal{A},0}(1^\lambda, \mathsf{tlen}, \mathsf{len}) = 1] - \Pr[\mathsf{Hyb}^{\mathcal{A},1}(1^\lambda, \mathsf{tlen}, \mathsf{len}) = 1]| \leq 2^{-\lambda^\gamma}.$$

Proof. Suppose there exists an adversary $\mathcal{A}$ such that

$$|\Pr[\mathsf{Hyb}^{\mathcal{A},0}(1^\lambda, \mathsf{tlen}, \mathsf{len}) = 1] - \Pr[\mathsf{Hyb}^{\mathcal{A},1}(1^\lambda, \mathsf{tlen}, \mathsf{len}) = 1]| = \epsilon(1^\lambda, \mathsf{tlen}, \mathsf{len})$$

for some ϵ that is not sub-exponentially small. Then, we show that $\mathcal{A}$ can be used to construct a reduction $\mathcal{B}$ that can break the sub-exponetial security of PRGs. We consider a sequence of hybrid experiments $\mathsf{H}^{\mathcal{A},i}$ for $i \in \{0, 1, \dots, \mathsf{tlen} + \mathsf{len} + 1\}$ where i corresponds to the level of tree where $\mathcal{B}$ will place its challenge in its interation with $\mathcal{A}$.

The experiment $\mathsf{H}^{\mathcal{A},i}$ parametrized by $i \in \{0, 1, \dots, \mathsf{tlen} + \mathsf{len}\}$ is as follows.

- The challenger honestly runs the **Gen** algorithm to obtain $\mathsf{sk} = s$.
- The adversary $\mathcal{A}$ can make the following queries adaptively:
  - **Eval**: $\mathcal{A}$ submits a pair (t, x). The challenger traverses down the path corresponding to $t||x$ and for each node on the path upto level i (where root node is at level 0), it checks if the node is marked. If it is marked, it continues down the path. Else, the challenger marks the node and samples a uniformly random bit-string of appropriate length ($\mathsf{tlen} + \mathsf{len}$ for intermediate nodes and $2(\mathsf{tlen} + \mathsf{len})$ for leaf node) for this node. After level i, the challenger computes the output value by starting with the uniformly random value set at level i on this path and evaluating the PRG G for the rest of the path based on the $(i + 1)^{th}$ to $(\mathsf{tlen} + \mathsf{len})^{th}$ bits of $t||x$.
  - **Constr**: $\mathcal{A}$ submits a tuple (t^*, x^*, y). For computing $\sigma^*, z, \mathsf{sk}_1, \dots, \mathsf{sk}_{\mathsf{tlen}+\ell}$, the challenger traverses down the respective paths in the tree and computes the values for each node on the path based the same way (uniformly random and consistent values till level i anf PRG evaluations subsequently) as described earlier for **Eval** queries and sets the response accordingly.
  - **Challenge**: $\mathcal{A}$ submits a challenge pair $(\widetilde{t}, \widetilde{x})$ satisfying the admissibility criteria as defined in the security definition in Section 3.4. The challenger responds with the value obtained by traversing down the path corresponding to $\widetilde{t}||\widetilde{x}$ in the same way as described earlier for **Eval** queries.
- $\mathcal{A}$ outputs a guess $i' \in \{0, 1\}$, the experiment outputs b'.

Observe that hybrid experiment $\mathsf{H}^{\mathcal{A},0}$ is same as $\mathsf{Hyb}^{\mathcal{A},0}$ and $\mathsf{H}^{\mathcal{A},\mathsf{tlen}+\mathsf{len}}$ is same as $\mathsf{Hyb}^{\mathcal{A},1}$. Further, as $\mathcal{A}$'s advantage in distinguishing $\mathsf{H}^{\mathcal{A},0}$ and $\mathsf{H}^{\mathcal{A},1}$ is at least $\epsilon(1^\lambda, \mathsf{tlen}, \mathsf{len})$. Then, there must exist $i \in \{1, \ldots, \mathsf{tlen} + \mathsf{len}\}$ such that

$$|\Pr[\mathsf{H}^{\mathcal{A},i-1}(1^\lambda, \mathsf{tlen}, \mathsf{len}) = 1] - \Pr[\mathsf{H}^{\mathcal{A},i}(1^\lambda, \mathsf{tlen}, \mathsf{len}) = 1]| \geq \frac{\epsilon(1^\lambda, \mathsf{tlen}, \mathsf{len})}{\mathsf{tlen} + \mathsf{len}}.$$

