Non-Interactive Anonymous Router with Quasi-Linear Router Computation

Rex Fernando[1], Elaine Shi [*1], Pratik Soni [†2], Nikhil Vanjani[1], and Brent Waters [‡3]

[1]Carnegie Mellon University
[2]University of Utah
[3]University of Texas at Austin and NTT Research

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. Shi and Wu's construction suffers from two drawbacks: 1) the router takes time *quadratic* in the number of senders to obliviously route their messages; and 2) the scheme is proven secure only in the presence of static corruptions.

In this work, we show how to construct a non-interactive anonymous router scheme with *sub-quadratic* router computation, and achieving security in the presence of *adaptive corruptions*. To get this result, we assume the existence of indistinguishability obfuscation and one-way functions. Our final result is obtained through a sequence of stepping stones. First, we show how to achieve the desired efficiency, but with security under static corruption and in a selective, single-challenge setting. Then, we go through a sequence of upgrades which eventually get us the final result. We devise various new techniques along the way which lead to some additional results. In particular, our techniques for reasoning about a *network of obfuscated programs* may be of independent interest.

[*]Supported by a DARPA SIEVE grant, a Packard Fellowship, a grant from Algorand Foundation, NSF awards under the grant numbers 2128519 and 2044679, and a grant from ONR under the award number N000142212064.

[†]Work was done partially when the author was visiting Carnegie Mellon University.

[‡]Supported by NSF CNS-1908611, CNS-2318701 and Simons Investigator award.

Contents

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, Cha88, Abe99, GRS99, SBS02, DMS04, ZZZR05, CGF10, BG12, CBM15, vdHLZZ15, TGL+17] – a key component of anonymous communication systems. These include mix-nets [Cha81, Abe99, BG12], the Dining Cryptographers' nets [Cha88, CGF10, APY20], onion routing [GRS99, DMS04, CL05, DS18], multi-party-computation-based approaches [HEK12, AKTZ17, SA19], multi-server PIR-write [OS97, GIKM00, CBM15], 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. However, Shi and Wu's scheme suffers from two main drawbacks.

- First, their token size and router computation per time step are both *quadratic* in the number of users n, that is, $O_\lambda(n^2)$. We also stress that the quadratic router computation drawback

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

pertains not only to the work of Shi and Wu [SW21]. As Gordon et al. [GKLX22] pointed out, even in classical, *interactive* anonymous routing constructions [Cha81, Cha88, HEK12, SA19], 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.

- Shi and Wu's construction is proven secure only under *static corruption*, i.e., the adversary must specify all corrupt senders and receivers upfront. This leaves open an interesting question whether we can construct a NIAR scheme that is secure against adaptive corruptions, i.e., when the adversary can dynamically decide which players to corrupt.

The status quo gives rise to the following natural questions:

1. *Can we have a NIAR scheme with* subquadratic *router computation?*
2. *Can we have a NIAR scheme secure against* adaptive *corruptions?*

1.1 Main Result

In this paper, we construct a new NIAR scheme that simultaneously answers both of the above questions affirmatively. In particular, our new NIAR construction achieves $\widetilde{O}_\lambda(n)$ router computation per time step where $\widetilde{O}_\lambda(\cdot)$ hides both $\mathsf{poly}(\lambda, \log n)$ factors; moreover, it achieves security in the presence of adaptive corruptions. In terms of assumptions, we need the existence of indistinguishability obfuscator (iO) [GGH+13, JLS21, GP20, WW20, BDGM20] and one-way functions.

Theorem 1.1 (Informal: NIAR with subquadratic router computation)**.** *Let λ be a security parameter. Let $n = n(\lambda)$ be the number of senders/receivers. Then, assuming the existence of indistinguishability obfuscator and one-way functions, there exists a NIAR scheme (in the receiver insider protection setting) that satisfies security under adaptive corruptions. Further, the asymptotical performance bounds are as follows:*

1. *the token size and router computation per time step is $\widetilde{O}_\lambda(n)$;*
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 $O_\lambda(1)$.*

Technical highlights. The above result is obtained through a sequence of stepping stones.

- *Techniques for reasoning about a network of obfuscated programs.* First, we show how to achieve the desired efficiency, but under a relaxed notion of security, that is, assuming static corruptions and a selective, single-challenge setting. To achieve this, we use a gate-by-gate obfuscation approach. Specifically, we break up one big circuit into a network of smaller circuits to obfuscate, through the use of a quasilinear-sized routing network. In this network, each smaller circuit is of size polylogarithmic in the number of senders, thus helping us meet our efficiency goals even after obfuscating each of the smaller circuits. We also devise new techniques for reasoning about a network of obfuscated programs. Specifically, we propose a new notion of a Somewhere Statistically Unforgeable (SSU) signature which may be of independent interest, and we show how to construct SSU signatures from either iO + one-way functions, or from fully homomorphic encryption.

- *New techniques for upgrading from selective and static security to fully adaptive security.* Next, we want to remove the static corruption and selective-single-challenge restrictions. What is interesting is that the *standard complexity leveraging techniques completely fail* in our context due to our efficiency requirements. Therefore, we devise various new techniques for upgrading the security of the scheme, which eventually gets us the final result. An important consequence of our techniques is that we only incur a *polynomial security loss* when performing the upgrades. A key insight in our upgrade is to consider the following single-inversion restriction on the adversary: it must submit two permutations seperated by a single inversion in the two worlds, i.e., the two permutations are almost identical except for swapping the destinations of a pair of senders. We prove that security w.r.t. a single inversion is in fact equivalent to security w.r.t two arbitrary permutations.

  Along the way, we also explore the relationship of different definitions of NIAR security, and get several additional results (see Section 1.2) which may be of independent interest.

1.2 Additional Results

Impossibility of simulation security for adaptive corruptions. Shi and Wu [SW21] showed that assuming static corruption, indistinguishability-based security is equivalent to simulation-based security for NIAR. We revisit the two definitional approaches in the context of adaptive corruption. Somewhat surprisingly, we show that indistinguishability-based security and simulation-based security are not equivalent in the context of adaptive corruption. In our paper, we focus on achieving indistinguishability-based security under adaptive corruptions, since we prove that the simulation-based notion is impossible for adaptive corruptions. However, our construction does satisfy simulation-based security under static corruptions due to the equivalence of the two notions under static corruptions.

Theorem 1.2 (Informal: Impossibility of simulation security for adaptive corruptions)**.** *There does not exist a NIAR scheme (in the receiver insider protection setting) that achieves simulation-based security under adaptive corruptions (even with subexponential security assumptions).*

Adaptively secure NIAR with $O(n^2)$ router computation from standard assumptions. Our techniques for upgrading from selective/static to adaptive security can be of independent interest. For example, we can apply the same upgrade techniques to the previous NIAR scheme by Shi and Wu [SW21], which gives an adaptively secure NIAR scheme with $O_\lambda(n^2)$ router computation from standard assumptions.

Corollary 1.3 (Informal: quadratic computation NIAR scheme assuming bilinear maps)**.** *Assume standard bilinear map assumptions. There exists a NIAR scheme (in the receiver insider protection setting) that satisfies security under adaptive corruptions, and the asymptotical performance bounds are as follows:*

1. *the token size and router computation per time step is $O_\lambda(n^2)$;*
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 $O_\lambda(1)$.*

Static-to-adaptive-corruption compiler for other settings. In Appendices H and I, we show that our static-to-adaptive-corruption compiler also works for non-interactive differentially

anonymous router schemes as introduced by Bünz et al. [BHMS21], and NIAR schemes with sender insider protection as introduced by Bunn et al. [BKO22].

1.3 Related Work and Open Questions

Techniques for reasoning about a network of obfuscated programs. To the best of our knowledge, the only other works that used the gate-by-gate obfuscation technique are the Jain and Jin [JJ22] and Canetti et al. [CLTV15]. However, our techniques are fundamentally different in nature from the previous works. With Jain and Jin's techniques, the evaluator will need to spend $\mathsf{poly}(n)$ time to evaluate each obfuscated gate whereas for our construction, each obfuscated gate takes only $\mathsf{poly}(\lambda, \log n)$ time to evaluate, which is important for our efficiency claims. Our network of iOs idea also differs fundamentally from Canetti et al. [CLTV15], which builds leveled fully-homomorphic encryption scheme from iO. In our setting, there are *multiple* encrypters some of whom may be *corrupt*, whereas in the setting of Canetti et al. [CLTV15], there is a *single* encrypter who is assumed to be *honest* — so their setting is a lot easier.

Another line of works [CHJV14, KLW15, BGL+15, AJS17, JJ22] constructs indistinguishability obfuscation for Turing machines and RAM programs. A natural question is whether obfuscating the routing network modelled as a Turing machine or RAM program can result in the required sub-quadratic routing efficiency. Unfortunately, prior approaches [CHJV14, KLW15, BGL+15, AJS17, JJ22] suffer from evaluation time that is polynomial in the input length — in the case of the routing network, it would result in $\mathsf{poly}(n)$ runtime. Therefore, we cannot directly use existing iO for Turing machines or RAMs as a blackbox to achieve the desired efficiency. This is also another way to see why our results are non-trivial.

Additional related work. The recent work of Bünz, Hu, Matsuo and Shi [BHMS21] made an 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.

Comparison with concurrent work. We stress that in this paper, we focus on constructing a NIAR scheme whose security is sufficient for instantiating a non-interactive anonymous shuffler. As mentioned earlier, the shuffler application is important in the context of distributed differentially private mechanisms in the so-called "shuffle model". For this application to work, we need the NIAR scheme to satisfy a notion of security called *receiver-insider protection*, that is, corrupt receivers (possibly colluding with the router and some corrupt senders) should not learn which honest senders have sent the message.

In comparison, the elegant concurrent work by Bunn, Kushilevitz, and Ostrovsky [BKO22] solves the *dual* problem as ours. Their syntax is the same as ours, but their security guarantees are for the *sender insider protection* setting, and are not sufficient for instantiating the shuffler application. In particular, in the sender insider protection setting, corrupt senders (possibly colluding with

the router and some corrupt receivers) should not learn which honest receivers are receiving their messages.

From a technical standpoint, *sender insider protection* is akin to Private Information Retrieval (PIR) [CGKS95, CG97]. In fact, if we allow quadratic router computation, a NIAR scheme with a sender insider protection is implied by PIR which is known from standard, polynomial-strength assumptions. By contrast, PIR does not directly lead to NIAR with *receiver*-insider security (even if router computation efficiency is a non-concern). In fact, NIAR with receiver insider protection is technically akin to *multi-client functional encryption (MCFE) with function-hiding security*. Technically, a NIAR scheme with receiver-insider protection implies a function-hiding MCFE for the selection functionality with the bounded, upfront key queries[2]. So far, the only known way to achieve receiver-insider security (i.e., the work by Shi and Wu [SW21]) also uses function-hiding, multi-client functional encryption as a building block. For this reason, receiver insider protection is technically more challenging based on the existing knowledge and techniques.

Bunn et al. [BKO22] did not discuss the issue of adaptive corruption in the context of their primitive. Interestingly, our work's static-to-adaptive compiler can also be applied to their sender insider protection setting.

Finally, from a technical perspective, Bunn et al.'s main idea is to use a rate-1 PIR scheme where they reuse the clients' PIR queries at the router over multiple sessions. To cut the router computation to quasi-linear, they also rely an oblivious routing network. In their paper, they construct and analyze a new oblivious routing network for this purpose. Alternatively, they can also directly use the same oblivious routing network that we use in our paper, which is directly borrowed from the earlier algorithms literature [ACN+20, RS21].

Open questions. Our feasibility results naturally raise several open questions for future work. Can we achieve subquadratic router computation from standard assumptions without using indistinguishability obfuscation? Can we construct a scheme with good concrete performance? Can we strengthen the security of the scheme to get full insider protection (as defined by [SW21]) from standard assumptions?

2 Technical Roadmap

To get the above result, we had to go through multiple intermediate steps, where we first construct schemes with relaxed security notions and then gradually lift them to full security. In this process, we develop several interesting new techniques and building blocks that may be of independent interest. At a very high level, our blueprint and techniques are summarized below:

2.1 Single Selective Challenge and Static Corruptions

Our first step is to construct a NIAR scheme with quasilinear router computation, but we relax the definitions and only require security when the adversary has to upfront commit to 1) all corrupt senders and receivers and 2) a single challenge time step along with the corresponding challenge plaintext vectors. In this step, we encounter multiple challenges.

Definitional challenge for single, selective security. First, it turns out that even defining a

[2]In this sense, our adaptive corruption result is also interesting in the context of function-hiding MCFE since how to get function-hiding MCFE under adaptive corruption was not known earlier. The recent work of Shi and Vanjani [SV23] showed a function-hiding MCFE scheme for inner-product computation under static corruption, relying on standard bilinear group assumptions.

meaningful selective notion of security (in the static corruption setting) is non-trivial, because it is unclear how the non-challenge rounds should behave. This definition should satisfy two goals: first, the non-challenge rounds should contain no information about the permutation. This is because our techniques below crucially rely on this. Second, the definition should generalize naturally to full security. We discuss these issues in more detail in Appendix D.

Gate-by-gate obfuscation for efficiency. Next, we consider how to get a NIAR scheme with quasilinear router computation under the relaxed security. To start with, it is helpful to imagine an inefficient scheme where the router's token is an obfuscated circuit that encodes the entire permutation π as well as encryption and decryption keys. Now, upon receiving the n incoming ciphertexts, the obfuscated circuit decrypts the incoming ciphertexts, applies the permutation π, and reencrypts the outcomes using each receiver's respective key. The intuition if we treat the obfuscation as a black box which completely hides its internals, then it should hide everything about π and the honest parties' plaintexts beyond what the corrupted parties are allowed to decrypt.

There are two problems with this approach. First, the seminal work of [BGI+01] showed that it is impossible to achieve an "virtual black-box" (VBB) obfuscation scheme that perfectly hides its internals. Second, even forgetting about the security analysis, all known obfuscation schemes have large polynomial blowup in the input size; there is no obfuscation scheme that even comes close to a quadratic blowup, let alone subquadratic. This clearly does not meet our efficiency requirement.

Our idea to solve this efficiency problem is to break up one big circuit into a network of smaller circuits to obfuscate, through the use of a quasilinear-sized routing network. In this routing network, each gate has only polylogarithmically sized inputs and outputs, and there are $O(n)$ such gates. Now, if we obfuscate each gate separately and create a network of obfuscated gates, then the total size of all obfuscated gates would be quasilinear.

It turns out that this idea would only work if the underlying routing network has a special "obliviousness" property, that is, a corrupt sender cannot infer from its own route the destinations of honest senders (see Definition B.1). Fortunately, we were able to get such a routing network using known techniques from the oblivious sorting literature (although the notion of "obliviousness" there is of a different nature).

New techniques for reasoning about a network of obfuscated programs. We now turn to the challenges involved in reasoning about the security of "networked obfuscated programs". To solve these challenges, we develop techniques which we believe have potential to be useful in future applications. In particular, when the output of one obfuscated gate is fed into another, we want to ensure that the adversary does not tamper with the output in between. To achieve this, we would like to have each obfuscated gate authenticate its own outputs, and have the next obfuscated gate verify the authentication information before proceeding to the computation.

As mentioned before, it is well-known that VBB obfuscation is impossible, and it is only possible to achieve a much more restrictive notion called indistinguishability obfuscation (iO). iO achieves a much weaker notion of security: it only guarantees that obfuscations of two *functionally equivalent* programs are indistinguishable. As is evident from prior works, computationally secure primitives are generally incompatible with the functional equivalence requirements of iO. Therefore, we need to develop new iO-compatible techniques for authentication.

A new notion of somewhere-statistically-unforgeable signatures. To this end, one of our contributions is to introduce a new building block called a *Somewhere Statistically Unforgeable (SSU)* signature scheme. Informally, in an SSU signature scheme, there are three modes to sample the signing and verification keys: *normal mode*, *punctured mode*, and the *binding mode*.

- Normal mode: the signing and verification keys behave same as in standard digital signatures.
- Punctured mode: the signing key is *punctured* w.r.t. a set of points X but the verification key is normal. Intuitively, this means that *no* valid signature can be computed using the signing algorithm for points outside of the set X when using the *punctured* signing key.
- Binding mode: here, both the signing and verification keys are *binding* w.r.t. a sets of points X. Intuitively, this means that, with overwhelming probability, *no* valid signatures exist for points outside of the set X w.r.t. a randomly sampled *binding* verification key. In other words, this means that statistical unforgeability holds *somewhere* (points outside the set X) in the binding mode.

SSU signatures can be used to sign/verify tuples of the form (t, m), where t denotes a round and m denotes a message. For a fixed round t^* and message m^*, we specifically focus on a set X_{t^*,m^*} which contains pairs (t, m) as follows:

- For all $t \neq t^*$, $(t, m) \in X_{t^*,m^*}$ for all $m \in \{0,1\}^{\mathsf{len}}$.
- For $t = t^*$, there is a single pair $(t^*, m^*) \in X_{t^*,m^*}$, and for all $m' \neq m^*$, $(t^*, m') \notin X_{t^*,m^*}$.

Intuitively, we can use this restriction to restrict the behavior of the network of circuits during the challenge round t^*. Note that both the round t^* and the message m^* must be fixed when generating the keys of the signature scheme, hence (among other reasons) why the techniques here achieve a selective notion of security for the NIAR scheme.

For our network of iO proof to go through, we need the following important property from the SSU signature. We require that the distributions to be computationally indistinguishable:

$$\begin{pmatrix} \textit{punctured signing key,} \\ \textit{normal verification key} \end{pmatrix} \equiv_c \begin{pmatrix} \textit{binding signing key,} \\ \textit{binding verification key} \end{pmatrix}$$

This property is critical when we perform a layer-by-layer hybrid argument in our proofs.

We stress that for technical reasons explained below, this property is important for our "networked obfuscated programs" techniques to work. This property also differentiates our SSU signature scheme from previous known puncturable signature schemes [HIJ+17, BSW16, GWZ22]. The main difference from previous puncturable signature schemes is that we need the two verification keys to be indistinguishable even in the presence of some signing key, whereas the previous schemes required that the two verification keys be indistinguishable in absence of any signing key.

We show how to construct such a SSU signature scheme from puncturable PRFs, single-point binding (SPB) signatures, and single-point binding (SPB) hash functions [3]. In Section 2.4, we provide some intuition of how we constuct such SSU signatures. We know how to construct puncturable PRFs from one-way functions [GGM86, BW13, BGI14, KPTZ13], SPB signatures from one-way functions [GWZ22], and SPB hash function from indistinguishability obfuscation or leveled fully homomorphic encryption [GWZ22]. Plugging in these instantiations, we obtain the following theorem which may be of independent interest:

Theorem 2.1 (Informal: SSU signatures)**.** *Assuming the existence of one-way functions and indistinguishability obfuscation, or assuming leveled fully homomorphic encryption, there exists a somewhere statistically unforgeable signature scheme for the family of sets X_{t^*,m^*} defined above.*

[3] Informally speaking, SPB signatures have a special single-point binding property which states that it is possible to generate a special verification key w.r.t. a message m^* s.t. it only accepts a single signature for m^*. Similarly, SPB hash functions have a special single-point binding property which states that it is possible to generate a special hash key w.r.t. a message m^* s.t. no hash collisions exist on m^*.

Network of iOs: proof highlight. We sketch our proof outline, focusing on the part that makes use of the aforementioned property of our SSU signature. In our construction, a ciphertext encrypts the message as well as the route it should be sent along. Imprecisely speaking then, each gate does the following: decrypts the incoming ciphertexts and verifies the input message signature (to authenticate the message) and route signature (to authenticate the route); and if valid, it performs the routing, encrypts the output messages (along with the routing information), and uses an output signing key to sign them. Our proof goes through a sequence of hybrids sketched below[4].

- First, starting from the real-world experiment, through a sequence of hybrids, we hard-wire the route signatures on corrupt wires (which are shared across all time steps) — we defer the details of these hybrids to the subsequent technical sections so we can focus on the part of proof that uses aforementioned property of our SSU signatures.
- Next, through a layer-by-layer hybrid sequence, we want to switch to a world in which for challenge time step t^*, honest and filler wires' ciphertexts and signatures (for both messages and routes) are hard-wired in the obfuscated gate and the obfuscated gate only accepts an incoming ciphertext that matches the hiredwired one. Except for the honest-to-corrupt wires in the last layers which are hard-wired encryptions of the actual messages to be received by the corrupt receiver, for all other honest/filler wires, the hired-wired ciphertexts are encryptions of fillers. In this new world, the challenge ciphertexts for honest senders are also random encryptions of fillers; therefore, for the challenge time step t^*, the adversary's view contains no information about honest-to-honest and honest-to-corrupt routes, as well as honest-to-honest messages.

  As described below, this layer-by-layer hybrid is where we critically need the aforementioned security property from the SSU signatures.

  - Assuming that layer i's input verification key is already switched to binding, we can then switch layer i's output signing key to a punctured signing key by using iO security (since the binding verification key already ensure that the punctured messages will never pass through);
  - Next, we make the following replacement by relying on the security of the SSU signature scheme:

$$\begin{aligned} & (\text{punctured signing key: layer } i, && \text{normal verification key: layer } i+1) \\ \Longrightarrow\ & (\text{binding signing key: layer } i, && \text{binding verification key: layer } i+1) \end{aligned}$$

  - At this moment, by relying on iO security, we can hard-wire the ciphertexts and signatures for t^* on honest/filler wires, such that the obfuscated gate only accepts the input on the wire if it matches the hard-wired value.

2.2 Removing the Selective Challenge Restriction

Recall that the techniques above are able to achieve a limited notion of security, which we call "selective single-challenge" security. The selective notion requires that the adversary submit not just two permutations $\pi^{(0)}$ and $\pi^{(1)}$ upfront, but additionally commit to both a challenge round t^* and the set $\{x^{(0)}_{u,t^*}, x^{(1)}_{u,t^*}\}_{u\in\mathcal{H}_S}$ of challenge plaintexts to be used during round t^* for the honest senders $\mathcal{H}_S$, two for each honest sender. Recall that we use the SSU signature scheme above, and

[4]Our formal proof in the technical sections actually first proves single, selective-challenge, static security only for an adversary subject to the following restrictions: it must corrupt all receivers, and submit two permutations that differ by a single inversion. We prove that even this weaker version is sufficient for our upgrade all the way to full security under adaptive corruption.

we puncture the signing and verification keys at round t^* and at the target plaintexts, which we hardcode in the obfuscated gates during the inner hybrids. This is essentially why we need this data upfront.