Next, notice that the only difference between $\mathsf{H}^{\mathcal{A},i-1}$ and $\mathsf{H}^{\mathcal{A},i}$ is how the values for nodes at level i are generated. In $\mathsf{H}^{\mathcal{A},i-1}$, they are PRG evaluation on the value of their parent node. In $\mathsf{H}^{\mathcal{A},i}$, they are random values. The reduction $\mathcal{B}$ will try to plant the PRG challenges at level i as follows. The PRG challenger $\mathcal{C}$ chooses a random bit $b \xleftarrow{\$} \{0, 1\}$. If $b = 0$, $\mathcal{C}$ chooses polynomial number of PRG seeds s and sends PRG evaluations $G(s)$ to $\mathcal{B}$ as PRG challenges. If $b = 1$, $\mathcal{C}$ sends polynomial number of uniformly random values to $\mathcal{B}$ as PRG challenges. $\mathcal{B}$ chooses a random value $i \in \{1, \ldots, \mathsf{tlen} + \mathsf{len}\}$. Then, $\mathcal{B}$ interacts with $\mathcal{A}$ and for all of $\mathcal{A}$'s queries, $\mathcal{B}$ computes all node values for all levels except level i as described in $\mathsf{H}^{\mathcal{A},i-1}$. For computing node values at level i, $\mathcal{B}$ uses the PRG challenges it obtained. Finally, $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$ to $\mathcal{B}$. $\mathcal{B}$ outputs b' to $\mathcal{C}$.

Observe that in the above game, if $b = 0$, then, $\mathcal{A}$'s view corresponds to $\mathsf{H}^{\mathcal{A},i-1}$, and if $b = 1$, then, $\mathcal{A}$'s view corresponds to $\mathsf{H}^{\mathcal{A},i}$ Therefore, $\mathcal{B}$'s winning probability is as follows:

$$\begin{aligned}
\Pr[\mathcal{B} \text{ wins}] &= \frac{1}{2}\Pr[b' = 0|b = 0] + \frac{1}{2}\Pr[b' = 1|b = 1] \\
&= \frac{1}{2}\big(1 - \Pr[b' = 1|b = 0]\big) + \frac{1}{2}\Pr[b' = 1|b = 1] \\
&= \frac{1}{2} + \frac{1}{2}\big(\Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0]\big) \\
&= \frac{1}{2} + \frac{1}{2}(\Pr[\mathsf{H}^{\mathcal{A},i}(1^\lambda, \mathsf{tlen}, \mathsf{len}) = 1] - \\
&\Pr[b' = 1|b = 0] - \Pr[\mathsf{H}^{\mathcal{A},i-1}(1^\lambda, \mathsf{tlen}, \mathsf{len}) = 1]) \\
&\geq \frac{1}{2} + \frac{1}{2}\frac{\epsilon(1^\lambda, \mathsf{tlen}, \mathsf{len})}{\mathsf{tlen} + \mathsf{len}}.
\end{aligned}$$

Therefore, $\mathcal{B}$ can break the sub-exponetial security of PRGs. □

Lemma A.3. *Suppose that PRGs are sub-exponentially secure. Then, there exists a constant* $0 < \gamma < 1$ *such that for any probabilistic polynomial-time (p.p.t.)* admissible *adversary* $\mathcal{A}$*, for any* len *that is polynomially bounded in* λ*, there exists* λ_0 *such that for any* $\lambda > \lambda_0$*,*

$$|\Pr[\mathsf{Hyb}^{\mathcal{A},1}(1^\lambda, \mathsf{tlen}, \mathsf{len}) = 1] - \Pr[\mathsf{Hyb}^{\mathcal{A},2}(1^\lambda, \mathsf{tlen}, \mathsf{len}) = 1]| \leq 2^{-\lambda^\gamma}.$$

Proof. Similar to Lemma A.2. □

B Proof of Lemma 2.4

As mentioned in Section 2.6, to better understand our selective security notion for NIAR, it helps to think about an adaptive single-challenge counterpart which is very similar to the above selective single-challenge definition, except that the adversary need not commit to the challenge time step t^* and the challenge honest plaintexts ahead of time.

Definition B.1 (Adaptive single-challenge security for NIAR)**.** We say that a NIAR scheme satisfies adaptive single-challenge security (with receiver insider protection), iff for any non-uniform p.p.t. admissible adversary $\mathcal{A}$, $\mathcal{A}$'s views in the following experiments $\mathsf{AdSingleCh}^{0,\mathcal{A}}(1^\lambda, \mathsf{tlen}, \mathsf{len})$ and $\mathsf{AdSingleCh}^{1,\mathcal{A}}(1^\lambda, \mathsf{tlen}, \mathsf{len})$ are computationally indistinguishable.