Standard complexity leveraging fails. The standard tool to achieve such a transformation is complexity leveraging. Namely, to run the adaptive-query single-challenge with the selective scheme, we guess the values t^* and $\{x^{(0)}_{u,t^*}, x^{(1)}_{u,t^*}\}_{u\in\mathcal{H}_S}$ at the beginning of the experiment. This incurs a security loss proportional to 2^α, where α is the number of bits needed to represent t^* and $\{x^{(0)}_{u,t^*}, x^{(1)}_{u,t^*}\}_{u\in\mathcal{H}_S}$. We stress that due to our efficiency requirements, complexity leveraging fails completely even if we are willing to accept the (sub-)exponential loss in the security reduction. More specifically, α can be as large as $O(n)$, which means that the selective-secure NIAR scheme would have to be $2^{O(n)}$-secure for the reduction to be meaningful. Thus we must adopt a security parameter greater than n in all the underlying primitives, including the iO scheme, resulting in $\mathsf{poly}(n)$ cost and thus defeating our efficiency goals. This problem seems inherent with our techniques, because as explained above, we hardcode information about each $x^{(b)}_{u,t^*}$ for all honest u in the obfuscated gates.

Removing the selective challenge restriction through equivalence to single-inversion security. For removing the selective challenge restriction, our key insight is to define a single-inversion notion of NIAR security, and using single-inversion security as a stepping stone. In normal NIAR security (under static corruption), we want security to hold for two arbitrary admissible permutations. In single-inversion security, we consider two admissible permutations where only a pair of honest senders' destinations are swapped.

If we can prove equivalence to single-inversion security, then we can do the selective-query to adaptive-query upgrade for single-inversion security. In this case, a standard complexity leveraging argument has only polynomial security loss as explained below. Specifically, with single-inversion security, the reduction only needs to guess the challenge time step t^* and the challenge plaintexts of the two swapped honest users in the two worlds — without loss of generality, we can assume that each sender's plaintext is a single bit, since we can always get a multi-bit scheme by parallel composition of multiple single-bit schemes. Further, we assume that the reduction is given an upper bound on the p.p.t. adversary's running time. Therefore, the space of the guesses is polynomially bounded.

The remaining technicality is proving equivalence to single-inversion security. At first sight, it might be tempting to conclude that this is obvious, since given $\pi_{(0)}$ and $\pi_{(1)}$, we can always swap a pair of honest senders at a time to eventually transform $\pi_{(0)}$ to $\pi_{(1)}$. However, correctly implementing this idea is subtle. Specifically, we need any pair of adjacent hybrids to be not trivially distinguishable by the adversary, where the adversary is subject to the admissibility rules of the beginning and the end hybrids. We prove that given any beginning and end hybrids, we can indeed construct a sequence of intermediate hybrids, each time swapping a pair of honest senders' destinations and their messages, such that each pair of adjacent hybrids satisfy the aforementioned constraint.

2.3 Achieving Security for Adaptive Corruptions

So far, we have constructed a NIAR scheme that achieves security under static corruptions but adaptive queries. The last question remaining is how to upgrade the scheme to get security even under adaptive corruptions.

A first idea that comes to mind is to again attempt complexity leveraging. Again, unfortunately, due to our efficiency requirements, complexity leveraging completely does not work even if we

are willing to suffer from (sub-)exponential losses in the reduction. Suppose that the reduction guesses which set of players the adversary will corrupt. Since there are 2^n possible guesses, for the parameters to work in the complexity leveraging, we must adopt a security parameter that is greater than n in the underlying iO scheme, which results in at least $\mathsf{poly}(n)$ blowup.

A new compiler for upgrading to adaptive corruptions. Instead of complexity leveraging, we construct a new compiler that compiles a NIAR scheme secure under static corruption to a NIAR scheme secure under adaptive corruption, with only polynomial loss in the security reduction.

The compiler is very simple: each sender will first encrypt their plaintext using a PRF that is secure against selective opening (which is implied by standard PRF security as shown by Abraham et al. [ACD+19]), before encrypting it using the NIAR scheme that is secure under static corruption. For proving that this construction secure against adaptive corruptions, we will go through the following key steps:

- First, suppose we want to prove single-inversion security when all the receivers are corrupt. Now, when the reduction receives the two permutations $\pi_{(0)}$ and $\pi_{(1)}$, it may assume that only the inverted pair of senders are honest. Therefore, in this case, security under adaptive corruption is the same as security under static corruption.
- Next, still assuming that all receivers are corrupt, we want to prove security under adaptive corruption for any two arbitrary permutations. For this step, we need to prove the equivalence of security under two arbitrary permutations and single-inversion security, but now for the scenario when the senders can be adaptively corrupted. The technicalities in this proof are similar to the counterpart for the static corruption case; however, the argument becomes somewhat more involved now that the senders can be adaptively corrupt.
- Finally, we show how to remove the assumption where the receivers must be all corrupted upfront, and allow the adversary to adaptively corrupt the receivers. This step will rely on the selective-opening security of the PRF which is implied by standard PRF security [ACD+19].

2.4 SSU Signature Construction

In this section, we give an informal overview of our SSU signature construction. Our scheme is inspired by the well-known Merkle signature scheme [Mer79] which can upgrade a one-time signature scheme such as Lamport signatures [Lam79] to a multi-use signature. Recall that the Merkle signature construction works as follows:

- There is a signing key and verification key pair (for a one-time signature scheme) at every node u in the tree denoted $(\mathsf{sk}_u, \mathsf{vk}_u)$, and the pair $(\mathsf{sk}_u, \mathsf{vk}_u)$ are sampled using $\mathsf{PRF}_k(u)$. The final verification key is $\mathsf{vk}_{\text{root}}$, and the secret signing key is $(k, \mathsf{sk}_{\text{root}})$.
- To sign a new message m, pick the next unused leaf, and consider the path from the root to the leaf. Let $\{\mathsf{vk}_0 = \mathsf{vk}_{\text{root}}, \mathsf{vk}_1, \mathsf{vk}_2, \ldots, \mathsf{vk}_d\}$ be the verification keys corresponding to nodes on the path from the root to the selected leaf, and let $\{\mathsf{vk}'_1, \ldots, \mathsf{vk}'_d\}$ denote the verification keys for the siblings of these nodes. The signer uses sk_0 to sign $H(\mathsf{vk}_1, \mathsf{vk}'_1)$, uses sk_1 to sign $H(\mathsf{vk}_2, \mathsf{vk}'_2)$, and so on where we use $H(\cdot)$ to denote a hash function. Finally, use sk_d to sign hash of the actual message $H(m)$. The resulting signature contains all $d+1$ signatures as well as $\{\mathsf{vk}_1, \mathsf{vk}'_1, \ldots, \mathsf{vk}_d, \mathsf{vk}'_d\}$.
- Verification is done in the most natural manner.

Recall that we want to construct an SSU signature scheme for the set X_{t^*,m^*}, such that in the binding mode, the only message that can be signed for the time step t^* is m^*. We will modify the Merkle signature scheme as follows:

- Imagine that each leaf of the tree corresponds to some time step t. To sign a message x under the time step t, the signer will use the leaf node corresponding to t.
- We use a punctured PRF instead of a standard PRF to generate the $(\mathsf{vk}_u, \mathsf{sk}_u)$ pair for every tree node u.
- Instead of an arbitrary one-time signature scheme, we want to use a one-time signature scheme with a single-point binding (SPB) property, that is, there is a binding setup which takes a message m^* as input, and generates a verification key vk^* such that the only message that can pass verification is m^*; and moreover, a computationally bounded adversary cannot tell that vk^* is generated using the binding mode.
- Instead of using a normal hash function $H(\cdot)$, we will use a single-point binding (SPB) hash function. We will create one SPB hash instance per level of the tree, and include the hash keys in both the signing and verification keys. An SPB hash function has a binding setup mode that takes m^* as input and generates a binding hash key hk^* such that m^* does not have any collision; moreover, a computationally bounded adversary cannot tell that hk^* is a binding key.

Punctured key. To puncture the signing key such that one can sign only x^* at t^*, puncture the PRF key such that one is unable to compute the signing and verification key pairs on the path from the root to the leaf t^*. Additionally, we can use the unpunctured key to pre-sign the message m^* and t^* and include this signature in the punctured signing key.

Binding key. For the binding-mode setup, we want to generate a binding signing key and a binding verification key such that for t^*, only the message x^* has a unique valid signature. The binding mode also punctures the PRF key in the same way as the punctured mode. However, for the path from the root to the leaf at t^*, we no longer generate the $(\mathsf{sk}_u, \mathsf{vk}_u)$ honestly by using the unpunctured PRF key. Instead, we will call the binding setup of the SPB signatures and SPB hashes. Specifically, on the path from the root to the leaf t^*, we will run the binding setup algorithms of the SPB signature scheme such that at the leaf t^*, we can only sign hash of $t^*||x^*$; and at any non-leaf node on the path, we can only sign a unique hash (of the two children's verification keys). Further, we run the binding setup algorithms of the SPB hash functions, such that at level i of the tree, the pair $(\mathsf{vk}_i, \mathsf{vk}'_i)$ to be hashed has no collisions, where $(\mathsf{vk}_i, \mathsf{vk}'_i)$ are verification keys corresponding to the level-i node on the path to leaf t^* (recall that vk_i is generated using the binding mode of the SPB signature), and its sibling. After we generate all these keys, we again pre-sign (t^*, x^*) using these keys.

As a result, the binding verification key is $\mathsf{vk}_{\text{root}}$ and the hash keys which are generated using the binding mode of the SPB signature and hash schemes; and the binding signing key is the punctured PRF key, as well as the pre-signed signature for (t^*, m^*), and the binding hash keys.

Formal description and proofs. We defer the formal description of the SSU signature and its proofs to Section 5 and Appendix C.

Organization of rest of the paper. In Section 3, we define NIAR, and in Section 4 and Appendix B, we present preliminaries. In Section 5, we define SSU signatures and in Appendix C, we present a construction along with proofs of correctness and security. In Section 6, we construct a NIAR scheme secure against a static and all-receiver-corrupting adversary and present the security proof

in Appendices D and E. In Appendix F, we present a compiler that transforms the above NIAR scheme to one with full security, i.e., removing the static and all-receiver-corrupting restrictions on the adversary. In Appendices H and I, we show that this compiler also works for differentially anonymous and sender insider protection settings. In Appendix G, we present the impossibility of NIAR simulation security for adaptive corruptions.

3 Definitions for NIAR

In this section, we define the syntax and security for NIAR, focusing on the strongest definition of full security against adaptive corruptions.

3.1 Syntax

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

- $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{len}, n, \pi)$: the trusted **Setup** algorithm takes the security parameter 1^λ, 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 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 pk and 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 message x.

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

3.2 NIAR Full Security

In this section, we present a security notion for NIAR against a very strong adversary. In particular, we allow such an adversary to (a) adaptively corrupt the set of senders and receivers, and (b) adaptively ask for encryptions of chosen plaintext under the senders' keys that are not yet corrupted. Our security definition is a strict generalization of the "receiver-insider corruption" notion introduced by Shi and Wu [SW21] which captured only static corruptions of users.

We formalize our definition via the experiment $\mathsf{NIARFull}^{b,\mathcal{A}}$ which is parametrized by some challenge bit b and a ***stateful*** non-uniform p.p.t. adversary $\mathcal{A}$. At the beginning of the experiment, adversary $\mathcal{A}$ submits two challenge permutations $\pi^{(0)}$ and $\pi^{(1)}$ over $[n]$ for its choice of n. At any

time in the experiment, the adversary can choose to corrupt a sender or receiver, and it will receive the secret key for the newly corrupted player. The adversary receives tk, and then in each time step, it can submit two plaintext vectors $\{x_{u,t}^{(0)}, x_{u,t}^{(1)}\}_{u \in \mathcal{H}_S}$ for the set of currently honest senders $\mathcal{H}_S$. The challenger will encrypt the plaintexts indexed by $b \in \{0,1\}$, and at the end of the experiment, the adversary's job is to distinguish which world b the challenger is in. The adversary must be subject to a set of admissibility rules such that it cannot trivially distinguish which world it is in.

More formally, our full NIAR security game is defined as follows, where $\mathsf{Cor}(\cdot)$ is the following oracle: upon receiving a sender or receiver identity,

- return its corresponding secret key;
- in case the newly corrupted player is a sender, additionally return all the historical random coins consumed by the **Enc** algorithm during the previous time steps;
- update the honest sender set $\mathcal{H}_S$ and honest receiver set $\mathcal{H}_R$ accordingly.

NIAR full security experiment $\mathsf{NIARFull}^{b,\mathcal{A}}(1^\lambda)$**.**

1. $(n, \mathsf{len}, \pi^{(0)}, \pi^{(1)}) \leftarrow \mathcal{A}(1^\lambda)$.
2. $\mathcal{H}_S = [n]$, $\mathcal{H}_R = [n]$.
3. $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{len}, n, \pi^{(b)})$.
4. For $t = 1, 2, \ldots$:
   - if $t = 1$: $\{x_{u,t}^{(0)}, x_{u,t}^{(1)}\}_{u\in\mathcal{H}_S} \leftarrow \mathcal{A}^{\mathsf{Cor}(\cdot)}(\mathsf{tk})$;
     else $\{x_{u,t}^{(0)}, x_{u,t}^{(1)}\}_{u\in\mathcal{H}_S} \leftarrow \mathcal{A}^{\mathsf{Cor}(\cdot)}(\{\mathsf{CT}_{u,t-1}\}_{u\in\mathcal{H}_S})$.
   - for all $u \in \mathcal{H}_S$, $\mathsf{CT}_{u,t} \leftarrow \mathbf{Enc}(\mathsf{ek}_u, x_{u,t}^{(b)}, t)$.
5. At any time, $\mathcal{A}$ may halt and output an arbitrary function of its view. The experiment then also halts and returns the output of $\mathcal{A}$.

In the above definition, if the adversary wants to specify an initially corrupt set, it can simply make calls to the corruption oracle $\mathsf{Cor}(\cdot)$ at the beginning of $t = 1$. Therefore, without loss of generality, we may assume that the initially corrupt set before the challenger calls **Setup** is empty.

Admissibility. We state some admissibility rules on the adversary to make sure that the adversary cannot trivially distinguish whether it is in world $b = 0$ or $b = 1$. Our admissibility rule corresponds to the "receiver-insider protection" version of Shi and Wu [SW21], which is sufficient for building a non-interactive anonymous shuffler. Basically, we assume that senders know their receivers but receivers do not know their senders. Therefore, if the adversary corrupts some senders, the adversary will know the corrupt senders' receivers. We remark that Shi and Wu [SW21] additionally described a "full insider protection" notion where it is assumed that neither senders nor receivers know who they are paired with. Their "full insider protection" construction requires polynomial in n evaluation time and uses indistinguishable obfuscation and bilinear group assumptions [SW21]. It remains an open question how to reduce the evaluation time for the "full insider protection" version.

Henceforth, if a player remains honest at the end of the execution, we say that it is *eventually honest*; otherwise we say that it is *eventually corrupt*. We say that $\mathcal{A}$ is *admissible* iff with probability 1, the following holds where $\mathcal{H}_S$ and $\mathcal{H}_R$ refer to the eventually honest sender and receiver set, and define $\mathcal{K}_R = [n] \setminus \mathcal{H}_R, \mathcal{K}_S = [n] \setminus \mathcal{H}_S$ to be the eventually corrupt sender and receiver sets:

1. For all eventually corrupt senders $u \in \mathcal{K}_S$, $\pi^{(0)}(u) = \pi^{(1)}(u)$.
2. For any eventually corrupt sender $u \in \mathcal{K}_S$, for any t in which u was not corrupt yet, $x^{(0)}_{u,t} = x^{(1)}_{u,t}$. In other words, here we require that in the two alternate worlds $b = 0$ or $b = 1$, every eventually corrupt sender should be sending the same message in all rounds before it was corrupted.
3. For all rounds t, and for any $v \in \mathcal{K}_R \cap \pi^{(0)}(\mathcal{H}_S) = \mathcal{K}_R \cap \pi^{(1)}(\mathcal{H}_S)$, $x^{(0)}_{u_0,t} = x^{(1)}_{u_1,t}$ where for $b \in \{0,1\}$, $u_b := (\pi^{(b)})^{-1}(v)$. In other words, here we require that in the two alternate worlds $b = 0$ or 1, every eventually corrupt receiver receiving from an eventually honest sender must receive the same message in all rounds.

Definition 3.1 (NIAR full security)**.** We say that a NIAR scheme is $\mathsf{fully\ secure}$ iff for any non-uniform p.p.t. *admissible* $\mathcal{A}$, its views in the two experiments $\mathsf{NIARFull}^{0,\mathcal{A}}(1^\lambda)$ and $\mathsf{NIARFull}^{1,\mathcal{A}}(1^\lambda)$ are computationally indistinguishable.

4 Preliminaries

Whenever we refer to an adversary in the paper henceforth, we implicitly mean it to be a *non-uniform* adversary. We discuss the notations next and defer the additional preliminaries to Appendix B.

4.1 Notations

We say that a function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is negligible, if for every constant $c > 0$ and for all sufficiently large $\lambda \in \mathbb{N}$ we have $\mathsf{negl}(\lambda) < \lambda^{-c}$. Two distribution ensembles $\{X^\lambda_0\}_\lambda$ and $\{X^\lambda_1\}_\lambda$ are computationally indistinguishable if for every p.p.t. adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[x \leftarrow X^\lambda_0 : \mathcal{A}(x) = 0] - \Pr[x \leftarrow X^\lambda_1 : \mathcal{A}(x) = 0]| \leq \mathsf{negl}(\lambda)$. 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 $(y_1, \ldots, y_n) \leftarrow f(t, x_1, \ldots, x_n)$.

5 Somewhere Statistically Unforgeable (SSU) Signatures

In this section, we define SSU signatures and provide an informal construction.

5.1 Definition

We consider an SSU signature scheme where the signing and verification algorithms both take a counter t (i.e., time step) in addition to the message x to be signed. We refer to t as the *round*. Specifically, an SSU signature scheme contains the following algorithms:

- $(\mathsf{sk}, \mathsf{vk}, \mathsf{pp}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len})$: takes as input the security parameter 1^λ, the length of the round $\mathsf{tlen} \geq 0$, the length of the messages to be signed $\mathsf{len} > 0$, and outputs a signing key sk, a verification key vk, and a public parameter pp.
- $\sigma \leftarrow \mathbf{Sign}(\mathsf{pp}, \mathsf{sk}, t, x)$: a *deterministic* algorithm that takes as input a public paramter pp, a singing key sk, along with a round $t \in \{0,1\}^{\mathsf{tlen}}$ ($t = \perp$ in case $\mathsf{tlen} = 0$) and a message $x \in \{0,1\}^{\mathsf{len}}$ and outputs a signature σ for x w.r.t. t.
- (0 or 1) $\leftarrow \mathbf{Vf}(\mathsf{pp}, \mathsf{vk}, t, x, \sigma)$: takes as input a public paramter pp, a verification key vk, a round t, a message x, and a signature σ, and outputs either 1 for accept or 0 for reject.

- $(\mathsf{sk}, \widetilde{\mathsf{sk}}, \mathsf{vk}, \mathsf{pp}) \leftarrow \mathbf{PuncturedSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*)$: 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 $t^* \in \{0,1\}^{\mathsf{tlen}}$ ($t^* = \perp$ in case $\mathsf{tlen} = 0$) and a message $x^* \in \{0,1\}^{\mathsf{len}}$, and outputs a signing key sk, a punctured signing key $\widetilde{\mathsf{sk}}$, a verification key vk, and a public paramter pp.
- $(\mathsf{sk}^*, \mathsf{vk}^*, \mathsf{pp}^*) \leftarrow \mathbf{BindingSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*)$: 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 $t^* \in \{0,1\}^{\mathsf{tlen}}$ ($t^* = \perp$ in case $\mathsf{tlen} = 0$) and message $x^* \in \{0,1\}^{\mathsf{len}}$, and outputs a binding signing key sk^*, a binding verification key vk^*, and a binding public paramter pp^*.
- $\sigma \leftarrow \mathbf{PSign}(\mathsf{pp}, \widetilde{\mathsf{sk}}, t, x)$: a *deterministic* algorithm that takes as input a public paramter pp, a punctured signing key $\widetilde{\mathsf{sk}}$ generated by **PuncturedSetup**, a round t and a message x, and outputs a signature σ for x w.r.t. t.

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

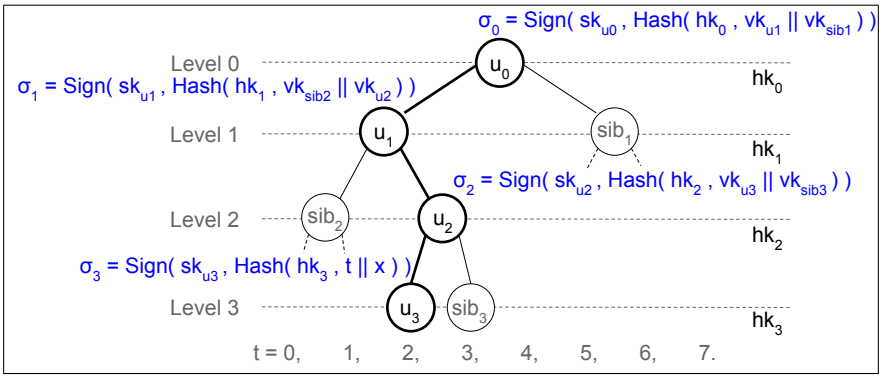
- For all $\lambda, \mathsf{len}, \mathsf{tlen} \in \mathbb{N}$, $t \in \{0,1\}^{\mathsf{tlen}}, x \in \{0,1\}^{\mathsf{len}}$,
$$\Pr\left[\begin{array}{r}(\mathsf{sk}, \mathsf{vk}, \mathsf{pp}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len}) \\ \sigma \leftarrow \mathbf{Sign}(\mathsf{pp}, \mathsf{sk}, t, x)\end{array} : \mathbf{Vf}(\mathsf{pp}, \mathsf{vk}, t, x, \sigma) = 1\right] = 1.$$
- For all $\lambda, \mathsf{len}, \mathsf{tlen} \in \mathbb{N}$, $t^*, t \in \{0,1\}^{\mathsf{tlen}}$, $x^*, x \in \{0,1\}^{\mathsf{len}}$ such that it is *not* the case that $t = t^*$ and $x \neq x^*$,
$$\Pr\left[\begin{array}{r}(\mathsf{sk}, \widetilde{\mathsf{sk}}, \mathsf{vk}, \mathsf{pp}) \leftarrow \mathbf{PuncturedSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*) : \\ \mathbf{Sign}(\mathsf{pp}, \mathsf{sk}, t, x) = \mathbf{PSign}(\mathsf{pp}, \widetilde{\mathsf{sk}}, t, x)\end{array}\right] = 1.$$

Definition 5.1 (Security for SSU Signatures)**.** An SSU signature is said to be secure if it has the following properties:

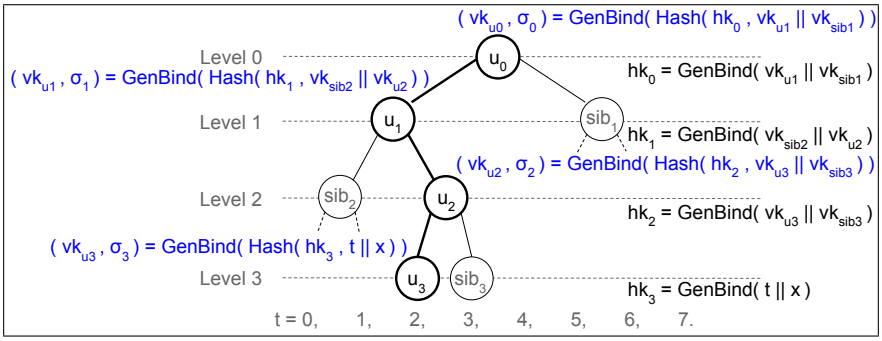
- **Identical distribution of normal keys output by Setup and PuncturedSetup.** For any λ, $\mathsf{len}, \mathsf{tlen} \in \mathbb{N}$, any $t^* \in \{0,1\}^{\mathsf{tlen}}$, any $x^* \in \{0,1\}^{\mathsf{len}}$, we have the following where $\equiv$ denotes identical distribution:
$$\begin{aligned}&\{(\mathsf{sk}, \mathsf{vk}, \mathsf{pp}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len}) : \text{ output } (\mathsf{sk}, \mathsf{vk}, \mathsf{pp})\} \\ \equiv &\{(\mathsf{sk}, \widetilde{\mathsf{sk}}, \mathsf{vk}, \mathsf{pp}) \leftarrow \mathbf{PuncturedSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*) : \text{ output } (\mathsf{sk}, \mathsf{vk}, \mathsf{pp})\}\end{aligned}$$
- **Indistinguishability of punctured and binding setups.** For any len and tlen that are polynomially bounded by λ, any $t^* \in \{0,1\}^{\mathsf{tlen}}$, any $x^* \in \{0,1\}^{\mathsf{len}}$, we have the following where $\approx$ denotes computational indistinguishability:
$$\begin{aligned}&\{(\mathsf{sk}, \widetilde{\mathsf{sk}}, \mathsf{vk}, \mathsf{pp}) \leftarrow \mathbf{PuncturedSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*) : \text{ output } (\widetilde{\mathsf{sk}}, \mathsf{vk}, \mathsf{pp})\} \\ \approx &\{(\mathsf{sk}^*, \mathsf{vk}^*, \mathsf{pp}^*) \leftarrow \mathbf{BindingSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*) : \text{ output } (\mathsf{sk}^*, \mathsf{vk}^*, \mathsf{pp}^*)\}\end{aligned}$$
- **Statistical unforgeability at (t^*, x^*).** For any $\mathsf{len}, \mathsf{tlen}$ that are polynomially bounded in λ, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for any $t^* \in \{0,1\}^{\mathsf{tlen}}$, $x^* \in \{0,1\}^{\mathsf{len}}$, for any λ,
$$\Pr\left[\begin{array}{r}(\mathsf{sk}^*, \mathsf{vk}^*, \mathsf{pp}^*) \leftarrow \mathbf{BindingSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*) : \\ \exists\ (\sigma, x) \text{ s.t. } x \neq x^* \ \wedge\ \mathbf{Vf}(\mathsf{pp}^*, \mathsf{vk}^*, t^*, x, \sigma) = 1\end{array}\right] \leq \mathsf{negl}(\lambda),$$
$$\Pr\left[\begin{array}{r}(\mathsf{sk}^*, \mathsf{vk}^*, \mathsf{pp}^*) \leftarrow \mathbf{BindingSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*) : \\ \exists\ \sigma \neq \mathbf{PSign}(\mathsf{pp}^*, \mathsf{sk}^*, t^*, x^*) \text{ s.t. } \mathbf{Vf}(\mathsf{pp}^*, \mathsf{vk}^*, t^*, x^*, \sigma) = 1\end{array}\right] \leq \mathsf{negl}(\lambda).$$

5.2 SSU Signatures: Informal Construction

Let $\Sigma = (\Sigma.\mathbf{Gen}, \Sigma.\mathbf{Sign}, \Sigma.\mathbf{Vf}, \Sigma.\mathbf{GenBind})$ be a single-point binding (SPB) signature scheme. Let $\mathsf{H} = (\mathsf{H}.\mathbf{Gen}, \mathsf{H}.\mathbf{Hash}, \mathsf{H}.\mathbf{GenBind})$ be a single-point binding (SPB) hash function. Let PPRF be a puncturable PRF. The SSU signature scheme is based on a binary tree of SPB signatures intuitively described in Figures 1a and 1b. We present the formal construction in Appendix C.



(a) Sign and PuncturedSetup. For each node u_i, $(\mathsf{sk}_{\mathsf{u}_i}, \mathsf{vk}_{\mathsf{u}_i})$ is generated using $\Sigma.\mathbf{Gen}$ with $\mathsf{PPRF.Eval}(K, \mathsf{u}_i)$ as the randomness seed. A signature σ on message x for $t = 2$ consists of $\sigma := ((\sigma_0, \mathsf{vk}_{\mathsf{u}_1}, \mathsf{vk}_{\mathsf{sib}_1}), (\sigma_1, vk_{\mathsf{u}_2}, \mathsf{vk}_{\mathsf{sib}_2}), (\sigma_2, \mathsf{vk}_{\mathsf{u}_3}, \mathsf{vk}_{\mathsf{sib}_3}), \sigma_3)$. **PuncturedSetup** at the point (t, x) outputs a punctured key $\widetilde{\mathsf{sk}}$ that consists of the PPRF key punctured at the set $\{\mathsf{u}_0, \mathsf{u}_1, \mathsf{u}_2, \mathsf{u}_3\}$ and σ.



(b) BindingSetup. For nodes $\mathsf{u}_i \in \{\mathsf{u}_0, \mathsf{u}_1, \mathsf{u}_2, \mathsf{u}_3\}$, $(\mathsf{vk}_{\mathsf{u}_i}, \sigma_u)$ is generated using $\Sigma.\mathbf{GenBind}$ with $\mathsf{PPRF.Eval}(K, \mathsf{u}_i)$ as the randomness seed. **BindingSetup** at the point (t, x) outputs a binding key sk^* that consists of the PPRF key punctured at the set $\{\mathsf{u}_0, \mathsf{u}_1, \mathsf{u}_2, \mathsf{u}_3\}$ and a signature σ on message x for $t = 2$, where $\sigma := ((\sigma_0, \mathsf{vk}_{\mathsf{u}_1}, \mathsf{vk}_{\mathsf{sib}_1}), (\sigma_1, vk_{\mathsf{u}_2}, \mathsf{vk}_{\mathsf{sib}_2}), (\sigma_2, \mathsf{vk}_{\mathsf{u}_3}, \mathsf{vk}_{\mathsf{sib}_3}), \sigma_3)$.

Figure 1: SSU Signatures informal construction

6 NIAR for a Static and All-Receiver-Corrupting Adversary

In this section, we first introduce a basic NIAR scheme which we prove secure under an adversary who is restricted to make all corruption queries upfront, and moreover, it must always corrupt all receivers — we call such an adversary a static, all-receiver-corrupting adversary. This is same as the adversary in the full security game in Definition 3.1 except with the aforementioned restrictions. For sake of completeness, we define this version of security in Definition A.1.

Later in Appendix F, we give a compiler that transforms the scheme in this section to one with full security, i.e., removing the static and all-receiver-corrupting restrictions on the adversary.

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 $0, 1, \ldots, L$. Each sender $u \in [n]$ is assigned to the $(2u-1)$-th wire in the input layer (i.e., layer-0), and each receiver $v \in [n]$ is assigned to the $(2v-1)$-th wire in 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 $L \times [2n]$ wires where we index the i-th wire in the ℓ-th layer by the tuple $(\ell, i) \in [L] \times [2n]$.[5] 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 $\mathsf{Output}_{(\ell,g)} \subseteq [2n]$. 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 $\{(\ell+1, w)\}_{w \in \mathsf{Output}_{(\ell,g)}}$. Finally, recall that a route 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 $\ell \in [L]$. Based on the description of routing network, also recall that $j_1 = 2u - 1 \in [2n]$ and $j_L = 2v - 1 \in [2n]$.

6.1 Construction

Simplifying assumption. Throughout this section, we shall assume that the message length $\mathsf{len} = 1$. This assumption is without loss of generality, since we can always parallel-compose multiple NIAR schemes where $\mathsf{len} = 1$ to get a NIAR scheme for $\mathsf{len} > 1$.

We now describe our basic NIAR scheme in detail.

Keys associated with wires. In our construction, each wire (ℓ, i) in the routing network will have the following associated with it:

- A PRF key $k_{(\ell,i)}$, which will be used to encrypt and decrypt the (signed) message along with its routing information on the wire;
- A message signing key tuple $(\mathsf{mpp}_{(\ell,i)}, \mathsf{msk}_{(\ell,i)}, \mathsf{mvk}_{(\ell,i)})$, which will later be used by the corresponding sender or obfuscated gate to sign the message to be sent to the wire;
- A route signing key tuple $(\mathsf{rpp}_{(\ell,i)}, \mathsf{rsk}_{(\ell,i)}, \mathsf{rvk}_{(\ell,i)})$, which will be used by the **Setup** algorithm to sign the routes and by the obfuscated gates to verify the routes before performing the routing.

[5] 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.

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

- For each wire $i \in \mathsf{Input}_{(\ell,g)}$ in layer ℓ: $k_{(\ell,i)}$, $\mathsf{mpp}_{(\ell,i)}$, $\mathsf{mvk}_{(\ell,i)}$, $\mathsf{rpp}_{(\ell,i)}$, $\mathsf{rvk}_{(\ell,i)}$.
- For each wire $i \in \mathsf{Output}_{(\ell,g)}$ in layer $\ell+1$: $k_{(\ell+1,i)}$, $\mathsf{mpp}_{(\ell+1,i)}$, $\mathsf{msk}_{(\ell+1,i)}$.

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. It computes as follows.

1. For each input wire $i \in \mathsf{Input}_{(\ell,g)}$:
   (a) If $\ell = 1$ and i is even, continue to next i. // *Filler, ignored.*
   (b) **Decrypt and authenticate the message/route:**
      i. Compute $y = \mathsf{CT}_{(\ell,i)} \oplus \mathsf{PRF}(k_{(\ell,i)}, t)$ and parse y as $(x, \overline{\mathsf{rte}}, \mathsf{msig})$.
      ii. Abort if $\mathsf{Sig}.\mathbf{Vf}(\mathsf{mpp}_{(\ell,i)}, \mathsf{mvk}_{(\ell,i)}, t, (x, \overline{\mathsf{rte}}), \mathsf{msig}) = 0$.
      iii. If $x = \perp_{\mathsf{filler}}$ and $\overline{\mathsf{rte}} = \perp_{\mathsf{filler}}$, continue to the next i. // *Filler, ignored.*
      iv. Parse $\overline{\mathsf{rte}}$ as $(\mathsf{rte}, \mathsf{rsig})$, rte as $(j_1, \ldots, j_L)$, and rsig as $(\mathsf{rsig}_1, \ldots, \mathsf{rsig}_L)$. Abort if $j_\ell \neq i$ or the next hop $j_{\ell+1} \notin \mathsf{Output}_{(\ell,g)}$ or $\mathsf{Sig}.\mathbf{Vf}(\mathsf{rpp}_{(\ell,i)}, \mathsf{rvk}_{(\ell,i)}, 1, \mathsf{rte}, \mathsf{rsig}_\ell) = 0$.
   (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. Else if $\ell + 1 < L$ (intermediate layer), first compute a new message signature $\mathsf{msig}' = \mathsf{Sig}.\mathbf{Sign}(\mathsf{mpp}_{(\ell+1,j)}, \mathsf{msk}_{(\ell+1,j)}, t, (x, \overline{\mathsf{rte}}))$. Then, compute the ciphertext $\mathsf{CT}_{(\ell+1,j)} = (x, \overline{\mathsf{rte}}, \mathsf{msig}') \oplus \mathsf{PRF}(k_{(\ell+1,j)}, t)$.
      iv. Else if $\ell + 1 = L$ (output layer), compute the ciphertext $\mathsf{CT}_{(L,j)} = x \oplus \mathsf{PRF}(k_{(L,j)}, t)$.
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 $x = \perp_{\mathsf{filler}}$ and $\overline{\mathsf{rte}} = \perp_{\mathsf{filler}}$.
   (b) Compute $\mathsf{msig}' = \mathsf{Sig}.\mathbf{Sign}(\mathsf{mpp}_{(\ell+1,j)}, \mathsf{msk}_{(\ell+1,j)}, t, (x, \overline{\mathsf{rte}}))$.
   (c) Compute $\mathsf{CT}_{(\ell+1,j)} = (x, \overline{\mathsf{rte}}, \mathsf{msig}') \oplus \mathsf{PRF}(k_{(\ell+1,j)}, t)$.
3. Output $\{\mathsf{CT}_{(\ell+1,i)} : i \in \mathsf{Output}_{(\ell,g)}\}$.

Figure 2: The circuit $\mathsf{Gate}_{(\ell,g)}$.

Circuit for each gate. We first describe the circuit for each gate to be obfuscated later in our construction. The circuit $\mathsf{Gate}_{(\ell,g)}$ denotes the g-th gate in the ℓ-th layer. It receives a ciphertext on each input wire and decrypts it using a PRF key to obtain a tuple $(x, \overline{\mathsf{rte}}, \mathsf{msig})$, where x is a

message, $\overline{\mathsf{rte}}$ is the authenticated route, and msig is a message signature. It verifies the message signature msig on the tuple $(x, \overline{\mathsf{rte}})$. Next, it performs route authentication and prepares the output ciphertext which varies depending on whether the wire is filler or not. A wire is filler if $x = \bot_{\mathsf{filler}}$ and $\overline{\mathsf{rte}} = \bot_{\mathsf{filler}}$. For a filler input wire, no route authentication is performed as there is no route associated with it. Computing output ciphertext for filler output wires is deferred to later as the circuit does not know which are filler output wires at the moment. For a non-filler input wire i, the circuit parses $\overline{\mathsf{rte}} = (\mathsf{rte}, \mathsf{rsig})$ and verifies that the route rte is valid using rsig. Then, it parses $\mathsf{rte} = (j_1, \ldots, j_L)$. If $j_\ell = i$, then it finds the corresponding non-filler output wire $j_{\ell+1}$ from the rte and computes a new message signature msig' and then a new ciphertext for the output wire in the natural manner. At the end, all output wires which do not have any ciphertext assigned to them are interpreted as filler wires and the circuit computes message signature and ciphertext similarly by setting $x = \bot_{\mathsf{filler}}$ and $\overline{\mathsf{rte}} = \bot_{\mathsf{filler}}$. In Figure 2 we describe the circuit formally and in more detail, where Sig is a SSU signature scheme constructed in Section 5 and PRF is a puncturable PRF.

We next describe the **Setup** algorithm.

Setup Algorithm. Given a routing permutation π, the **Setup** algorithm first sets $\mathsf{tlen} = \log^2(\lambda)$. Then, it runs the $\mathsf{AssignRoutes}$ algorithm to sample a set of edge-disjoint routes $\{\mathsf{rte}_u\}_{u\in[n]}$ 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 $(\mathsf{rsk}_{(\ell,i)}, \mathsf{rvk}_{(\ell,i)}, \mathsf{rpp}_{(\ell,i)})$ for signing routes, and (c) a signature key pair $(\mathsf{msk}_{(\ell,i)}, \mathsf{mvk}_{(\ell,i)}, \mathsf{mpp}_{(\ell,i)})$ for signing messages. Looking ahead, when proving security, the route signature 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 plaintexts".

Given the above set of keys, consider a sender/receiver pair (u, v) with route $\mathsf{rte}_u = (j_1, \ldots, j_L)$. Then sender u's sender key ek_u and receiver v's decryption key rk_v are defined as follows, where rsig_ℓ is the signature on rte_u computed using the route public param $\mathsf{rpp}_{(\ell,j_\ell)}$ and route signing key $\mathsf{rsk}_{(\ell,j_\ell)}$.

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

Lastly, the routing token tk is then defined as follows, where the circuit $\mathsf{Gate}_{(\ell,g)}$ is as described in Figure 2.

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

More formally, the **Setup** algorithm is as in Figure 3.

Setup$(1^\lambda, \mathsf{len} = 1, n, \pi)$: on inputs the security parameter 1^λ, the individual message length $\mathsf{len} = 1$, the number of parties n, and the permutation π, **Setup** does the following:

1. Set $\mathsf{tlen} = \log^2(\lambda)$.
2. **Sampling Routes**: Run the $\mathsf{AssignRoutes}$ procedure (Appendix B.1) on inputs $(1^\lambda, n, \pi)$. Abort if it outputs $\perp$. Else parse the output as a set of edge-disjoint routes $\{\mathsf{rte}_u\}_{u\in[n]}$ between each sender/receiver pair. Let $0, \ldots, L$ be the layers in the resulting network. Let G be the number of gates contained in each of the $L-1$ intermediate layers. Let W be the number of incoming and outgoing wires in each gate. Then, for all $u \in [n]$, the size of the string rte_u is $\mathsf{len}_{\mathsf{rte}} = L \cdot \log(2n)$.
3. **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) To sign and verify routes of length $\mathsf{len}_{\mathsf{rte}}$, sample route signature keys
   $$(\mathsf{rsk}_{(\ell,i)}, \mathsf{rvk}_{(\ell,i)}, \mathsf{rpp}_{(\ell,i)}) \leftarrow \mathsf{Sig}.\mathbf{Setup}(1^\lambda, 0, \mathsf{len}_{\mathsf{rte}}) \ .$$
   Suppose that the resulting route signatures will be of size $\mathsf{poly}_{\mathsf{rsig}}(\lambda)$ for some polynomials $\mathsf{poly}_{\mathsf{rsig}}$. Then, the messages signed will be of length $\mathsf{len}_m = \mathsf{tlen} + 1 + L \cdot \log(2n) + L \cdot \mathsf{poly}_{\mathsf{rsig}}(\lambda)$. To sign and verify messages of length len_m, sample message signature keys
   $$(\mathsf{msk}_{(\ell,i)}, \mathsf{mvk}_{(\ell,i)}, \mathsf{mpp}_{(\ell,i)}) \leftarrow \mathsf{Sig}.\mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len}_m) \ .$$
4. **Signing Routes**: For each sender $u \in [n]$ do the following:
   (a) Parse $\mathsf{rte}_u = (j_1, \ldots, j_L)$. Sign rte_u using route signing keys for each wire along rte_u, that is, for $\ell \in [L]$ compute $\mathsf{rsig}_\ell = \mathsf{Sig}.\mathbf{Sign}(\mathsf{rpp}_{(\ell,j_\ell)}, \mathsf{rsk}_{(\ell,j_\ell)}, 1, \mathsf{rte})$.
   (b) Set $\overline{\mathsf{rte}}_u = (\mathsf{rte}_u, \mathsf{rsig}_u = (\mathsf{rsig}_1, \ldots, \mathsf{rsig}_L))$.
5. **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 2.
   (b) Set $\mathsf{tk} = \{\overline{\mathsf{Gate}}_{(\ell,g)} : \ell \in [L-1], g \in [G]\}$.
6. **Setting Sender Keys**: For each $u \in [n]$, set $\mathsf{ek}_u = (k_{(1,j_1)}, \mathsf{mpp}_{(1,j_1)}, \mathsf{msk}_{(1,j_1)}, \overline{\mathsf{rte}}_u)$.
7. **Setting Receiver Keys**: For each $v \in [n]$, set $\mathsf{rk}_v = k_{(L,2v-1)}$.
8. Output $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk})$.

Figure 3: The **Setup** algorithm

Next, we describe 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 $(x, \overline{\mathsf{rte}}_u)$ for round t, and encrypts the tuple $(x, \overline{\mathsf{rte}}_u, \mathsf{msig})$ using its PRF key.

$\mathbf{Enc}(\mathsf{ek}_u, x_u, t)$ 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 $(k, \mathsf{mpp}, \mathsf{msk}, \overline{\mathsf{rte}}_u)$.
2. Compute the message signature $\mathsf{msig} = \mathsf{Sig}.\mathbf{Sign}(\mathsf{mpp}, \mathsf{msk}, t, (x_u, \overline{\mathsf{rte}}_u))$.
3. Compute the ciphertext $\mathsf{CT}_u = (x_u, \overline{\mathsf{rte}}_u, \mathsf{msig}) \oplus \mathsf{PRF}(k, t)$.
4. Output CT_u.

Routing Algorithm. The router receives a routing token tk from the **Setup** algorithm. It consists of obfuscation of each gate in the routing network as described in Figure 2. During each round t, the router receives n ciphertexts $\mathsf{CT}_1, \ldots, \mathsf{CT}_n$ from the n senders. Before processing the ciphertexts through the routing network, the router sets the $2n$ ciphertexts $\mathsf{CT}_{(1,1)}, \ldots, \mathsf{CT}_{(1,2n)}$ for the first layer as follows. For all $i \in [n]$, it sets $\mathsf{CT}_{(1,2i-1)} = \mathsf{CT}_i$ as the real ciphertexts and $\mathsf{CT}_{(1,2i)} = \perp_{\mathsf{filler}}$ as the *filler* ciphertexts, where $\perp_{\mathsf{filler}}$ is a special string. Next, the router uses the token tk to route the $2n$ ciphertexts in the first layer through the routing network to obtain the $2n$ ciphertexts in the last layer L: $\mathsf{CT}_{(L,1)}, \ldots, \mathsf{CT}_{(L,2n)}$. Finally, to all receivers $i \in [n]$, the router sends the ciphertexts $\mathsf{CT}'_i = \mathsf{CT}_{(L,2i-1)}$. More formally,

$\mathbf{Rte}(\mathsf{tk}, t, \mathsf{CT}_1, \mathsf{CT}_2, \ldots, \mathsf{CT}_n)$ on input the router token tk along with the round number t, and ciphertexts $\mathsf{CT}_1, \ldots, \mathsf{CT}_n$, 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 $\mathsf{CT}_{(1,2k-1)} = \mathsf{CT}_k$. // *Real ciphertexts*
   (b) For all $k \in [n]$, set $\mathsf{CT}_{(1,2k)} = \perp_{\mathsf{filler}}$. // *Filler ciphertexts*
3. Compute network of iO obfuscated gates layer-by-layer, that is, for layer $\ell = 1, \ldots, L-1$, 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 u learns its intended message by just decrypting the received ciphertext using its PRF key. More formally,

$\mathbf{Dec}(\mathsf{rk}_u, \mathsf{CT}'_u, t)$ on input user u's receiver key rk_u, output ciphertext CT'_u, and a time step t, does the following: Output $y = \mathsf{CT}'_u \oplus \mathsf{PRF}(\mathsf{rk}_u, t)$.