Adaptive single-challenge experiment $\mathsf{AdSingleCh}^{b,\mathcal{A}}(1^\lambda, \mathsf{tlen}, \mathsf{len})$**.**

- $n, \mathcal{K}_S, \mathcal{K}_R, \pi^{(0)}, \pi^{(1)} \leftarrow \mathcal{A}(1^\lambda, \mathsf{tlen}, \mathsf{len})$;
- $(\{\mathsf{ek}_u\}_{u \in \mathcal{K}_S}, \{\mathsf{ek}_u^{(0)}\}_{u \in \mathcal{H}_S}, \{\mathsf{ek}_u^{(1)}\}_{u \in \mathcal{H}_S}, \{\mathsf{rk}_u\}_{u \in [n]}, \mathsf{tk})$
  $\leftarrow \mathbf{Setup}^*(1^\lambda, \mathsf{tlen}, \mathsf{len}, n, \perp, \pi^{(0)}, \pi^{(1)}, \mathcal{K}_S)$;
- $\perp \leftarrow \mathcal{A}(\mathsf{tk}, \{\mathsf{ek}_u\}_{u \in \mathcal{K}_S}, \{\mathsf{rk}_u\}_{u \in \mathcal{K}_R})$;
- For $t = 1, 2, \ldots$:
  - $(\{x_{u,t}^{(0)}\}_{u \in \mathcal{H}_S}, \{x_{u,t}^{(1)}\}_{u \in \mathcal{H}_S}, \delta_t) \leftarrow \mathcal{A}(\perp)$ where $\delta_t \in \{0, 1, \text{“}\texttt{challenge}\text{”}\}$;
  - if $\delta_t \in \{0,1\}$, then for $u \in \mathcal{H}_S$, let $\mathsf{CT}_{u,t} := \mathbf{Enc}(\mathsf{ek}_u^{(\delta_t)}, x_{u,t}^{(\delta_t)}, t)$;
  - else if $\delta_t = \text{“}\texttt{challenge}\text{”}$, then for $u \in \mathcal{H}_S$, let $\mathsf{CT}_{u,t} := \mathbf{Enc}(\mathsf{ek}_u^{(b)}, x_{u,t}^{(b)}, t)$;
  - $\perp \leftarrow \mathcal{A}(\{\mathsf{CT}_{u,t-1}\}_{u \in \mathcal{H}_S})$;

The adversary $\mathcal{A}$ is said to be admissible iff with probability 1, the following hold:

- There is a unique time step henceforth denoted t^* in which $\mathcal{A}$ sets δ_t to be “`challenge`”; and
- $\mathsf{Leak}^*(\pi^{(0)}, \mathcal{K}_S, \mathcal{K}_R, \{x_{u,t^*}^{(0)}\}_{u \in \mathcal{H}_S}) = \mathsf{Leak}^*(\pi^{(1)}, \mathcal{K}_S, \mathcal{K}_R, \{x_{u,t^*}^{(1)}\}_{u \in \mathcal{H}_S})$.

Clearly, Definition 2.3 is a selective relaxation of Definition B.1. The following lemma states that Definition B.1 implies the full security definition, i.e., Definition 2.1.

Lemma B.2 (Restatement of Lemma 2.4)**.** *A NIAR scheme that is secure by Definition B.1 is also secure by Definition 2.1.*

Proof. We can show that any NIAR scheme that is secure by Definition B.1 is also secure by Definition 2.1 through a straightforward hybrid argument. Imagine that the challenger always uses $\mathbf{Setup}^*$ to generate two sets of honest encryption keys and the same set of corrupt sender keys. We can now consider a sequence of $Q+1$ hybrids where Q is the total number of encryption queries. In the i-th hybrid where $i \in [Q+1]$, the challenger answers the first $i-1$ encryption queries using $b = 1$, and answers the remaining queries using $b = 0$. It is not hard to see that the first hybrid is identically distributed as $\mathsf{NIAR\text{-}Expt}^0$, and the last hybrid is identically distributed as $\mathsf{NIAR\text{-}Expt}^1$. Any pair of adjacent hybrids are computationally indistinguishable through a straightforward reduction to the adaptive single-challenge security. □