6.2 Efficiency Analysis

Recall that we assume $\mathsf{len} = 1$ since for multi-bit messages, since we can always parallel-compose multiple NIAR schemes where $\mathsf{len} = 1$ to get a NIAR scheme for $\mathsf{len} > 1$. In the analysis below, we argue that the router computation per time is bounded by $\widetilde{O}_\lambda(n)$ where $\widetilde{O}_\lambda$ hides $\mathsf{poly}(\lambda, \log n)$ factors for some fixed $\mathsf{poly}(\cdot)$.

Recall that the routing network consists of layers $0, \ldots, L$, where $L = O(\log n)$. In each of the $L-1$ intermediate layers, there are $G = 2n/W$ number of gates, where $W = O(\log^2 \lambda)$ is the number of incoming and outgoing wires in each gate.

Size of hardcoded values in each gate. Each incoming wire has the following hardcoded: PRF key of size $\mathsf{poly}(\lambda)$, route public parameters of size $\mathsf{poly}(\lambda)$ and route verification key of size $\mathsf{poly}(\lambda)$, message public parameters of size $\mathsf{poly}(\lambda)$ and message verification key of size $\mathsf{poly}(\lambda)$. Each outgoing wire has the following hardcoded: PRF key of size $\mathsf{poly}(\lambda)$, message public parameters of size $\mathsf{poly}(\lambda)$ and message signing key of size $\mathsf{poly}(\lambda, \mathsf{tlen}) = \mathsf{poly}(\lambda)$ as $\mathsf{tlen} = \log^2(\lambda)$.

Size of ciphertexts. Each route signature is of size $\mathsf{poly}_{\mathsf{rsig}}(\lambda)$. and each message signature is of size $\mathsf{poly}_{\mathsf{msig}}(\lambda)$. Therefore, the ciphertexts are of size $\mathsf{tlen} + 1 + L \cdot \log(2n) + L \cdot \mathsf{poly}_{\mathsf{rsig}}(\lambda) + \mathsf{poly}_{\mathsf{msig}}(\lambda) = \mathsf{poly}(\lambda, \log n)$.

Size and running time of each gate. Each gate has W incoming and outgoing wires and each gate processes W ciphertexts, where $W = O(\log^2 \lambda)$. Therefore, each gate has $\mathsf{poly}(\lambda)$ amount of hardcoded information and processes $\mathsf{poly}(\lambda, \log n)$ amount of inputs. Based on the operations inside each gate, we can then conclude that each gate is of size $\mathsf{poly}(\lambda, \log n)$. Then, accounting for the polynomial blowup of the iO obfuscator, we can conclude that the size of each obfuscated gate is still $\mathsf{poly}(\lambda, \log n)$ and the router can run each obfuscated circuit in time $\mathsf{poly}(\lambda, \log n)$.

Router computation per time step. Observe that for each time step, the router computes each of the obfuscated circuits at most once. Since there are at most $\tilde{O}(n)$ 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 $\mathsf{poly}(\cdot)$.

Sender and recevier key sizes, computation and communication per time step. Sender key size is bounded by the size of the route which is $\widetilde{O}_\lambda(1)$. For every sender, computation and communication per time step is $\widetilde{O}_\lambda(1)$. Each receiver's key contains a PRF key which is $O_\lambda(1)$ in size. For every receiver, computation and communication per time step is $O_\lambda(1)$.

6.3 Static Security Theorem

In Appendices D and E, we prove the following theorem, which shows that the above construction satisfies static security as long as the adversary always corrupts all receivers. In Appendix F, we give a compiler that further compiles the scheme to one that satisfies full security under adaptive corruptions, and without any restrictions on the adversary.

Theorem 6.1. *Suppose* PRF *is a secure puncturable PRF,* Sig *is a secure deterministic SSU signature scheme, and* iO *is a secure indistinguishability obfuscation scheme. Then, our NIAR*

construction in Section 6.1 satisfies full static corruption security (Definition A.1) subject to an all-receiver-corrupting adversary.

We give a proof roadmap of Theorem 6.1 below.

Proof roadmap. We prove Theorem 6.1 through a sequence of steps.

- In Definition D.1, we define indistinguishability w.r.t. inversions against an adversary that additionally satisfies the selective single-challenge restriction. Then, we present an Upgrade Theorem stated in Theorem D.2 which shows how to remove the selective single-challenge and inversion restrictions.
- Next, to complete the proof of Theorem 6.1, it suffices to prove security under the selective single-challenge and single inversion restrictions. We show this in Theorem E.1.

References

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

[ACD+19] Ittai Abraham, T-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *PODC*, 2019.

[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.

[AJS17] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation for turing machines: constant overhead and amortization. In *Annual International Cryptology Conference*, pages 252–279. Springer, 2017.

[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+01] 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.

[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.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 439–448, 2015.

[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.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *International conference on the theory and application of cryptology and information security*, pages 280–300. Springer, 2013.

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

[CG97] Benny Chor and Niv Gilboa. Computationally private information retrieval (extended abstract). In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 304–313, New York, NY, USA, 1997. ACM.

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

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 41–50, 1995.

[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.

[CHJV14] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Indistinguishability obfuscation of iterated circuits and ram programs. *Cryptology ePrint Archive*, 2014.

[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.

[DTT10] Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and prgs. In *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*, pages 649–665. Springer, 2010.

[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.

[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.

[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.

[GT20] Ashrujit Ghoshal and Stefano Tessaro. On the memory-tightness of hashed elgamal. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30*, pages 33–62. Springer, 2020.

[GWZ22] Jiaxin Guan, Daniel Wichs, and Mark Zhandry. Incompressible cryptography. In *Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part I*, pages 700–730. Springer, 2022.

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

[HIJ+17] Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-interactive multiparty computation without correlated randomness. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 181–211, Cham, 2017. Springer International Publishing.

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

[JJ22] Abhishek Jain and Zhengzhong Jin. Indistinguishability obfuscation via mathematical proofs of equivalence. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1023–1034. IEEE, 2022.

[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.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 419–428, 2015.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 669–684, 2013.

[KWZ22] Venkata Koppula, Brent Waters, and Mark Zhandry. Adaptive multiparty nike. In *Theory of Cryptography: 20th International Conference, TCC 2022, Chicago, IL, USA, November 7–10, 2022, Proceedings, Part II*, pages 244–273. Springer, 2022.

[Lam79] Leslie Lamport. Constructing digital signatures from a one way function. 1979.

[Mer79] Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* Stanford university, 1979.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings*, pages 111–126. Springer, 2002.

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

[RÖ2] 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.

[Sha48] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[SV23] Elaine Shi and Nikhil Vanjani. Multi-client inner product encryption: Function-hiding instantiations without random oracles. In *IACR International Conference on Public-Key Cryptography*, pages 622–651. Springer, 2023.

[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 Definition: NIAR Security under Static Corruption

We review the full security notion under static corruption, defined first by Shi and Wu [SW21]. 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. Security is defined with respect to the following experiment, parametrized by a bit $b \in \{0,1\}$, and a ***stateful*** adversary $\mathcal{A}$.

Static corruption experiment $\mathsf{NIARStatic}^{b,\mathcal{A}}(1^\lambda)$**.**

1. $(n, \mathsf{len}, \mathcal{K}_S, \mathcal{K}_R, \pi^{(0)}, \pi^{(1)}) \leftarrow \mathcal{A}(1^\lambda)$.
2. $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{len}, n, \pi^{(b)})$.
3. $\perp \leftarrow \mathcal{A}(\mathsf{tk}, \{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{rk}_u\}_{u\in\mathcal{K}_R})$.
4. For $t = 1, 2, \ldots$:
   - $\{x^{(0)}_{u,t}, x^{(1)}_{u,t}\}_{u\in\mathcal{H}_S} \leftarrow \mathcal{A}(\perp)$.
   - for $u \in \mathcal{H}_S$, $\mathsf{ct}_{u,t} \leftarrow \mathbf{Enc}(\mathsf{ek}_u, x^{(b)}_{u,t}, t)$
   - $\perp \leftarrow \mathcal{A}(\{\mathsf{CT}_{u,t}\}_{u\in\mathcal{H}_S})$.
5. At any time, $\mathcal{A}$ may halt and output an arbitrary function of its view. The experiment then also halts and returns the output of $\mathcal{A}$.

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

1. For all $u \in \mathcal{K}_S$, $\pi^{(0)}(u) = \pi^{(1)}(u)$; and
2. For all rounds t, and 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 A.1 (Security against static corruptions)**.** We say that a NIAR scheme satisfies security against static corruption, iff for any non-uniform p.p.t. admissible $\mathcal{A}$, its views in the two experiments $\mathsf{NIARStatic}^{0,\mathcal{A}}(1^\lambda)$ and $\mathsf{NIARStatic}^{1,\mathcal{A}}(1^\lambda)$ are computationally indistinguishable.

B Additional Preliminaries

B.1 Routing Networks

Imagine we have a directed 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. Suppose that producer $u \in [n]$ is assigned source vertex $2u - 1 \in [2n]$. Suppose that consumer $u \in [n]$ is

assigned destination vertex $2u-1 \in [2n]$. 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. Then, producer u is mapped to consumer $\pi(u)$. In the routing network, this translates to routing producer u's product from source vertex $2u-1$ to destination vertex $2\pi(u)-1$. 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 $u \in [n]$: 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 intermediate-layer vertices are henceforth called *gates*. Directed edges, henceforth called *wires*, exist only between adjacent layers ℓ and $\ell+1$.

- *Efficiency.* Each gate has $W = O(\log^2 \lambda)$ incoming wires and W outgoing wires. The network has $O(\log n)$ layers, and each intermediate layer has $G = 2n/W$ gates. Each gate has $O(\log^2 \lambda)$ incoming wires and $O(\log^2 \lambda)$ outgoing wires. The number of wires between any two adjacent layers is exactly $2n$.

- *Obliviousness.* The network and the corresponding $\mathsf{AssignRoutes}(1^\lambda, n, \pi)$ 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 B.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 two routing permutations π_0 and π_1 on $[n]$, let $C(\pi_0, \pi_1)$ be the set of senders that have the same destinations in π_0 and π_1, it holds that for either $b \in \{0,1\}$, the following two distributions have statistical distance at most $\mathsf{negl}(\lambda)$:

- Sample $(\mathsf{rte}_1, \ldots, \mathsf{rte}_n) \leftarrow \mathsf{AssignRoutes}(1^\lambda, n, \pi_b)$, and output $(\{\mathsf{rte}_u\}_{u \in C(\pi_0,\pi_1)}, \{\mathsf{rte}_u\}_{u \in [n] \setminus C(\pi_0,\pi_1)})$;
- Sample $\left(\{\mathsf{rte}_u\}_{u \in C(\pi_0,\pi_1)}, \{\mathsf{rte}_u^\beta\}_{u \in [n]\setminus C(\pi_0,\pi_1), \beta \in \{0,1\}}\right) \leftarrow \mathsf{AssignRoutes}^*(1^\lambda, n, \pi_0, \pi_1)$, and output $\left(\{\mathsf{rte}_u\}_{u \in C(\pi_0,\pi_1)}, \{\mathsf{rte}_u^b\}_{u \in [n]\setminus C(\pi_0,\pi_1)}\right)$.

The definition says that the simulated $\mathsf{AssignRoutes}^*$ algorithm takes in two routing permutations π_0 and π_1. For a sender $u \in C(\pi_0, \pi_1)$ with the same destinations in π_0 and π_1, $\mathsf{AssignRoutes}^*$ outputs a single route rte_u for u. For a sender $u \notin C(\pi_0, \pi_1)$ with different destinations in π_0 and π_1, $\mathsf{AssignRoutes}^*$ outputs two routes rte_u^0 and rte_u^1 for u. Not only so, for either $b \in \{0,1\}$, the union of the routes for senders' in $C(\pi_0, \pi_1)$, and the b-th set of routes $\{\mathsf{rte}_u^b\}_{u \notin C(\pi_0,\pi_1)}$ for everyone else output by the simulated $\mathsf{AssignRoutes}^*$ must be statistically indistinguishable from running the real-world $\mathsf{AssignRoutes}$ using permutation π_b.

Intuitively, in our NIAR scheme, only those who are in $C(\pi_0, \pi_1)$ can possibly be corrupt. Therefore, the definition decomposes the generation of the possibly-corrupt sender' routes from the remaining honest senders' destinations. In this sense, the possibly-corrupt senders' routes do not leak information about honest senders' destinations (beyond what is already leaked by the possibly-corrupt senders' destinations, and barring a negligibly small statistical difference).

Remark B.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 we can obtain a routing network that satisfies Definition B.1 using techniques from Asharov et al. [ACN+20] and Ramachandran and Shi [RS21]. Asharov et al. [ACN+20] and Ramachandran and Shi [RS21] propose a butterfly network where each gate has polylogarithmically many incoming and outgoing wires. We can compose two instances of their butterfly network back-to-back. The first instance is to route each input element (i.e., those on input wires $2u - 1$ for $u \in [n]$) to a random and distinct wire in the output layer. The second instance will then route each element u to its correct destination, i.e., $2\pi(u) - 1$ of the output layer.

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.

B.2 Puncturable PRF

A puncturable pseudorandom function consists of the following algorithms:

- $\mathsf{sk} \leftarrow \mathbf{Gen}(1^\lambda, 1^\ell)$: takes in a security parameter 1^λ, the length 1^ℓ of the input messages where $\ell := \ell(\lambda)$ is a polynomial function in λ, and outputs a PRF key K.
- $\sigma \leftarrow \mathbf{Eval}(K, x)$: a *deterministic* function that takes in a PRF key K, a message $x \in \{0,1\}^\ell$, and outputs the evaluation outcome σ.
- $K^* \leftarrow \mathbf{Puncture}(K, S = \{x_1^*, \ldots, x_{|S|}^*\})$: takes in a PRF key K, the set of messages $S = \{x_1^*, \ldots, x_{|S|}^*\}$ that the PRF key needs to be punctured on, and outputs a punctured key K^*.
- $\sigma \leftarrow \mathbf{PEval}(K^*, x)$: a *deterministic* function that takes in a punctured PRF key K^*, and a message $x \in \{0,1\}^{\mathsf{len}}$, outputs the evaluation outcome σ.

Correctness. We say that a puncturable PRF scheme satisfies correctness if the punctured key preserves functionality when evaluated at unpunctured points. Formally, we require that for any λ, ℓ, any $S = \{x_1^*, \ldots, x_{|S|}^*\}$, any $x \in \{0,1\}^{\mathsf{len}}$ such that $x \notin S$,

$$\Pr\left[\begin{array}{l} K \leftarrow \mathbf{Gen}(1^\lambda, \ell), \\ K^* \leftarrow \mathbf{Puncture}(K, S) \end{array} : \mathbf{Eval}(K, x) = \mathbf{PEval}(K^*, x)\right] = 1$$

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

- The stateful adversary $\mathcal{A}$ sends a set S to the challenger. The challenger computes a PRF key $K \leftarrow \mathbf{Gen}(1^\lambda, 1^\ell)$. The challenger then computes the punctured key $K^* \leftarrow \mathbf{Puncture}(K, S)$ and sends it back to the adversary.
- The adversary $\mathcal{A}$ can adaptively make evaluation queries on messages $x_i \notin S$ to the challenger. For each such query, the challenger responds as follows. If $b = 0$, it sends $\mathbf{Eval}(K, x_i)$ to the adversary. If $b = 1$, it sends a uniformly random string to the adversary.
- $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$, the experiment outputs b'.

Definition B.3 (Puncturable PRF security)**.** We say that a puncturable PRF scheme is secure iff for any ℓ polynomially bounded by λ, for any non-uniform p.p.t. adversary $\mathcal{A}$, its views in the two experiments $\mathsf{ExptPPRF}^{\mathcal{A},b}(1^\lambda, 1^\ell)$ and $\mathsf{ExptPPRF}^{\mathcal{A},b}(1^\lambda, 1^\ell)$ are computationally indistinguishable.

Theorem B.4 ([GGM86,BW13,BGI14,KPTZ13])**.** *If one-way functions exist, then for all efficiently computable $\ell(\lambda)$, there exists a secure puncturable PRF.*

Shorthand notations. Sometimes for we use $\mathsf{PRF}(K, \cdot)$ as a shorthand for $\mathsf{PRF}.\mathbf{Eval}(K, \cdot)$ and $\mathsf{PRF}(K^*, \cdot)$ as a shorthand for $\mathsf{PRF}.\mathbf{PEval}(K^*, \cdot)$.

B.3 Single-Point Binding (SPB) Hash Function

Single-point binding hash functions were introduced and constructed in Guan et al. [GWZ22] and Koppula et al. [KWZ22]. Here, we modify their defintion in that the **GenBind** algorithm additionally also outputs a normal hash key. A single-point binding hash function is a triple $(\mathbf{Gen}, \mathbf{Hash}, \mathbf{GenBind})$ where:

- $\mathsf{hk} \leftarrow \mathbf{Gen}(1^\lambda, 1^\ell)$: takes as input the security parameter λ, and input length ℓ. It produces a hash key hk.
- $h = \mathbf{Hash}(\mathsf{hk}, m \in \{0,1\}^\ell)$: deterministically produces a hash digest h whose length is some fixed polynomial in λ independent of ℓ.
- $(\mathsf{hk}, \mathsf{hk}^*) \leftarrow \mathbf{GenBind}(1^\lambda, 1^\ell, m^* \in \{0,1\}^\ell)$: takes as input λ, input length ℓ, and a message m^*, and outputs a normal hash key $\widetilde{\mathsf{hk}}$ and a binding hash key hk^*.

Correctness. An SPB hash function is said to be correct iff the following holds. For any ℓ which is upper bounded by some fixed polynomial function in λ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any $m^* \in \{0,1\}^\ell$, for any λ,

$$\Pr\left[\begin{array}{r}(\mathsf{hk}, \mathsf{hk}^*) \leftarrow \mathbf{GenBind}(1^\lambda, 1^\ell, m^*) : \\ \exists m \neq m^* \text{ s.t. } \mathbf{Hash}(\mathsf{hk}, m) \neq \mathbf{Hash}(\mathsf{hk}^*, m)\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Security. An SPB hash function is said to be secure iff it satisfies the following properties.

- **Identical distribution of normal hash keys.** For any $\lambda, \ell, m^* \in \{0,1\}^\ell$, we have the following where $\equiv$ denotes identical distribution.

$$\begin{aligned}&\{\mathsf{hk} \leftarrow \mathbf{Gen}(1^\lambda, 1^\ell) : \text{ output } \mathsf{hk}\} \\ \equiv &\{(\mathsf{hk}, \mathsf{hk}^*) \leftarrow \mathbf{GenBind}(1^\lambda, 1^\ell, m^*) : \text{ output } \mathsf{hk}\}\end{aligned}$$

- **Indistinguishability of normal and binding hash keys.** For all ℓ that is polynomially bounded by λ, all $m^* \in \{0,1\}^\ell$, we have the following where $\approx$ denotes computational indistinguishability.

$$\{(\mathsf{hk}, \mathsf{hk}^*) \leftarrow \mathbf{GenBind}(1^\lambda, 1^\ell, m^*) : \text{ output } (m^*, \mathsf{hk})\}$$
$$\approx \{(\mathsf{hk}, \mathsf{hk}^*) \leftarrow \mathbf{GenBind}(1^\lambda, 1^\ell, , m^*) : \text{ output } (m^*, \mathsf{hk}^*)\}$$

- **Statistically binding at m^*.** For any ℓ that is polynomially bounded in λ, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for all λ, all $m^* \in \{0,1\}^\ell$,

$$\Pr\left[\begin{array}{r}(\mathsf{hk}, \mathsf{hk}^*) \leftarrow \mathbf{GenBind}(1^\lambda, 1^\ell, m^*) : \\ \exists m \neq m^* \text{ s.t. } \mathbf{Hash}(\mathsf{hk}^*, m) = \mathbf{Hash}(\mathsf{hk}^*, m^*)\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Instantiations. Guan et al. [GWZ22] provided two constructions of SPB hash functions, one from indistinguishability obfuscation, and the other from leveled fully homomorphic encryption.

B.4 Single-Point Binding (SPB) Signatures

Single-point binding signatures were introduced and constructed in Guan et al. [GWZ22] and Koppula et al. [KWZ22]. We will use it as a building block for our SSU signatures construction. A single-point binding signature scheme is a quadruple of algorithms $(\mathbf{Gen}, \mathbf{Sign}, \mathbf{Vf}, \mathbf{GenBind})$ defined as follows:

- $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathbf{Gen}(1^\lambda, 1^\ell)$: is a *randomized* algorithm that takes as input the security parameter λ and message length $\ell := \ell(\lambda)$ which is a polynomial function in λ. It outputs a signing key sk and a verification key vk whose lengths are upper bounded by some fixed polynomial function in λ and ℓ.
- $\sigma \leftarrow \mathbf{Sign}(\mathsf{sk}, m \in \{0,1\}^\ell)$: is a *deterministic* algorithm that takes as input a signing key sk and a message m. It outputs a signature σ whose length is upper bounded by a fixed polynomial function in λ and ℓ.
- $(0 \text{ or } 1) \leftarrow \mathbf{Vf}(\mathsf{vk}, m \in \{0,1\}^\ell, \sigma)$: is a *deterministic* algorithm that takes as input a verification key vk, a message m and a signature σ on m and outputs 1 if the signature is valid, else 0.
- $(\mathsf{vk}^*, \sigma^*) \leftarrow \mathbf{GenBind}(1^\lambda, 1^\ell, m^* \in \{0,1\}^\ell)$: is a *randomized* algorithm that takes as input the security parameter λ, message length ℓ, and a message m^*. It outputs a binding verification key vk^* and a signature σ^* on message m^*.

Correctness. A single-point binding signature scheme is said to be correct iff the following holds.

- For all $\lambda, \ell \in \mathbb{N}$, and all $m \in \{0,1\}^\ell$,

$$\Pr[(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathbf{Gen}(1^\lambda, 1^\ell), \sigma \leftarrow \mathbf{Sign}(\mathsf{sk}, m) : \mathbf{Vf}(\mathsf{vk}, m, \sigma) = 1] = 1$$

- For all $\lambda, \ell \in \mathbb{N}$, and all $m^* \in \{0,1\}^\ell$,

$$\Pr[(\mathsf{vk}^*, \sigma^*) \leftarrow \mathbf{GenBind}(1^\lambda, 1^\ell, m^*) : \mathbf{Vf}(\mathsf{vk}^*, m^*, \sigma^*) = 1] = 1$$

Security. A single-point binding signature scheme is said to be secure if it has the following properties:

- **Indistinguishability of normal and binding keys.** For any ℓ polynomially bounded by λ, any $m^* \in \{0,1\}^\ell$, we have the following where $\approx$ denotes computational indistinguishability.

$$\begin{aligned} & \{(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathbf{Gen}(1^\lambda, 1^\ell), \sigma \leftarrow \mathbf{Sign}(\mathsf{sk}, m^*) : \text{ output } (\mathsf{vk}, \sigma)\} \\ \approx \quad & \{(\mathsf{vk}^*, \sigma^*) \leftarrow \mathbf{GenBind}(1^\lambda, 1^\ell, m^*) : \text{ output } (\mathsf{vk}^*, \sigma^*)\} \end{aligned}$$

- **Statistically binding at m^*.** For any ℓ polynomially bounded by λ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all λ, and all $m^* \in \{0,1\}^\ell$,

$$\Pr\left[\begin{array}{c} (\mathsf{vk}^*, \sigma^*) \leftarrow \mathbf{GenBind}(1^\lambda, 1^\ell, m^*) : \\ \exists\ \sigma \text{ and } m \neq m^* \text{ s.t. } \mathbf{Vf}(\mathsf{vk}^*, m, \sigma) = 1 \end{array}\right] \leq \mathsf{negl}(\lambda),$$

$$\Pr\left[\begin{array}{c} (\mathsf{vk}^*, \sigma^*) \leftarrow \mathbf{GenBind}(1^\lambda, m^*) : \\ \exists\ \sigma \neq \sigma^* \text{ s.t. } \mathbf{Vf}(\mathsf{vk}^*, m^*, \sigma) = 1 \end{array}\right] \leq \mathsf{negl}(\lambda).$$

  This means that with overwhelming probability over the choice of the binding verification key vk^* output by **GenBind**, any message $m \neq m^*$ does not have a valid signature that would be accepted by vk^*. Further, there is a unique signature σ^* for message m^* that vk^* accepts.

Instantiations. Guan et al. [GWZ22] provided low-rate and high-rate constructions of SPB signature schemes. We will use the low-rate construction in this paper. Guan et al. [GWZ22] showed how to construct low-rate SPB signature scheme assuming one-way functions.

B.5 Indistinguishability Obfuscation

The notion of indistinguishability obfuscation (iO) was first defined in [BGI+01]. 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 B.5 (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

$$\Pr\left[C'(x) = C(x) : C' \leftarrow \mathsf{iO}(1^\lambda, C)\right] = 1.$$

- **Polynomial 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), the distributions $\{\mathsf{iO}(1^\lambda, C_{0,\lambda})\}_\lambda$ and $\{\mathsf{iO}(1^\lambda, C_{1,\lambda})\}_\lambda$ are computationally indistinguishable.

B.6 Compression Lemma

In our proof for impossibility of fully simulation security of NIAR, we use the compression lemma that was formalized in earlier works [DTT10, GT20] which roughly means that it is impossible to compress every element in a set with cardinality c to a string less than $\log c$ bits long, even relative to a random string. We state the compression lemma here as a proposition.

Proposition B.6. *Suppose there is an (not necessarily efficient) encoding procedure* $\mathsf{Encode} : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ *and a (not necessarily efficient) decoding procedure* $\mathsf{Decode} : \mathcal{Y} \times \mathcal{R} \to \mathcal{X}$ *such that*

$$\Pr_{x \in \mathcal{X}, r \in \mathcal{R}}[\mathsf{Decode}(\mathsf{Encode}(x, r), r) = x] \geq \epsilon,$$

then $\log |\mathcal{Y}| \geq \log |\mathcal{X}| - \log (1/\epsilon)$.

B.7 Selective Opening Security for PRFs

In our adaptive corruption scheme, we need a PRF that is secure against selective opening attacks. The definition of such a PRF was given in the work of Abraham et al. [ACD+19]. Moreover, they showed that the standard PRF security notion implies selective opening security except with a polynomial loss in the security failure probability.

Specifically, selective opening security is defined as follows, borrowing verbatim from Abraham et al. [ACD+19].

We consider a selective opening adversary that interacts with a challenger. The adversary can request to create new PRF instances, query existing instances with specified messages, selectively corrupt instances and obtain the secret keys of these instances, and finally, we would like to claim that for instances that have not been corrupt, the adversary is unable to distinguish the PRFs' evaluation outcomes on any future message from random values from an appropriate domain. More formally, we consider the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

$\underline{\mathsf{PRFExpt}_b^{\mathcal{A}}(1^\lambda)}$: $\mathcal{A}(1^\lambda)$ can adaptively interact with $\mathcal{C}$ through the following queries:

- *Create instance.* The challenger $\mathcal{C}$ creates a new PRF instance by calling the honest PRF key generation algorithm $\mathbf{Gen}(1^\lambda)$. Henceforth, the instance will be assigned an index that corresponds to the number of "create instance" queries made so far. The i-th instance's secret key will be denoted sk_i.
- *Evaluate.* The adversary $\mathcal{A}$ specifies an index i that corresponds to an instance already created and a message m, and the challenger computes $r \leftarrow \mathsf{PRF}_{\mathsf{sk}_i}(m)$ and returns r to $\mathcal{A}$.
- *Corrupt.* The adversary $\mathcal{A}$ specifies an index i, and the challenger $\mathcal{C}$ returns sk_i to $\mathcal{A}$ (if the i-th instance has been created).
- *Challenge.* The adversary $\mathcal{A}$ specifies an index i^* that must have been created and a message m. If $b = 0$, the challenger returns a completely random string of appropriate length. If $b = 1$, the challenger computes $r \leftarrow \mathsf{PRF}_{\mathsf{sk}_{i^*}}(m)$ and returns r to $\mathcal{A}$.

We say that $\mathcal{A}$ is admissible iff with probability 1, every challenge tuple (i^*, m) it submits satisfies the following: 1) $\mathcal{A}$ does not make a corruption query on i^* throughout the game; and 2) $\mathcal{A}$ does not make any evaluation query on the tuple (i^*, m).

Definition B.7 (Selective opening security of a PRF family)**.** We say that a PRF scheme satisfies pseudorandomness under selective opening iff for any admissible non-uniform p.p.t. adversary $\mathcal{A}$, its views in $\mathsf{PRFExpt}_0^{\mathcal{A}}(1^\lambda)$ and $\mathsf{PRFExpt}_1^{\mathcal{A}}(1^\lambda)$ are computationally indistinguishable.

C SSU Signatures: Construction and Proof

In this section we show how how to construct SSU Signatures and provide the security proof of the construction.

C.1 SSU Signatures Construction

Let $\Sigma = (\Sigma.\mathbf{Gen}, \Sigma.\mathbf{Sign}, \Sigma.\mathbf{Vf}, \Sigma.\mathbf{GenBind})$ be a single-point binding (SPB) signature scheme as defined in Appendix B.4. Let $\mathsf{H} = (\mathsf{H}.\mathbf{Gen}, \mathsf{H}.\mathbf{Hash}, \mathsf{H}.\mathbf{GenBind})$ be a single-point binding (SPB) hash function as in Appendix B.3. Let PPRF be a puncturable PRF.

The SSU signature scheme is based on a binary tree of SPB signatures intuitively described in Figures 1a and 1b in Section 5.2. Formally, consider a binary tree of depth $\mathsf{tlen}+1$ consisting of levels $0, 1, \ldots, \mathsf{tlen}$. At level tlen, there are 2^{tlen} leaf nodes, each corresponding to a time step $t \in \{0,1\}^{\mathsf{tlen}}$ based on the binary representation of t.

We describe the structure of this binary tree next.

- Every node has a unique identifier denoted as either u_i or sib_i in Figures 1a and 1b.
- At level 0, there is a root node u_0 whose associated SPB signature key tuple $(\mathsf{sk}_{\mathsf{u}_0}, \mathsf{vk}_{\mathsf{u}_0})$ will be computed during the **Setup** algorithm.
- Each intermediate or leaf node u_i will also have an associated SPB signature key tuple that can be computed on the fly using the PPRF key K. The keys will be computed as $(\mathsf{sk}_{\mathsf{u}_i}, \mathsf{vk}_{\mathsf{u}_i}) = \Sigma.\mathbf{Gen}(1^\lambda, 1^\ell; \mathsf{PPRF}.\mathbf{Eval}(K, \mathsf{u}_i))$. Here, we use the notation $\Sigma.\mathbf{Gen}(1^\lambda, 1^\ell; \mathsf{PPRF}.\mathbf{Eval}(K, \mathsf{u}_i))$ to mean running the $\Sigma.\mathbf{Gen}(1^\lambda, 1^\ell)$ algorithm and seeding its random tape with the coins generated by $\mathsf{PPRF}.\mathbf{Eval}(K, \mathsf{u}_i)$.
- Each level $j \in \{0, 1, \ldots, \mathsf{tlen}\}$ also has a hash key hk_j associated with it that will be computed during **Setup**.

The SSU signature scheme is as follows. Recall that the SPB hash outputs a hash digest whose length is a fixed polynomial in λ, and independent of the input length — henceforth let $\ell_h(\lambda)$ denote this length. Let $\ell_{\mathsf{vk}}(\lambda)$ be the length of the verification key when we run $\Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)})$.

$\mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len})$:

1. Compute $(\mathsf{sk}_{\mathsf{u}_0}, \mathsf{vk}_{\mathsf{u}_0}) \leftarrow \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)})$.
2. Compute $K \leftarrow \mathsf{PPRF}.\mathbf{Gen}(1^\lambda, 1^{\mathsf{tlen}+1})$.
3. For $j \in \{0, \ldots, \mathsf{tlen}-1\}$, sample $\mathsf{hk}_j \leftarrow \mathsf{H}.\mathbf{Gen}(1^\lambda, 1^{2\ell_{\mathsf{vk}}(\lambda)})$. Finally, sample $\mathsf{hk}_{\mathsf{tlen}} \leftarrow \mathsf{H}.\mathbf{Gen}(1^\lambda, 1^{\mathsf{len}+\mathsf{tlen}})$.
4. Output $\mathsf{sk} = (\mathsf{sk}_{\mathsf{u}_0}, K)$, $\mathsf{vk} = \mathsf{vk}_{\mathsf{u}_0}$ and $\mathsf{pp} = \{\mathsf{hk}_j\}_{j\in\{0,\ldots,\mathsf{tlen}\}}$.

$\mathbf{Sign}(\mathsf{pp}, \mathsf{sk}, t, x)$:

1. Parse $\mathsf{pp} = \{\mathsf{hk}_j\}_{j\in\{0,\ldots,\mathsf{tlen}\}}$ and $\mathsf{sk} = (\mathsf{sk}_{\mathsf{u}_0}, K)$.
2. Let the nodes on the path from the root to t (excluding the root u_0) be $\mathsf{u}_1, \ldots, \mathsf{u}_{\mathsf{tlen}}$ and let their siblings be $\mathsf{sib}_1, \ldots, \mathsf{sib}_{\mathsf{tlen}}$ as illustrated in Figure 1a.
3. For all $j \in [\mathsf{tlen}]$: compute $(\mathsf{sk}_{\mathsf{u}_j}, \mathsf{vk}_{\mathsf{u}_j}) = \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)}; \mathsf{PPRF}.\mathbf{Eval}(K, \mathsf{u}_j))$ and $(\mathsf{sk}_{\mathsf{sib}_j}, \mathsf{vk}_{\mathsf{sib}_j}) = \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h}; \mathsf{PPRF}.\mathbf{Eval}(K, \mathsf{sib}_j))$.

4. Compute $\alpha_0, \ldots, \alpha_{\mathsf{tlen}}$ as follows:

$$\alpha_j = \begin{cases} \mathsf{vk}_{\mathsf{u}_{j+1}} || \mathsf{vk}_{\mathsf{sib}_{j+1}} & \text{if } j \neq \mathsf{tlen} \text{ and } j^{th}\text{-bit of } t \text{ is } 0, \\ \mathsf{vk}_{\mathsf{sib}_{j+1}} || \mathsf{vk}_{\mathsf{u}_{j+1}} & \text{if } j \neq \mathsf{tlen} \text{ and } j^{th}\text{-bit of } t \text{ is } 1, \\ t || x & \text{if } j = \mathsf{tlen}. \end{cases}$$

5. Sign the hash of the message α_{tlen} with the signing key of the leaf node $\mathsf{u}_{\mathsf{tlen}}$ corresponding to t and also sign the hash of the verification keys of the nodes and their siblings (denoted α_j for level $j+1$) on the path (excluding root node) from root to t by their parent's signing key (denoted $\mathsf{sk}_{\mathsf{u}_j}$ for level j) as follows.

$$\text{for all } j \in \{0, \ldots, \mathsf{tlen}\}\text{: } \sigma_j = \Sigma.\mathbf{Sign}(\mathsf{sk}_{\mathsf{u}_j}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}_j, \alpha_j)).$$

6. Output $\sigma = ((\sigma_0, \mathsf{vk}_{\mathsf{u}_1}, \mathsf{vk}_{\mathsf{sib}_1}), \ldots, (\sigma_{\mathsf{tlen}-1}, \mathsf{vk}_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{vk}_{\mathsf{sib}_{\mathsf{tlen}}}), \sigma_{\mathsf{tlen}})$.

$\mathbf{Vf}(\mathsf{pp}, \mathsf{vk}, t, x, \sigma)$:

1. Let the nodes on the path (excluding root node) from root to t be $\mathsf{u}_1, \ldots, \mathsf{u}_{\mathsf{tlen}}$ and let their siblings be $\mathsf{sib}_1, \ldots, \mathsf{sib}_{\mathsf{tlen}}$.
2. Parse $\sigma = ((\sigma_0, \mathsf{vk}_{\mathsf{u}_1}, \mathsf{vk}_{\mathsf{sib}_1}), \ldots, (\sigma_{\mathsf{tlen}-1}, \mathsf{vk}_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{vk}_{\mathsf{sib}_{\mathsf{tlen}}}), \sigma_{\mathsf{tlen}})$, $\mathsf{vk} = \mathsf{vk}_{\mathsf{u}_0}$, and $\mathsf{pp} = \{\mathsf{hk}_j\}_{j \in \{0,\ldots,\mathsf{tlen}\}}$.
3. Compute $\alpha_0, \ldots, \alpha_{\mathsf{tlen}}$ as described in step 4 of **Sign** algorithm.
4. Output 1 iff all the following checks pass, else output 0.

$$\text{for all } j \in \{0, \ldots, \mathsf{tlen}\}\text{: } \Sigma.\mathbf{Vf}(\mathsf{vk}_{\mathsf{u}_j}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}_j, \alpha_j), \sigma_j) = 1.$$

$\mathbf{PuncturedSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*)$:

Compute $(\mathsf{sk}, \mathsf{vk}, \mathsf{pp})$ *just like in* **Setup** *algorithm. We spell out the details below for completeness.*

1. Compute $(\mathsf{sk}_{\mathsf{u}_0}, \mathsf{vk}_{\mathsf{u}_0}) \leftarrow \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)})$.
2. Compute $K \leftarrow \mathsf{PPRF}.\mathbf{Gen}(1^\lambda, 1^{\mathsf{tlen}+1})$.
3. For $j \in \{0, \ldots, \mathsf{tlen}-1\}$, let $\mathsf{hk}_j \leftarrow \mathsf{H}.\mathbf{Gen}(1^\lambda, 1^{2\ell_{\mathsf{vk}}(\lambda)})$; and let $\mathsf{hk}_{\mathsf{tlen}} \leftarrow \mathsf{H}.\mathbf{Gen}(1^\lambda, 1^{\mathsf{len}+\mathsf{tlen}})$.
4. Let $\mathsf{sk} = (\mathsf{sk}_{\mathsf{u}_0}, K)$, $\mathsf{vk} = \mathsf{vk}_{\mathsf{u}_0}$, $\mathsf{pp} = \{\mathsf{hk}_j\}_{j \in \{0,\ldots,\mathsf{tlen}\}}$.

Compute signature $\widetilde{\sigma}$ *on* (t^*, x^*) *just like in* **Sign** *algorithm. We spell out the details below for completeness.*

5. Let the nodes on the path (excluding root node) from root to t^* be $\mathsf{u}_1, \ldots, \mathsf{u}_{\mathsf{tlen}}$ and let their siblings be $\mathsf{sib}_1, \ldots, \mathsf{sib}_{\mathsf{tlen}}$.
6. For all $j \in [\mathsf{tlen}]$, compute $(\mathsf{sk}_{\mathsf{u}_j}, \mathsf{vk}_{\mathsf{u}_j}) = \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)}; \mathsf{PPRF}.\mathbf{Eval}(K, \mathsf{u}_j))$, and $(\mathsf{sk}_{\mathsf{sib}_j}, \mathsf{vk}_{\mathsf{sib}_j}) = \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)}; \mathsf{PPRF}.\mathbf{Eval}(K, \mathsf{sib}_j))$.
7. Compute $\alpha_0, \ldots, \alpha_{\mathsf{tlen}}$ as described in step 4 of **Sign** algorithm.

8. Compute the signature on (t^*, x^*) and the signatures on the verification keys of the nodes and their siblings on the path (excluding root node) from root to t^*:

$$\text{for all } j \in \{0, \ldots, \mathsf{tlen}\}\text{: } \widetilde{\sigma}_j = \Sigma.\mathbf{Sign}(\mathsf{sk}_{\mathsf{u}_j}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}_j, \alpha_j))$$

9. Let $\widetilde{\sigma} = ((\widetilde{\sigma}_0, \mathsf{vk}_{\mathsf{u}_1}, \mathsf{vk}_{\mathsf{sib}_1}), \ldots, (\widetilde{\sigma}_{\mathsf{tlen}-1}, \mathsf{vk}_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{vk}_{\mathsf{sib}_{\mathsf{tlen}}}), \widetilde{\sigma}_{\mathsf{tlen}})$.

Puncture the PRF key and compute the outputs.

10. Compute $K^* \leftarrow \mathsf{PPRF}.\mathbf{Puncture}(K, \{\mathsf{u}_j\}_{j\in\{0,1,\ldots,\mathsf{tlen}\}})$ as the punctured PRF key.
11. Let $\widetilde{\mathsf{sk}} = (\widetilde{\sigma}, t^*, x^*, K^*)$. Output $(\mathsf{sk}, \widetilde{\mathsf{sk}}, \mathsf{vk}, \mathsf{pp})$.

$\underline{\mathbf{BindingSetup}}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*)$:

1. Let the nodes on the path (excluding root node) from root to t^* be $\mathsf{u}_1, \ldots, \mathsf{u}_{\mathsf{tlen}}$ and let their siblings be $\mathsf{sib}_1, \ldots, \mathsf{sib}_{\mathsf{tlen}}$.
2. Compute $K \leftarrow \mathsf{PPRF}.\mathbf{Gen}(1^\lambda, 1^{\mathsf{tlen}+1})$ and $K^* \leftarrow \mathsf{PPRF}.\mathbf{Puncture}(K, \{\mathsf{u}_j\}_{j\in\{0,1,\ldots,\mathsf{tlen}\}})$.
3. For all $j \in [\mathsf{tlen}]$, compute $(\mathsf{sk}_{\mathsf{sib}_j}, \mathsf{vk}_{\mathsf{sib}_j}) = \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)}; \mathsf{PPRF}.\mathbf{Eval}(K, \mathsf{sib}_j))$.
4. Compute the binding keys and associated signatures in the following sequence:
   - Let $\alpha^*_{\mathsf{tlen}} = t^* || x^*$.
   - Compute $(\mathsf{hk}_{\mathsf{tlen}}, \mathsf{hk}^*_{\mathsf{tlen}}) \leftarrow \mathsf{H}.\mathbf{GenBind}(1^\lambda, 1^{\mathsf{len}+\mathsf{tlen}}, \alpha^*_{\mathsf{tlen}})$.
   - Compute $(\mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}, \sigma^*_{\mathsf{tlen}}) = \Sigma.\mathbf{GenBind}(1^\lambda, 1^{\ell_h(\lambda)}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_{\mathsf{tlen}}, \alpha^*_{\mathsf{tlen}}))$.
   - For $j = \mathsf{tlen} - 1, \ldots, 0$ in decreasing order:
     - Let $\alpha^*_j = \begin{cases} \mathsf{vk}^*_{\mathsf{u}_{j+1}} || \mathsf{vk}_{\mathsf{sib}_{j+1}} & \text{if } j^{th}\text{-bit of } t \text{ is } 0 \\ \mathsf{vk}_{\mathsf{sib}_{j+1}} || \mathsf{vk}^*_{\mathsf{u}_{j+1}} & \text{if } j^{th}\text{-bit of } t \text{ is } 1 \end{cases}$.
     - Compute $(\mathsf{hk}_j, \mathsf{hk}^*_j) \leftarrow \mathsf{H}.\mathbf{GenBind}(1^\lambda, 1^{2\ell_{\mathsf{vk}}(\lambda)}, \alpha^*_j)$.
     - Compute $(\mathsf{vk}^*_{\mathsf{u}_j}, \sigma^*_j) = \Sigma.\mathbf{GenBind}(1^\lambda, 1^{\ell_h(\lambda)}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_j, \alpha^*_j))$.
5. Let $\sigma^* = ((\sigma^*_0, \mathsf{vk}^*_{\mathsf{u}_1}, \mathsf{vk}_{\mathsf{sib}_1}), \ldots, (\sigma^*_{\mathsf{tlen}-1}, \mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{vk}_{\mathsf{sib}_{\mathsf{tlen}}}), \sigma^*_{\mathsf{tlen}})$.
6. Let $\mathsf{sk}^* = (\sigma^*, t^*, x^*, K^*)$, $\mathsf{vk}^* = \mathsf{vk}^*_{\mathsf{u}_0}$, $\mathsf{pp}^* = \{\mathsf{hk}^*_j\}_{j\in\{0,\ldots,\mathsf{tlen}\}}$.
7. Output $(\mathsf{sk}^*, \mathsf{vk}^*, \mathsf{pp}^*)$.

$\underline{\mathbf{PSign}}(\mathsf{pp}, \widetilde{\mathsf{sk}}, t, x)$:

1. Parse $\mathsf{pp} = \{\mathsf{hk}_j\}_{j\in\{0,\ldots,\mathsf{tlen}\}}$. and $\widetilde{\mathsf{sk}} = (\widetilde{\sigma}, t^*, x^*, K^*)$. Let the nodes on the path (excluding root node) from root to t^* be $\mathsf{u}^*_1, \ldots, \mathsf{u}^*_{\mathsf{tlen}}$ and let their siblings be $\mathsf{sib}^*_1, \ldots, \mathsf{sib}^*_{\mathsf{tlen}}$. Then, $\widetilde{\sigma} = ((\widetilde{\sigma}_0, \mathsf{vk}_{\mathsf{u}^*_1}, \mathsf{vk}_{\mathsf{sib}^*_1}), \ldots, (\widetilde{\sigma}_{\mathsf{tlen}-1}, \mathsf{vk}_{\mathsf{u}^*_{\mathsf{tlen}}}, \mathsf{vk}_{\mathsf{sib}^*_{\mathsf{tlen}}}), \widetilde{\sigma}_{\mathsf{tlen}})$.
2. If $t = t^*$ and $x = x^*$, output $\widetilde{\sigma}$.
3. Else if $t = t^*$ and $x \neq x^*$, output $\perp$.

4. Else, compute the signature as follows:

   - Let the nodes on the path (excluding root node) from root to t be $\mathsf{u}_1, \ldots, \mathsf{u}_{\mathsf{tlen}}$ and let their siblings be $\mathsf{sib}_1, \ldots, \mathsf{sib}_{\mathsf{tlen}}$. Suppose that the bit description of t and t^* match in the first $z \in \{0, \ldots, \mathsf{tlen}-1\}$ indices, that is, for $j \in [z]$, $\mathsf{u}_j = \mathsf{u}_j^*$ and $\mathsf{sib}_j = \mathsf{sib}_j^*$. Further, $\mathsf{u}_{z+1} = \mathsf{sib}_{z+1}^*$ and $\mathsf{sib}_{z+1} = \mathsf{u}_{z+1}^*$.
   - For all $j \in [z+1]$, choose $\mathsf{vk}_{\mathsf{sib}_j}$ from $\widetilde{\mathsf{sk}}$. If $z \neq \mathsf{tlen}-1$, then, for all $j \in [\mathsf{tlen}] \setminus [z+1]$: compute $(\mathsf{sk}_{\mathsf{sib}_j}, \mathsf{vk}_{\mathsf{sib}_j}) = \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)}; \mathsf{PPRF}.\mathbf{PEval}(K^*, \mathsf{sib}_j))$.
   - For all $j \in [z]$, choose $\mathsf{vk}_{\mathsf{u}_j}$ from $\widetilde{\mathsf{sk}}$. For all $j = [\mathsf{tlen}] \setminus [z]$: compute $(\mathsf{sk}_{\mathsf{u}_j}, \mathsf{vk}_{\mathsf{u}_j}) = \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)}; \mathsf{PPRF}.\mathbf{PEval}(K^*, \mathsf{u}_j))$.
   - Compute $\alpha_0, \ldots, \alpha_{\mathsf{tlen}}$ as described in step 4 of **Sign** algorithm.
   - Compute the signature on (t, x) and the signatures on the verification keys of the nodes and their siblings on the path (excluding root node) from root node to t as follows.

$$\text{for all } j \in \{z+1, \ldots, \mathsf{tlen}\}\text{: } \sigma_j = \Sigma.\mathbf{Sign}(\mathsf{sk}_{\mathsf{u}_j}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}_j, \alpha_j))$$
$$\text{for all } j \in \{0, \ldots, z\}\text{: } \sigma_j = \widetilde{\sigma}_j$$

   - Output $\sigma = ((\sigma_0, \mathsf{vk}_{\mathsf{u}_1}, \mathsf{vk}_{\mathsf{sib}_1}), \ldots, (\sigma_{\mathsf{tlen}-1}, \mathsf{vk}_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{vk}_{\mathsf{sib}_{\mathsf{tlen}}}), (\sigma_{\mathsf{tlen}}, t, x))$.

Key and signature sizes. In the above algorithm, the size of various keys and signatures are as follows:

- $|\mathsf{sk}| = |\mathsf{sk}_{\mathsf{u}_0}| + |K| = \mathsf{poly}(\lambda, \mathsf{tlen})$ for some fixed polynomial poly.
- $|\mathsf{vk}| = |\mathsf{vk}^*| = \ell_{\mathsf{vk}}(\lambda)$.
- $|\mathsf{pp}| = |\mathsf{pp}^*| = (\mathsf{tlen}+1) \cdot \mathsf{poly}(\lambda)$ for some fixed polynomial poly.
- $|\sigma| = |\widetilde{\sigma}| = |\sigma^*| = (\mathsf{tlen}+1) \cdot \mathsf{poly}(\lambda)$ for some fixed polynomial poly.
- $|\widetilde{\mathsf{sk}}| = |\mathsf{sk}^*| = \mathsf{len} + \mathsf{poly}(\lambda, \mathsf{tlen})$ for some fixed polynomial poly.

We prove the correctness of the above construction in Appendix C.2. Next, we present the main theorem statement of security of the above construction. We prove this theorem in Appendix C.3.

Theorem C.1. *Suppose that* PPRF *is a secure puncturable PRF,* Σ *is a secure SPB signature scheme, and* H *is a secure SPB hash function. Then, the construction in Appendix C.1 is a secure SSU signature scheme (See Definition 5.1).*

We know how to construct puncturable PRFs from one-way functions [GGM86, BW13, BGI14, KPTZ13], SPB signatures from one-way functions [GWZ22], and SPB hash function from indistinguishability obfuscation or leveled fully homomorphic encryption [GWZ22]. Plugging in these instantiations, we obtain the following corollary.

Corollary C.2 (Restatement of Theorem 2.1)**.** *Assuming the existence of one-way functions and indistinguishability obfuscation, or assuming leveled fully homomorphic encryption, there exists a somewhere statistically unforgeable signature scheme.*

C.2 Correctness of Construction

For proving correctness, we need to show two things and we do them below.

First correctness requirement: we need to show that for all $\lambda, \mathsf{len}, \mathsf{tlen} \in \mathbb{N}$, $t \in \{0,1\}^{\mathsf{tlen}}, x \in \{0,1\}^{\mathsf{len}}$,

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

Recall that **Vf** outputs 1 if all the following checks pass.

$$\text{for all } j \in \{0,\ldots,\mathsf{tlen}\}\text{: } \Sigma.\mathbf{Vf}(\mathsf{vk}_{\mathsf{u}_j}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}_j,\alpha_j),\sigma_j)=1$$

If σ is honestly computed using the **Sign** algorithm, then, we have that

$$\text{for all } j \in \{0,\ldots,\mathsf{tlen}\}\text{: } \sigma_j = \Sigma.\mathbf{Sign}(\mathsf{sk}_{\mathsf{u}_j}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}_j,\alpha_j)).$$

As the $\mathsf{H}.\mathbf{Hash}$ algorithm of SPB hash function H is deterministic, therefore the values $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}_j,\alpha_j)$ computed by **Sign** and **Vf** are the same. Then, all the verification checks will passs if the SPB signature scheme Σ satisfies correctness.

Second correctness requirement: we need to show that 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 the case that either ($t = t^*$ and $x = x^*$) or ($t \neq t^*$), then the following holds true:

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

For the case ($t = t^*$ and $x = x^*$), observe that $\mathbf{Sign}(\mathsf{pp},\mathsf{sk},t,x)$ computes $\sigma = ((\sigma_0,\mathsf{vk}_{\mathsf{u}_1},\mathsf{vk}_{\mathsf{sib}_1}),\ldots,(\sigma_{\mathsf{tlen}-1},\mathsf{vk}_{\mathsf{u}_{\mathsf{tlen}}},\mathsf{vk}_{\mathsf{sib}_{\mathsf{tlen}}}),(\sigma_{\mathsf{tlen}},t,x))$. On the other hand, $\mathbf{PSign}(\mathsf{pp},\widetilde{\mathsf{sk}},t,x)$ computes $\widetilde{\sigma} = ((\widetilde{\sigma}_0,\mathsf{vk}_{\mathsf{u}_1},\mathsf{vk}_{\mathsf{sib}_1}),\ldots,(\widetilde{\sigma}_{\mathsf{tlen}-1},\mathsf{vk}_{\mathsf{u}_{\mathsf{tlen}}},\mathsf{vk}_{\mathsf{sib}_{\mathsf{tlen}}}),\widetilde{\sigma}_{\mathsf{tlen}})$ from $\widetilde{\mathsf{sk}}$. Notice that $\widetilde{\sigma}$ is computed by **PuncturedSetup** in exactly the same as σ by the **Sign** algorithm. Thus, the second correctness requirement is satisfied in this case.

For the case ($t \neq t^*$), let the nodes on the path (excluding root node) from root to t be $\mathsf{u}_1,\ldots,\mathsf{u}_{\mathsf{tlen}}$ and let their siblings be $\mathsf{sib}_1,\ldots,\mathsf{sib}_{\mathsf{tlen}}$. Let t and t^* have the same bit description upto first z bits for some $z \in \{0,1,\ldots,\mathsf{tlen}-1\}$.

- For $j \in [z+1]$: **PSign** chooses the verification key $\mathsf{vk}_{\mathsf{sib}_j}$ of the sibling sib_j from $\widetilde{\mathsf{sk}}$ but **Sign** computes it on the fly. By the same argument as in the case of ($t = t^*$ and $x = x^*$), these verification keys are nevertheless the same.

- For $j \in [\mathsf{tlen}] \setminus [z+1]$: **PSign** computes the keys $(\mathsf{sk}_{\mathsf{sib}_j},\mathsf{vk}_{\mathsf{sib}_j})$ of the sibling sib_j using $\mathsf{PPRF}.\mathbf{PEval}(K^*,\mathsf{sib}_j)$ as the random tape for $\Sigma.\mathbf{Gen}$. But, **Sign** computes it using $\mathsf{PPRF}.\mathbf{Eval}(K,\mathsf{sib}_j)$ as the random tape for $\Sigma.\mathbf{Gen}$. But note that K^* is not punctured on any of these sibling nodes. By correctness of puncturable PRFs, it follows that $\mathsf{PPRF}.\mathbf{Eval}(K,\mathsf{sib}_j) = \mathsf{PPRF}.\mathbf{PEval}(K^*,\mathsf{sib}_j)$. Therefore, the random tape used by **Sign** and **PSign** for computing these keys is the same, and hence, the keys of these siblings are also the same.

- For $j \in [z]$: **PSign** chooses $\mathsf{vk}_{\mathsf{u}_j}$ from $\widetilde{\mathsf{sk}}$, but **Sign** computes them on the fly. By the same argument as in the case ($t = t^*$ and $x = x^*$), these values are nevertheless the same.

- For $j \in [\mathsf{tlen}] \setminus [z]$: **PSign** computes the keys $(\mathsf{sk}_{\mathsf{u}_j}, \mathsf{vk}_{\mathsf{u}_j})$ of the node u_j using $\mathsf{PPRF}.\mathbf{Eval}(K^*, \mathsf{u}_j)$ as the random tape for $\Sigma.\mathbf{Gen}$. But, **Sign** computes it using $\mathsf{PPRF}.\mathbf{PEval}(K, \mathsf{u}_j)$ as the random tape for $\Sigma.\mathbf{Gen}$. But note that K^* is not punctured on any of these nodes. By correctness of puncturable PRFs, it follows that $\mathsf{PPRF}.\mathbf{Eval}(K, \mathsf{u}_j) = \mathsf{PPRF}.\mathbf{PEval}(K^*, \mathsf{u}_j)$. Therefore, the random tape used by **Sign** and **PSign** for computing these keys is the same, and hence, the keys of these nodes are also the same.
- Observe then that **Sign** and **PSign** compute $\sigma_{\mathsf{tlen}}, \sigma_{\mathsf{tlen}-1}, \ldots, \sigma_{z+1}$ exactly the same way as **Sign** and **PSign** use the same the hash key and signing key for computing each of these signatures. But $\sigma_z, \ldots, \sigma_0$ are computed differently by **Sign** and **PSign**. **PSign** chooses these from $\widetilde{\mathsf{sk}}$, but **Sign** computes them on the fly. By the same argument as in the case ($t = t^*$ and $x = x^*$), these values are nevertheless the same.

Thus, the second correctness requirement is satisfied in this case as well.

C.3 Proof of Security

In this section, we give the formal proof of security for our SSU signature scheme.

Theorem C.3 (Restatement of Theorem C.1)**.** *Suppose that* PPRF *is a secure puncturable PRF,* Σ *is a secure SPB signature scheme, and* H *is a secure SPB hash function. Then, the construction in Appendix C.1 is a secure SSU signature scheme (See Definition 5.1).*

To prove the above theorem, we need to prove multiple properties about the construction. We do so in Lemmas C.4, C.5 and C.11.

Lemma C.4. *The construction in Appendix C.1 satisfies identical distribution of normal keys output by* **Setup** *and* **PuncturedSetup** *(See Definition 5.1).*

Proof. We need to show that for any len and tlen that are polynomially bounded by λ, any $t^* \in \{0,1\}^{\mathsf{tlen}}$, any $x^* \in \{0,1\}^{\mathsf{len}}$, we have the following where $\equiv$ denotes identical distribution:

$$\begin{aligned}
&\{(\mathsf{sk}, \mathsf{vk}, \mathsf{pp}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len}) : \text{ output } (\mathsf{sk}, \mathsf{vk}, \mathsf{pp})\} \\
\equiv &\{(\mathsf{sk}, \widetilde{\mathsf{sk}}, \mathsf{vk}, \mathsf{pp}) \leftarrow \mathbf{PuncturedSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*) : \text{ output } (\mathsf{sk}, \mathsf{vk}, \mathsf{pp})\}
\end{aligned}$$

Observe that $\mathsf{sk}, \mathsf{vk}, \mathsf{pp}$ are computed by **PuncturedSetup** in steps 1 to 4. This is the same as steps 1 to 4 of **Setup**. Hence, $(\mathsf{sk}, \mathsf{vk}, \mathsf{pp})$ computed by **PuncturedSetup** and **Setup** are identically distributed. □

Lemma C.5. *Suppose that* PPRF *is a secure puncturable PRF. Suppose that* Σ *is a SPB signature scheme that satisfies computational indistinguishability of normal and binding keys (See Appendix B.4). Suppose that* H *is a SPB hash function that satisfies (i) identical distribution of normal hash keys and (ii)computational indistinguishability of normal and binding hash keys (See Appendix B.3). Then, the above construction satisfies computational indistinguishability of punctured and binding setup (See Definition 5.1).*

Proof. We need to prove that for any len and tlen that are polynomially bounded by λ, any $t^* \in \{0,1\}^{\mathsf{tlen}}$, any $x^* \in \{0,1\}^{\mathsf{len}}$, we have the following where $\approx$ denotes computational indistinguishability:

$$\begin{aligned}
&\{(\mathsf{sk}, \widetilde{\mathsf{sk}}, \mathsf{vk}, \mathsf{pp}) \leftarrow \mathbf{PuncturedSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*) : \text{ output } (\widetilde{\mathsf{sk}}, \mathsf{vk}, \mathsf{pp})\} \\
\approx &\{(\mathsf{sk}^*, \mathsf{vk}^*, \mathsf{pp}^*) \leftarrow \mathbf{BindingSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*) : \text{ output } (\mathsf{sk}^*, \mathsf{vk}^*, \mathsf{pp}^*)\}
\end{aligned}$$

Hencforth, we will denote the first distribution as $\mathcal{D}$ and the second distribution as $\mathcal{D}^*$. Next, we spell out the complete details of these two distributions.

Distribution $\mathcal{D}$: Recall that the values computed by **PuncturedSetup** are $\mathsf{vk} = \mathsf{vk}_{\mathsf{u}_0}$, $\mathsf{pp} = \{\mathsf{hk}_j\}_{j\in\{0,\ldots,\mathsf{tlen}\}}$, and $\widetilde{\mathsf{sk}} = (\widetilde{\sigma}, t^*, x^*, K^*)$, where $\widetilde{\sigma} = ((\widetilde{\sigma}_0, \mathsf{vk}_{\mathsf{u}_1}, \mathsf{vk}_{\mathsf{sib}_1}), \ldots, (\widetilde{\sigma}_{\mathsf{tlen}-1}, \mathsf{vk}_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{vk}_{\mathsf{sib}_{\mathsf{tlen}}}), \widetilde{\sigma}_{\mathsf{tlen}})$. Plugging in these values, $\mathcal{D}$ is the following.

$$\{(\widetilde{\sigma}_0, \mathsf{vk}_{\mathsf{u}_1}, \mathsf{vk}_{\mathsf{sib}_1}), \ldots, (\widetilde{\sigma}_{\mathsf{tlen}-1}, \mathsf{vk}_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{vk}_{\mathsf{sib}_{\mathsf{tlen}}}), \widetilde{\sigma}_{\mathsf{tlen}}, t^*, x^*, K^*, \mathsf{vk}_{\mathsf{u}_0}, \{\mathsf{hk}_j\}_{j\in\{0,\ldots,\mathsf{tlen}\}}\}$$

Rearranging some of the terms for ease of understanding of the changes later, we can rewrite the distribution as follows.

$$\boxed{\mathcal{D}: \ \{(\mathsf{vk}_{\mathsf{u}_0}, \widetilde{\sigma}_0, \mathsf{hk}_0), \ldots, (\mathsf{vk}_{\mathsf{u}_{\mathsf{tlen}}}, \widetilde{\sigma}_{\mathsf{tlen}}, \mathsf{hk}_{\mathsf{tlen}}), \{\mathsf{vk}_{\mathsf{sib}_j}\}_{j\in[\mathsf{tlen}]}, t^*, x^*, K^*\}}$$

Here, for all $j \in \{0, \ldots, \mathsf{tlen}\}$, hk_j is computed using $\mathsf{H}.\mathbf{Gen}$. Further, $\mathsf{vk}_{\mathsf{u}_0}$ is computed using $\Sigma.\mathbf{Gen}$ with its random tape containing uniformly random coins. For all $j \in [\mathsf{tlen}]$, $\mathsf{vk}_{\mathsf{u}_j}$ is computed using $\Sigma.\mathbf{Gen}$ with its random tape $\mathsf{PPRF}.\mathbf{Eval}(K, \mathsf{u}_j)$.

Distribution $\mathcal{D}^*$: Recall that the values computed by **BindingSetup** are $\mathsf{vk}^* = \mathsf{vk}^*_{\mathsf{u}_0}$, $\mathsf{pp}^* = \{\mathsf{hk}^*_j\}_{j\in\{0,\ldots,\mathsf{tlen}\}}$, and $\mathsf{sk}^* = (\sigma^*, t^*, x^*, K^*)$, where $\sigma^* = ((\sigma^*_0, \mathsf{vk}^*_{\mathsf{u}_1}, \mathsf{vk}_{\mathsf{sib}_1}), \ldots, (\sigma^*_{\mathsf{tlen}-1}, \mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{vk}_{\mathsf{sib}_{\mathsf{tlen}}}), \sigma^*_{\mathsf{tlen}})$. Plugging in these values, $\mathcal{D}^*$ is the following.

$$\{(\sigma^*_0, \mathsf{vk}^*_{\mathsf{u}_1}, \mathsf{vk}_{\mathsf{sib}_1}), \ldots, (\sigma^*_{\mathsf{tlen}-1}, \mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{vk}_{\mathsf{sib}_{\mathsf{tlen}}}), \sigma^*_{\mathsf{tlen}}, t^*, x^*, K^*, \mathsf{vk}^*_{\mathsf{u}_0}, \{\mathsf{hk}^*_j\}_{j\in\{0,\ldots,\mathsf{tlen}\}}\}$$

Rearranging some of the terms as above, we can rewrite $\mathcal{D}^*$ as follows.

$$\boxed{\mathcal{D}^*: \{(\mathsf{vk}^*_{\mathsf{u}_0}, \sigma^*_0, \mathsf{hk}^*_0), \ldots, (\mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}, \sigma^*_{\mathsf{tlen}}, \mathsf{hk}^*_{\mathsf{tlen}}), \{\mathsf{vk}_{\mathsf{sib}_j}\}_{j\in[\mathsf{tlen}]}, t^*, x^*, K^*\}}$$

$\mathcal{D}^*$ differs from $\mathcal{D}$ in the terms highlighted in blue. In more detail, these are as follows. For all $j \in \{0, \ldots, \mathsf{tlen}\}$, hk^*_j is binding on the value α^*_j and is computed using $\mathsf{H}.\mathbf{GenBind}$. Further, $\mathsf{vk}^*_{\mathsf{u}_0}$ is binding on the value $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_0, \alpha^*_0)$ and is computed using $\Sigma.\mathbf{GenBind}$ with its random tape containing uniformly random coins. For all $j \in [\mathsf{tlen}]$, $\mathsf{vk}^*_{\mathsf{u}_j}$ is binding on the value $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_j, \alpha^*_j)$ and is computed using $\Sigma.\mathbf{GenBind}$ with its random tape containing uniformly random coins.

To prove $\mathcal{D} \approx \mathcal{D}^*$, consider the following sequence of hybrid distributions.

Distribution $\mathcal{D}_0$: This is same as $\mathcal{D}$ except that for all $j \in \{0, \ldots, \mathsf{tlen}\}$, the normal hash key hk_j is computed using $\mathsf{H}.\mathbf{GenBind}$ instead of $\mathsf{H}.\mathbf{Gen}$ as follows. If$j \neq \mathsf{tlen}$, $(\mathsf{hk}_j, \mathsf{hk}^*_j) \leftarrow \mathsf{H}.\mathbf{GenBind}(1^\lambda, 1^{2\ell_{\mathsf{vk}}(\lambda)}, \alpha_j)$. If$j = \mathsf{tlen}$, $(\mathsf{hk}_{\mathsf{tlen}}, \mathsf{hk}^*_{\mathsf{tlen}}) \leftarrow \mathsf{H}.\mathbf{GenBind}(1^\lambda, 1^{\mathsf{tlen}+\mathsf{len}}, \alpha_j)$.

Distribution $\mathcal{D}_1$: This is same as $\mathcal{D}_0$ except that for all $j \in [\mathsf{tlen}]$, $\mathsf{vk}_{\mathsf{u}_j}$ are computed using $\Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)})$ with its random tape containing uniformly random coins instead of $\Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)}; \mathsf{PPRF}.\mathbf{Eval}(K, \mathsf{u}_j))$.

Distribution $\mathcal{D}_{2,i}$ for all $i \in \{\mathsf{tlen}+1, \ldots, 0\}$: The distribution is as follows.

$$\left\{\begin{array}{c}(\mathsf{vk}_{\mathsf{u}_0}, \widetilde{\sigma}_0, \mathsf{hk}_0), \ldots, (\mathsf{vk}_{\mathsf{u}_{i-2}}, \widetilde{\sigma}_{i-2}, \mathsf{hk}_{i-2}),\\ (\mathsf{vk}_{\mathsf{u}_{i-1}}, \widetilde{\sigma}_{i-1}, \mathsf{hk}_{i-1}), (\mathsf{vk}^*_{\mathsf{u}_i}, \sigma^*_i, \mathsf{hk}^*_i), \ldots, (\mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}, \sigma^*_{\mathsf{tlen}}, \mathsf{hk}^*_{\mathsf{tlen}}),\\ \{\mathsf{vk}_{\mathsf{sib}_j}\}_{j\in[\mathsf{tlen}]}, t^*, x^*, K^*\end{array}\right\}$$

It differs from distribution $\mathcal{D}_1$ in the terms highlighted in blue. These differing values are computed in the following sequence:

- Let $\alpha^*_{\mathsf{tlen}} = t^*||x^*$.
- Compute $(\mathsf{hk}_{\mathsf{tlen}}, \mathsf{hk}^*_{\mathsf{tlen}}) \leftarrow \mathsf{H}.\mathbf{GenBind}(1^\lambda, 1^{\mathsf{len}+\mathsf{tlen}}, \alpha^*_{\mathsf{tlen}})$.
- Compute $(\mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}, \sigma^*_{\mathsf{tlen}}) = \Sigma.\mathbf{GenBind}(1^\lambda, 1^{\ell_h(\lambda)}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_{\mathsf{tlen}}, \alpha^*_{\mathsf{tlen}}))$.
- For $j = \mathsf{tlen}-1, \ldots, i$ in decreasing order:
  - Let $\alpha^*_j = \begin{cases} \mathsf{vk}^*_{\mathsf{u}_{j+1}}||\mathsf{vk}_{\mathsf{sib}_{j+1}} & \text{if } j^{th}\text{-bit of } t \text{ is } 0 \\ \mathsf{vk}_{\mathsf{sib}_{j+1}}||\mathsf{vk}^*_{\mathsf{u}_{j+1}} & \text{if } j^{th}\text{-bit of } t \text{ is } 1 \end{cases}$.
  - Compute $(\mathsf{hk}_j, \mathsf{hk}^*_j) \leftarrow \mathsf{H}.\mathbf{GenBind}(1^\lambda, 1^{2\ell_{\mathsf{vk}}(\lambda)}, \alpha^*_j)$.
  - Compute $(\mathsf{vk}^*_{\mathsf{u}_j}, \sigma^*_j) = \Sigma.\mathbf{GenBind}(1^\lambda, 1^{\ell_h(\lambda)}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_j, \alpha^*_j))$.
- Compute $\widetilde{\sigma}_{i-1} = \Sigma.\mathbf{Sign}(\mathsf{sk}_{\mathsf{u}_{i-1}}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}_{i-1}, \alpha^*_{i-1}))$.

To complete the proof, we will argue that

$$\mathcal{D} \equiv \mathcal{D}_0 \approx \mathcal{D}_1 \equiv \mathcal{D}_{2,\mathsf{tlen}+1} \approx \mathcal{D}_{2,\mathsf{tlen}} \approx \ldots \approx \mathcal{D}_{2,0} \equiv \mathcal{D}^*.$$

$\mathcal{D}_1 \equiv \mathcal{D}_{2,\mathsf{tlen}+1}$**:** Observe that the distribution in $\mathcal{D}_{2,\mathsf{tlen}+1}$ is

$$\left\{ \begin{matrix} (\mathsf{vk}_{\mathsf{u}_0}, \widetilde{\sigma}_0, \mathsf{hk}_0), \ldots, (\mathsf{vk}_{\mathsf{u}_{\mathsf{tlen}-1}}, \widetilde{\sigma}_{\mathsf{tlen}-1}, \mathsf{hk}_{\mathsf{tlen}-1}), \\ (\mathsf{vk}_{\mathsf{u}_{\mathsf{tlen}}}, \widetilde{\sigma}_{\mathsf{tlen}}, \mathsf{hk}_{\mathsf{tlen}}), \{\mathsf{vk}_{\mathsf{sib}_j}\}_{j\in[\mathsf{tlen}]}, t^*, x^*, K^* \end{matrix} \right\}.$$

It differs from $\mathcal{D}_1$ in only one term $\widetilde{\sigma}_{\mathsf{tlen}}$. In $\mathcal{D}_{2,\mathsf{tlen}+1}$, the signature is $\widetilde{\sigma}_{\mathsf{tlen}} = \Sigma.\mathbf{Sign}(\mathsf{sk}_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}_{\mathsf{tlen}}, \alpha^*_{\mathsf{tlen}}))$. Whereas, in D_1, the signature is $\widetilde{\sigma}_{\mathsf{tlen}} = \Sigma.\mathbf{Sign}(\mathsf{sk}_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}_{\mathsf{tlen}}, \alpha_{\mathsf{tlen}}))$. As $\alpha^*_{\mathsf{tlen}} = \alpha_{\mathsf{tlen}} = t^*||x^*$, therefore, we get that $\widetilde{\sigma}_{\mathsf{tlen}}$ in the two distributions are identically distributed. Hence, $\mathcal{D}_1$ and $\mathcal{D}_{2,\mathsf{tlen}+1}$ are identically distributed.

$\mathcal{D}_{2,0} \equiv \mathcal{D}^*$**:** Observe that the distribution in $\mathcal{D}_{2,0}$ is

$$\left\{(\mathsf{vk}^*_{\mathsf{u}_0}, \sigma^*_0, \mathsf{hk}^*_0), \ldots, (\mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}, \sigma^*_{\mathsf{tlen}}, \mathsf{hk}^*_{\mathsf{tlen}}), \{\mathsf{vk}_{\mathsf{sib}_j}\}_{j\in[\mathsf{tlen}]}, t^*, x^*, K^*\right\}.$$

This description is exactly same as that of $\mathcal{D}^*$. It follows then that $\mathcal{D}_{2,0}$ and $\mathcal{D}^*$ are identically distributed.

In Lemma C.6, we show that $\mathcal{D} \equiv \mathcal{D}_0$. In Lemma C.7, we show that $\mathcal{D}_0 \approx \mathcal{D}_1$. In Lemma C.8, we show that $\mathcal{D}_{2,i+1} \approx \mathcal{D}_{2,i}$ for all $i \in \{\mathsf{tlen}, \ldots, 0\}$. This completes the proof.

□

Lemma C.6. *Suppose that* H *is a SPB hash function that satisfies identical distribution of normal hash keys, then,* $\mathcal{D} \equiv \mathcal{D}_0$.

Proof. The equivalence can be proven via a sequence of $\mathsf{tlen}+1$ hybrid distributions $\mathcal{D}_{0,i}$ for all $i \in \{0, \ldots, \mathsf{tlen}\}$. In $\mathcal{D}_{0,i}$, for all $j \le i$, hk_j will be computed as normal hash keys output by $\mathsf{H}.\mathbf{GenBind}(1^\lambda, 1^{2\ell_{\mathsf{vk}}(\lambda)}, \alpha_j)$ (or $\mathsf{H}.\mathbf{GenBind}(1^\lambda, 1^{\mathsf{tlen}+\mathsf{len}}, \alpha_{\mathsf{tlen}})$ in case $j = \mathsf{tlen}$) and all other hash keys will be computed as normal hash keys output by $\mathsf{H}.\mathbf{Gen}(1^\lambda, 1^{2\ell_{\mathsf{vk}}(\lambda)})$ (or $\mathsf{H}.\mathbf{Gen}(1^\lambda, 1^{\mathsf{tlen}+\mathsf{len}})$ in case $j = \mathsf{tlen}$). Then, we can show that $\mathcal{D} \equiv \mathcal{D}_{0,0} \equiv \mathcal{D}_{0,1} \equiv \ldots \equiv \mathcal{D}_{0,\mathsf{tlen}}$. Notice that here every adjacent distribution differs only in how one hash key is chosen and the equivalence can then be argued by the identical distribution of normal hash keys of the SPB hash function H. Lastly, $\mathcal{D}_{0,\mathsf{tlen}}$ is same as $\mathcal{D}_0$. This completes the proof. □

Lemma C.7. *Suppose that* PPRF *is a secure puncturable PRF, then,* $\mathcal{D}_0 \approx \mathcal{D}_1$.

Proof. Suppose that the path (excluding root node) from the root to t^* is $\mathsf{u}_1, \ldots, \mathsf{u}_{\mathsf{tlen}}$. Then, observe that both the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ contain the punctured PRF key K^* punctured at strings $\mathsf{u}_1, \ldots, \mathsf{u}_{\mathsf{tlen}}$. For all $i \in [\mathsf{tlen}]$, $\mathsf{vk}_{\mathsf{u}_i}$ are computed using $\Sigma.\mathbf{Gen}$ with its random tape differing in $\mathcal{D}_0$ and $\mathcal{D}_1$ as follows:

- In $\mathcal{D}_0$, the random tape is $\mathsf{PPRF}.\mathbf{Eval}(K, \mathsf{u}_i)$.
- In $\mathcal{D}_1$, the random tape contains uniformly random coins.

From security of puncturable PRFs, we know that the PRF evaluations on punctured points look indistinguishable from random. Therfore, if there exists a p.p.t. adversary $\mathcal{A}$ that can distinguish between $\mathcal{D}_0$ and $\mathcal{D}_1$ with non-neglegible advantage, then, a straightforward reduction $\mathcal{B}$ can be constructed that can break the PPRF security.

□

Lemma C.8. *Suppose that* Σ *is a SPB signature scheme that satisfies computational indistinguishability of normal and binding verification keys (See Appendix B.4). Suppose that* H *is a SPB hash function that satisfies computational indistinguishability of normal and binding hash keys. Then,* $\mathcal{D}_{2,i+1} \approx \mathcal{D}_{2,i}$ *for all* $i \in \{\mathsf{tlen}, \ldots, 0\}$.

Proof. The difference between $\mathcal{D}_{2,i+1}$ and $\mathcal{D}_{2,i}$ is as follows:

- $\mathcal{D}_{2,i+1}$ contains $(\mathsf{vk}_{\mathsf{u}_i}, \widetilde{\sigma}_i, \mathsf{hk}_i)$ where $\mathsf{vk}_{\mathsf{u}_i}, \mathsf{hk}_i$ are non-binding, $\widetilde{\sigma}_i$ is signature on $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}_i, \alpha_i^*)$. But $\mathcal{D}_{2,i}$ contains $(\mathsf{vk}^*_{\mathsf{u}_i}, \sigma_i^*, \mathsf{hk}_i^*)$, where hk_i^* is binding on the value α_i^*, $\mathsf{vk}^*_{\mathsf{u}_i}$ is binding on the value $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}_i^*, \alpha_i^*)$ and σ_i^* is a signature on $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}_i^*, \alpha_i^*)$.
- $\mathcal{D}_{2,i+1}$ contains signature $\widetilde{\sigma}_{i-1}$ on $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}_{i-1}, \alpha_{i-1})$ and $\mathcal{D}_{2,i}$ contains signature $\widetilde{\sigma}_{i-1}$ on $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}_i, \alpha^*_{i-1})$.

Essentially the two distributions are different in the i^{th} SPB hash key, the i^{th} SPB verification key and the signatures associated with them. To prove computational indistinguishability, we introduce an intermediate hybrid $\mathcal{D}'_{2,i}$ as follows.

Distribution $\mathcal{D}'_{2,i}$ **for all** $i \in \{0, \ldots, \mathsf{tlen}\}$**:** This is same as distribution $\mathcal{D}_{2,i+1}$ except that the normal hash key hk_i is replaced with the binding hash key hk_i^*. As a consequence, $\widetilde{\sigma}_i$ is now a signature on $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}_i^*, \alpha_i^*)$ instead of $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}_i, \alpha_i^*)$.

In Claim C.9, we show that $\mathcal{D}_{2,i+1} \approx \mathcal{D}'_{2,i}$. In Claim C.10, we show that $\mathcal{D}'_{2,i} \approx \mathcal{D}_{2,i}$. This completes the proof. □

Claim C.9. *Suppose that* H *is a SPB hash function that satisfies computational indistinguishability of normal and binding hash keys. Then, for all* $i \in \{0, \ldots, \mathsf{tlen}\}$, $\mathcal{D}_{2,i+1} \approx \mathcal{D}'_{2,i}$.

Proof. As noted in the description of $\mathcal{D}'_{2,i}$, it essentially differs from $\mathcal{D}_{2,i+1}$ in the i^{th} SPB hash key and the signature associated with it.

Suppose that there exists a p.p.t. adversary $\mathcal{A}$ that can distinguish between $\mathcal{D}_{2,i+1}$ and $\mathcal{D}'_{2,i}$ with non-neglegible advantage, then, we show a reduction $\mathcal{B}$ that can break the computational indistinguishability of normal and binding hash keys of the SPB hash function H as follows.

$\mathcal{B}$ receives (t^*, x^*) as inputs from $\mathcal{A}$. $\mathcal{B}$ computes the punctured PRF key K^* as in **BindingSetup** algorithm. Then, $\mathcal{B}$ computes the following:

- For all $j \in [\mathsf{tlen}]$, compute $(\mathsf{sk}_{\mathsf{sib}_j}, \mathsf{vk}_{\mathsf{sib}_j}) \leftarrow \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)})$.

- For all $j = \mathsf{tlen}, \ldots, i+1$ in decreasing order, compute: $(\mathsf{hk}_j, \mathsf{hk}^*_j) \leftarrow \mathsf{H}.\mathbf{GenBind}(1^\lambda, 1^{2\ell_{\mathsf{vk}}(\lambda)}, \alpha^*_j)$ (or $(\mathsf{hk}_{\mathsf{tlen}}, \mathsf{hk}^*_{\mathsf{tlen}}) \leftarrow \mathsf{H}.\mathbf{GenBind}(1^\lambda, 1^{\mathsf{tlen}+\mathsf{len}}, \alpha^*_j)$ if $j = \mathsf{tlen}$) , and $(\mathsf{vk}^*_{\mathsf{u}_j}, \sigma^*_j) \leftarrow \Sigma.\mathbf{GenBind}(1^\lambda, 1^{\ell_h(\lambda)}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_j, \alpha^*_j))$, where α^*_j is as defined in step 4 of **BindingSetup** algorithm.

- For $j = i$, $\mathcal{B}$ sends a challenge $(1^{2\ell_{\mathsf{vk}}(\lambda)}, \alpha^*_i)$ (or $(1^{\mathsf{tlen}+\mathsf{len}}, \alpha^*_i)$ if $i = \mathsf{tlen}$) to the SPB hash function challenger $\mathcal{C}$, where α^*_i is as defined in step 4 of **BindingSetup** algorithm. $\mathcal{C}$ computes $(\mathsf{hk}_i, \mathsf{hk}^*_i) \leftarrow \mathsf{H}.\mathbf{GenBind}(1^\lambda, 1^{2\ell_{\mathsf{vk}}(\lambda)}, \alpha^*_i)$ (or $(\mathsf{hk}_i, \mathsf{hk}^*_i) \leftarrow \mathsf{H}.\mathbf{GenBind}(1^\lambda, 1^{\mathsf{tlen}+\mathsf{len}}, \alpha^*_i)$ if $i = \mathsf{tlen}$). $\mathcal{C}$ flips a uniformly random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{C}$ sets $\mathsf{hk}' = \mathsf{hk}_i$. If $b = 1$, $\mathcal{C}$ sets $\mathsf{hk}' = \mathsf{hk}^*_i$. $\mathcal{C}$ sends hk' to $\mathcal{B}$. $\mathcal{B}$ computes $(\mathsf{sk}_{\mathsf{u}_i}, \mathsf{vk}_{\mathsf{u}_i}) \leftarrow \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)})$ and $\widetilde{\sigma}_i = \Sigma.\mathbf{Sign}(\mathsf{sk}_{\mathsf{u}_i}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}', \alpha^*_i))$.

- For all $j = i-1, \ldots, 0$ in decreasing order, compute: $\mathsf{hk}_j \leftarrow \mathsf{H}.\mathbf{Gen}(1^\lambda, 1^{2\ell_{\mathsf{vk}}(\lambda)})$, $(\mathsf{sk}_{\mathsf{u}_j}, \mathsf{vk}_{\mathsf{u}_j}) \leftarrow \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)})$ and $\widetilde{\sigma}_j = \Sigma.\mathbf{Sign}(\mathsf{sk}_{\mathsf{u}_j}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}_j, \alpha_j))$. Here, α_j is as defined in step 4 of **Sign** algorithm.

Finally, $\mathcal{B}$ sends the following to $\mathcal{A}$.

$$\left(\begin{array}{c} (\mathsf{vk}_{\mathsf{u}_0}, \widetilde{\sigma}_0, \mathsf{hk}_0), \ldots, (\mathsf{vk}_{\mathsf{u}_{i-2}}, \widetilde{\sigma}_{i-2}, \mathsf{hk}_{i-2}), \\ (\mathsf{vk}_{\mathsf{u}_{i-1}}, \widetilde{\sigma}_{i-1}, \mathsf{hk}_{i-1}), (\mathsf{vk}_{\mathsf{u}_i}, \widetilde{\sigma}_i, \mathsf{hk}'), (\mathsf{vk}^*_{\mathsf{u}_{i+1}}, \sigma^*_{i+1}, \mathsf{hk}^*_{i+1}), \ldots, (\mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}, \sigma^*_{\mathsf{tlen}}, \mathsf{hk}^*_{\mathsf{tlen}}), \\ \{\mathsf{vk}_{\mathsf{sib}_j}\}_{j \in [\mathsf{tlen}]}, t^*, x^*, K^* \end{array}\right)$$

$\mathcal{A}$ outputs a guess $b' \in \{0,1\}$ to $\mathcal{B}$ and $\mathcal{B}$ forwards b' to its challenger $\mathcal{C}$.

Observe that when $\mathcal{C}$ chooses $b = 0$, then, $\mathcal{B}$ perfectly simulates $\mathcal{D}_{2,i+1}$ to $\mathcal{A}$. And when $\mathcal{C}$ chooses $b = 1$, then, $\mathcal{B}$ perfectly simulates $\mathcal{D}'_{2,i}$ to $\mathcal{A}$. Therefore, if $\mathcal{A}$ distinguish between its two view with non-neglegible advantage, then, $\mathcal{B}$ can distinguish between its two views in its game with $\mathcal{C}$ with the same non-neglegible advantage and thus break the computational indistinguishability of normal and binding hash keys of the SPB hash function H.

□

Claim C.10. *Suppose that Σ is a SPB signature scheme that satisfies computational indistinguishability of normal and binding verification keys (See Appendix B.4). Then, for all $i \in \{0, \ldots, \mathsf{tlen}\}$, $\mathcal{D}'_{2,i} \approx \mathcal{D}_{2,i}$.*

Proof. The difference between distributions $\mathcal{D}'_{2,i}$ and $\mathcal{D}_{2,i}$ is as follows.

- $\mathcal{D}'_{2,i}$ contains $(\mathsf{vk}_{\mathsf{u}_i}, \widehat{\sigma}_i, \mathsf{hk}^*_i)$, but $\mathcal{D}_{2,i}$ contains $(\mathsf{vk}^*_{\mathsf{u}_i}, \sigma^*_i, \mathsf{hk}^*_i)$. In both, hk^*_i is binding on the value α^*_i. In $\mathcal{D}'_{2,i}$, $\mathsf{vk}_{\mathsf{u}_i}$ is non-binding, $\widehat{\sigma}_i$ is signature on $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_i, \alpha^*_i)$ computed using $\Sigma.\mathbf{Sign}$. In $\mathcal{D}_{2,i}$, $\mathsf{vk}^*_{\mathsf{u}_i}$ is binding on the value $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_i, \alpha^*_i)$ and σ^*_i is a signature on $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_i, \alpha^*_i)$ computed using $\Sigma.\mathbf{GenBind}$ algorithm.

- $\mathcal{D}_{2,i+1}$ contains signature σ_{i-1} on $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}_{i-1}, \alpha_{i-1})$ and $\mathcal{D}_{2,i}$ contains signature $\widehat{\sigma}_{i-1}$ on $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}_i, \alpha^*_{i-1})$. Recall here that α_{i-1} contains $\mathsf{vk}_{\mathsf{u}_i}$ whereas α^*_{i-1} contains $\mathsf{vk}^*_{\mathsf{u}_i}$.

Essentially the two distributions are different in the i^{th} SPB verification key and the signatures associated with it.

Suppose that there exists a p.p.t. adversary $\mathcal{A}$ that can distinguish between $\mathcal{D}'_{2,i}$ and $\mathcal{D}_{2,i}$ with non-neglegible advantage, then, we show a reduction $\mathcal{B}$ that can break the computational indistinguishability of normal and binding verification keys of the SPB signature scheme Σ as follows.

$\mathcal{B}$ receives (t^*, x^*) as inputs from $\mathcal{A}$. $\mathcal{B}$ computes the punctured PRF key K^* as in **BindingSetup** algorithm. Then, $\mathcal{B}$ computes the following where $\ell_j = 2\ell_{\mathsf{vk}}(\lambda)$ if $j \neq \mathsf{tlen}$ else $\ell_j = \mathsf{tlen} + \mathsf{len}$.

- For all $j \in [\mathsf{tlen}]$, compute $(\mathsf{sk}_{\mathsf{sib}_j}, \mathsf{vk}_{\mathsf{sib}_j}) \leftarrow \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)})$.
- For all $j = \mathsf{tlen}, \ldots, i+1$ in decreasing order, compute: $(\mathsf{hk}_j, \mathsf{hk}^*_j) \leftarrow \mathsf{H}.\mathbf{GenBind}(1^\lambda, 1^{\ell_j}, \alpha^*_j)$ and compute $(\mathsf{vk}^*_{\mathsf{u}_j}, \sigma^*_j) \leftarrow \Sigma.\mathbf{GenBind}(1^\lambda, 1^{\ell_h(\lambda)}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_j, \alpha^*_j))$, where α^*_j is as defined in step 4 of **BindingSetup** algorithm.
- For $j = i$, $\mathcal{B}$ computes $(\mathsf{hk}_i, \mathsf{hk}^*_i) \leftarrow \mathsf{H}.\mathbf{GenBind}(1^\lambda, 1^{\ell_i}, \alpha^*_i)$, where α^*_i is as defined in step 4 of **BindingSetup** algorithm. $\mathcal{B}$ sends a challenge $(1^{\ell_h(\lambda)}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_i, \alpha^*_i))$ to the SPB signature challenger $\mathcal{C}$. $\mathcal{C}$ flips a uniformly random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{C}$ computes $(\mathsf{sk}', \mathsf{vk}') \leftarrow \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)})$, $\sigma' = \Sigma.\mathbf{Sign}(\mathsf{pp}', \mathsf{sk}', \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_i, \alpha^*_i))$. If $b = 1$, $\mathcal{C}$ computes $(\mathsf{vk}', \sigma') \leftarrow \Sigma.\mathbf{GenBind}(1^\lambda, 1^{\ell_h(\lambda)}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_i, \alpha^*_i))$. $\mathcal{C}$ sends (vk', σ') to $\mathcal{B}$. $\mathcal{B}$ sets $\alpha_{i-1} = \mathsf{vk}' || \mathsf{vk}_{\mathsf{sib}_{i-1}}$ if the $i - 1^{th}$-bit of t^* is 0, else $\mathcal{B}$ sets $\alpha_{i-1} = \mathsf{vk}_{\mathsf{sib}_{i-1}} || \mathsf{vk}'$.
- For all $j = i-1, \ldots, 0$ in decreasing order, compute: $\mathsf{hk}_j \leftarrow \mathsf{H}.\mathbf{Gen}(1^\lambda, 1^{2\ell_{\mathsf{vk}}(\lambda)})$, $(\mathsf{sk}_{\mathsf{u}_j}, \mathsf{vk}_{\mathsf{u}_j}) \leftarrow \Sigma.\mathbf{Gen}(1^\lambda, 1^{\ell_h(\lambda)})$ and $\sigma_j = \Sigma.\mathbf{Sign}(\mathsf{sk}_{\mathsf{u}_j}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}_j, \alpha_j))$. Here, α_j is as defined in step 4 of **Sign** algorithm, except that it is as defined in the previous step when $j = i - 1$.

Finally, $\mathcal{B}$ sends the following to $\mathcal{A}$.

$$\left(\begin{array}{c} (\mathsf{vk}_{\mathsf{u}_0}, \sigma_0, \mathsf{hk}_0), \ldots, (\mathsf{vk}_{\mathsf{u}_{i-2}}, \sigma_{i-2}, \mathsf{hk}_{i-2}), \\ (\mathsf{vk}_{\mathsf{u}_{i-1}}, \sigma_{i-1}, \mathsf{hk}_{i-1}), (\mathsf{vk}', \sigma', \mathsf{hk}^*_i), (\mathsf{vk}^*_{\mathsf{u}_{i+1}}, \sigma^*_{i+1}, \mathsf{hk}^*_{i+1}), \ldots, (\mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}, \sigma^*_{\mathsf{tlen}}, \mathsf{hk}^*_{\mathsf{tlen}}), \\ \{\mathsf{vk}_{\mathsf{sib}_j}\}_{j \in [\mathsf{tlen}]}, t^*, x^*, K^* \end{array}\right)$$

$\mathcal{A}$ outputs a guess $b' \in \{0,1\}$ to $\mathcal{B}$ and $\mathcal{B}$ forwards b' to its challenger $\mathcal{C}$.

Observe that when $\mathcal{C}$ chooses $b = 0$, then, $\mathcal{B}$ perfectly simulates $\mathcal{D}'_{2,i}$ to $\mathcal{A}$. And when $\mathcal{C}$ chooses $b = 1$, then, $\mathcal{B}$ perfectly simulates $\mathcal{D}_{2,i}$ to $\mathcal{A}$. Therefore, if $\mathcal{A}$ distinguish between its two view with non-neglegible advantage, then, $\mathcal{B}$ can distinguish between its two views in its game with $\mathcal{C}$ with the same non-neglegible advantage and thus break the computational indistinguishability of normal and binding verification keys of the SPB signatures scheme Σ.

□

Lemma C.11. *Suppose that Σ is a SPB signature scheme satisfying statistical binding. Suppose that H is a SPB hash function satisfying statistical binding. Then, the construction in Appendix C.1 satisfies statistical unforgeability at (t^*, x^*) (See Definition 5.1).*

Proof. Let $(\mathsf{sk}^*, \mathsf{vk}^*, \mathsf{pp}^*) \leftarrow \mathbf{BindingSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}, t^*, x^*)$ and $\sigma^* = \mathbf{PSign}(\mathsf{pp}^*, \mathsf{sk}^*, t^*, x^*)$. Then, $\mathsf{vk}^* = \mathsf{vk}^*_{\mathsf{u}_0}, \mathsf{pp}^* = \{\mathsf{hk}^*_j\}_{j \in \{0,\ldots,\mathsf{tlen}\}}$. Let the nodes on the path (excluding root node) from root to t^* be $\mathsf{u}_1, \ldots, \mathsf{u}_{\mathsf{tlen}}$ and let their siblings be $\mathsf{sib}_1, \ldots, \mathsf{sib}_{\mathsf{tlen}}$. Then, σ^* is as follows.

$$\sigma^* = ((\sigma^*_0, \mathsf{vk}^*_{\mathsf{u}_1}, \mathsf{vk}_{\mathsf{sib}_1}), \ldots, (\sigma^*_{\mathsf{tlen}-1}, \mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{vk}_{\mathsf{sib}_{\mathsf{tlen}}}), \sigma^*_{\mathsf{tlen}}).$$

We need to prove two things and we do them one after the other below.

(i) for $t = t^*, x = x^*$, there does not exist $\sigma' \neq \sigma^*$ such that $\mathbf{Vf}(\mathsf{pp}^*, \mathsf{vk}^*, t^*, x^*, \sigma') = 1$: We prove by contradiction. Suppose such a signature $\sigma' \neq \sigma^*$ exists. Let $\sigma' = ((\sigma'_0, \mathsf{vk}'_{\mathsf{u}_1}, \mathsf{vk}'_{\mathsf{sib}_1}), \ldots, (\sigma'_{\mathsf{tlen}-1}, \mathsf{vk}'_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{vk}'_{\mathsf{sib}_{\mathsf{tlen}}}), \sigma'_{\mathsf{tlen}})$. For all $j \in \{0, \ldots, \mathsf{tlen} - 1\}$, define α'_j with respect to $\mathsf{vk}'_{\mathsf{u}_j}, \mathsf{vk}'_{\mathsf{sib}_j}$ the same way as α^*_j is defined with respect to $\mathsf{vk}^*_{\mathsf{u}_j}, \mathsf{vk}_{\mathsf{sib}_j}$ in step 4 of **BindingSetup** algorithm.

This means,

$$\begin{aligned}
\Sigma.\mathbf{Vf}(\mathsf{vk}^*_{\mathsf{u}_0}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_0, \alpha'_0), \sigma'_0) &= 1,\\
\Sigma.\mathbf{Vf}(\mathsf{vk}'_{\mathsf{u}_1}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_1, \alpha'_1), \sigma'_1) &= 1,\\
&\vdots\\
\Sigma.\mathbf{Vf}(\mathsf{vk}'_{\mathsf{u}_{\mathsf{tlen}-1}}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_{\mathsf{tlen}-1}, \alpha'_{\mathsf{tlen}-1}), \sigma'_{\mathsf{tlen}-1}) &= 1,\\
\Sigma.\mathbf{Vf}(\mathsf{vk}'_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_{\mathsf{tlen}}, \alpha^*_{\mathsf{tlen}}), \sigma'_{\mathsf{tlen}}) &= 1.
\end{aligned}$$

By the statistical binding of SPB signatures, it follows that the binding key $\mathsf{vk}^*_{\mathsf{u}_0}$ only accepts a unique signature for $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_0, \alpha^*_0)$. This implies that with overwhelming probability $\sigma'_0 = \sigma^*_0$ and $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_0, \alpha'_0) = \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_0, \alpha^*_0)$. By the statistical binding of SPB hash key hk^*_0 at α^*_0, this also implies with overwhelming probability that $\alpha'_0 = \alpha^*_0$. Therefore, $\mathsf{vk}'_{\mathsf{u}_1} = \mathsf{vk}^*_{\mathsf{u}_1}$ and $\mathsf{vk}'_{\mathsf{sib}_1} = \mathsf{vk}_{\mathsf{sib}_1}$

The second verification check is then equivalent to

$$\Sigma.\mathbf{Vf}(\mathsf{vk}^*_{\mathsf{u}_1}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_1, \alpha'_1), \sigma'_1) = 1.$$

By the same argument as before, we get that with overwhelming probability $\sigma'_1 = \sigma^*_1$, $\mathsf{vk}'_{\mathsf{u}_2} = \mathsf{vk}^*_{\mathsf{u}_2}$. $\mathsf{vk}'_{\mathsf{sib}_2} = \mathsf{vk}_{\mathsf{sib}_2}$. By induction, then, we get that with overwhelming probability, $\sigma'_{i-1} = \sigma^*_{i-1}, \mathsf{vk}'_{\mathsf{u}_i} = \mathsf{vk}^*_{\mathsf{u}_i}, \mathsf{vk}'_{\mathsf{sib}_i} = \mathsf{vk}_{\mathsf{sib}_i}$ for all $i = 3, \ldots, \mathsf{tlen}$.

The last verification check is then equivalent to

$$\Sigma.\mathbf{Vf}(\mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_{\mathsf{tlen}}, \alpha^*_{\mathsf{tlen}}), \sigma'_{\mathsf{tlen}}) = 1.$$

By the statistical binding of SPB signatures, it follows that the binding key $\mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}$ accepts a unique signature for $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_{\mathsf{tlen}}, \alpha^*_{\mathsf{tlen}})$. This implies $\sigma'_{\mathsf{tlen}} = \sigma^*_{\mathsf{tlen}}$.

Combining all the above observations, it follows that $\sigma' = \sigma^*$. This is contradictory, and hence, completes the proof of this part.

(ii) for $t = t^*, x \neq x^*$, there does not exist σ' such that $\mathbf{Vf}(\mathsf{pp}^*, \mathsf{vk}^*, t^*, x, \sigma') = 1$: We will prove this by contradiction. Suppose such a signature σ' exists for some $x \neq x^*$. Let the nodes on the path (excluding root node) from root to t^* be $\mathsf{u}_1, \ldots, \mathsf{u}_{\mathsf{tlen}}$ and let their siblings be $\mathsf{sib}_1, \ldots, \mathsf{sib}_{\mathsf{tlen}}$. Then, we can parse the signature as $\sigma' = ((\sigma'_0, \mathsf{vk}'_{\mathsf{u}_1}, \mathsf{vk}'_{\mathsf{sib}_1}), \ldots, (\sigma'_{\mathsf{tlen}-1}, \mathsf{vk}'_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{vk}'_{\mathsf{sib}_{\mathsf{tlen}}}), \sigma'_{\mathsf{tlen}})$. Suppose that $\sigma^* = \mathbf{PSign}(\mathsf{pp}^*, \mathsf{sk}^*, t^*, x^*)$, where $\sigma^* = ((\sigma^*_0, \mathsf{vk}^*_{\mathsf{u}_1}, \mathsf{vk}_{\mathsf{sib}_1}), \ldots, (\sigma^*_{\mathsf{tlen}-1}, \mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}, \mathsf{vk}_{\mathsf{sib}_{\mathsf{tlen}}}), \sigma^*_{\mathsf{tlen}})$. Then, similar to the previous part, one can argue that it must be the case that with overwhelming probability $\sigma'_{i-1} = \sigma^*_{i-1}, \mathsf{vk}'_{\mathsf{u}_i} = \mathsf{vk}^*_{\mathsf{u}_i}, \mathsf{vk}'_{\mathsf{sib}_i} = \mathsf{vk}_{\mathsf{sib}_i}$ for all $i = 1, 2, 3, \ldots, \mathsf{tlen}$. Then, if $\mathbf{Vf}$ algorithm accepts σ', it must be the case that $\mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}$ accepts σ'_{tlen} for the message $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*, t^*||x)$ where $x \neq x^*$. By the statistical binding of SPB signature, we know that $\mathsf{vk}^*_{\mathsf{u}_{\mathsf{tlen}}}$ can only accept a signature for the message $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*, t^*||x^*)$. Therfore, it must be the case that $\mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_{\mathsf{tlen}}, t^*||x) = \mathsf{H}.\mathbf{Hash}(\mathsf{hk}^*_{\mathsf{tlen}}, t^*||x^*)$. By the statistical binding of SPB hash key $\mathsf{hk}^*_{\mathsf{tlen}}$ at $t^*||x^*$, this implies with overwhelming probability that $x = x^*$. This is contradictory, and hence, completes the proof of this part.

□

D Upgrading Security Under Static Corruption

In this section, we prove an upgrade theorem which shows that to prove the static security of our scheme, it suffices to prove a weaker notion of security that imposes additional constraints on the adversary.

D.1 Definition: Selective Single-Challenge Security w.r.t. a Single Inversion

We now define a weaker security notion called selective single-challenge seucrity w.r.t. inversion. In the security game, the adversary $\mathcal{A}$ must be subject to a set of restrictions (besides having to corrupt all players upfront):

- $\mathcal{A}$ must submit a pair of permutations $\pi^{(0)}$ and $\pi^{(1)}$ that differ by only a single inversion, i.e., $\pi^{(1)}$ can be obtained from $\pi^{(0)}$ by swapping a pair of senders' destinations — henceforth we use s and s' to denote this pair of senders; and
- there is only a single challenge time step, and $\mathcal{A}$ must commit to the challenge time step and challenge plaintexts for honest senders upfront.

Introducing a simulated setup algorithm. To make such a definition possible, we must first impose an additional requirement on the syntax. Specifically, we introduce a simulated setup algorithm $\mathbf{Setup}^*$ that is never used in the real world, but needed for the security definition. Specifically, the simulated setup algorithm $\mathbf{Setup}^*(1^\lambda, \mathsf{len}, n, \pi^{(0)}, \pi^{(1)})$ takes in both permutations $\pi^{(0)}, \pi^{(1)}$ that differ by one inversion pertaining to a pair of senders s and s', and it outputs $\{\mathsf{ek}_u\}_{u\in[n]\setminus\{s,s'\}}$, $(\mathsf{ek}_s^{(0)}, \mathsf{ek}_{s'}^{(0)})$, $(\mathsf{ek}_s^{(1)}, \mathsf{ek}_{s'}^{(1)})$, $\{\mathsf{rk}_u\}_{u\in[n]}$, and tk. Importantly, observe that a single set of sender keys $\{\mathsf{ek}_u\}_{u\in[n]\setminus\{s,s'\}}$, for $[n]\setminus\{s,s'\}$ and the routing token tk must be simultaneously compatible with two different sets of sender keys $(\mathsf{ek}_s^{(0)}, \mathsf{ek}_{s'}^{(0)})$, $(\mathsf{ek}_s^{(1)}, \mathsf{ek}_{s'}^{(1)})$, for the swapped senders s and s', corresponding to the two worlds $b = 0$ and $b = 1$, respectively. More precisely, we want the following property:

Marginal distribution of simulated setup statistically close as the real setup:

- for either $b \in \{0, 1\}$, the terms $\left(\{\mathsf{ek}_u\}_{u\in[n]\setminus\{s,s'\}}, (\mathsf{ek}_s^{(b)}, \mathsf{ek}_{s'}^{(b)}), \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}\right)$ output by $\mathbf{Setup}^*$ has negligibly small statistical distance from the output of the real $\mathbf{Setup}(1^\lambda, \mathsf{len}, n, \pi^{(b)})$.

With this additional simulated setup algorithm, we are ready to define selective single-challenge security for a single inversion.

Experiment $\mathsf{NIARStatic\text{-}SelSingleCh\text{-}Inv}^{b,\mathcal{A}}(1^\lambda)$**.**

- $n, \mathsf{len}, \mathcal{K}_S, \mathcal{K}_R, \pi^{(0)}, \pi^{(1)}, t^*, \{x_{u,t^*}^{(0)}, x_{u,t^*}^{(1)}\}_{u\in\mathcal{H}_S} \leftarrow \mathcal{A}(1^\lambda)$;
- $(\{\mathsf{ek}_u\}_{u\in[n]\setminus\{s,s'\}}, \mathsf{ek}_s^{(0)}, \mathsf{ek}_{s'}^{(0)}, \mathsf{ek}_s^{(1)}, \mathsf{ek}_{s'}^{(1)}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}^*(1^\lambda, \mathsf{len}, n, \pi^{(0)}, \pi^{(1)})$;
- $\perp \leftarrow \mathcal{A}(\mathsf{tk}, \{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{rk}_u\}_{u\in\mathcal{K}_R})$;
- 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 $u \in \{s, s'\}$, let $\mathsf{CT}_{u,t} := \mathbf{Enc}(\mathsf{ek}_u^{(\delta_t)}, x_{u,t}, t)$; for all other $u \in \mathcal{H}_S$, let $\mathsf{CT}_{u,t} := \mathbf{Enc}(\mathsf{ek}_u, x_{u,t}, t)$;
  - else if $t = t^*$: for $u \in \{s, s'\}$, let $\mathsf{CT}_{u,t^*} := \mathbf{Enc}(\mathsf{ek}_u^{(b)}, x_{u,t^*}^{(b)}, t^*)$; for all other $u \in \mathcal{H}_S$, let $\mathsf{CT}_{u,t^*} := \mathbf{Enc}(\mathsf{ek}_u, x_{u,t^*}^{(b)}, t^*)$;
  - $\perp \leftarrow \mathcal{A}(\{\mathsf{CT}_{u,t}\}_{u\in\mathcal{H}_S})$;

The adversary is said to be admissible, iff with probability 1, $\mathsf{Leak}^*(\pi^{(0)}, \mathcal{K}_S, \mathcal{K}_R, \{x^{(0)}_{u,t^*}\}_{u\in\mathcal{H}_S}) = \mathsf{Leak}^*(\pi^{(1)}, \mathcal{K}_S, \mathcal{K}_R, \{x^{(1)}_{u,t^*}\}_{u\in\mathcal{H}_S})$ 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 D.1 (Selective single-challenge security w.r.t. inversion under static corruption)**.** We say that a NIAR scheme (augmented with a **Setup*** algorithm) satisfies selective security *w.r.t. inversion* under static corruption, iff for any non-uniform p.p.t. admissible adversary $\mathcal{A}$ which, *with probability* 1*, submits two permutations* π_0 *and* π_1 *that differ by a single inverstion*, $\mathcal{A}$'s views in $\mathsf{NIARStatic\text{-}SelSingleCh\text{-}Inv}^{0,\mathcal{A}}(1^\lambda)$ and $\mathsf{NIARStatic\text{-}SelSingleCh\text{-}Inv}^{1,\mathcal{A}}(1^\lambda)$ are computationally indistinguishable.

D.2 Upgrade Theorem for Static Corruption

In this section, we prove an upgrade theorem for the static corruption setting: we start with a NIAR scheme that is secure when the adversary is subject to the single selective-challenge and single inversion restrictions, and show that the same scheme also satisfies security without the single selective-challenge and single inversion restrictions. It turns out that *we only need to prove the upgrade theorem for the special case when the adversary always corrupts all receivers with probability* 1. This is because later in Appendix F, we show how to compile a NIAR scheme secure under static corruption as long as the adversary always corrupt all receivers, to a NIAR scheme that is fully secure even under adaptive corruptions, and without the restriction that all receivers must be corrupt.

All-receiving-corrupting adversary. Henceforth, if an adversary corrupts all receivers with probability 1, we say that it is an "all-receiver-corrupting adversary". Moreover, if corruption is static, we also refer to such an adversary as a "static, all-receiver-corrupting adversary".

Theorem D.2 (Upgrade theorem for static corruption: removing the selective single challenge and single inversion restrictions)**.** *Given a NIAR scheme which works for single-bit messages, and moreover satisfies selective security w.r.t. inversion (i.e., Definition D.1), under a static, all-receiver-corrupting adversary, then it also satisfies full static corruption security (Definition A.1) subject to an all-receiver-corrupting adversary.*

D.3 Proof of the Upgrade Theorem

We prove Theorem D.2 in the remainder of this section.

D.3.1 Removing the Selective Restriction

We define an adaptive single-challenge notion also w.r.t. inversion, which removes the restriction that the adversary $\mathcal{A}$ must commit to the challenge t^* and challenge plaintexts for honest users upfront.

Definition D.3 (Adaptive single-challenge security w.r.t. inversion under static corruption)**.** We say that a NIAR scheme (augmented with a $\mathbf{Setup}^*$ algorithm) satisfies adaptive single-challenge security w.r.t. inversion under static corruption, iff for any non-uniform p.p.t. admissible adversary $\mathcal{A}$, its views in the following experiments $\mathsf{NIARStatic\text{-}AdSingleCh\text{-}Inv}^{0,\mathcal{A}}(1^\lambda)$ and $\mathsf{NIARStatic\text{-}AdSingleCh\text{-}Inv}^{1,\mathcal{A}}(1^\lambda)$ are computationally indistinguishable.

Experiment $\mathsf{NIARStatic\text{-}AdSingleCh\text{-}Inv}^{b,\mathcal{A}}(1^\lambda)$**.**

- $n, \mathsf{len}, \mathcal{K}_S, \mathcal{K}_R, \pi^{(0)}, \pi^{(1)} \leftarrow \mathcal{A}(1^\lambda)$;
- $(\{\mathsf{ek}_u\}_{u\in[n]\setminus\{s,s'\}}, \mathsf{ek}_s^{(0)}, \mathsf{ek}_{s'}^{(0)}, \mathsf{ek}_s^{(1)}, \mathsf{ek}_{s'}^{(1)}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}^*(1^\lambda, \mathsf{len}, n, \pi^{(0)}, \pi^{(1)})$;
- $\perp \leftarrow \mathcal{A}(\mathsf{tk}, \{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{rk}_u\}_{u\in\mathcal{K}_R})$;
- 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 $\mathsf{CT}_{u,t} := \mathbf{Enc}(\mathsf{ek}_u^{(\delta_t)}, x_{u,t}, t)$ where $\mathsf{ek}_u^{(\delta_t)} := \mathsf{ek}_u$ if $u \notin \{s, s'\}$;
  - else if $\delta_t =$ “$\texttt{challenge}$”, then for $u \in \mathcal{H}_S$, let $\mathsf{CT}_{u,t} := \mathbf{Enc}(\mathsf{ek}_u^{(b)}, x_{u,t}^{(b)}, t)$ where $\mathsf{ek}_u^{(b)} := \mathsf{ek}_u$ if $u \notin \{s, s'\}$;
  - $\perp \leftarrow \mathcal{A}(\{\mathsf{CT}_{u,t}\}_{u\in\mathcal{H}_S})$;

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 “$\texttt{challenge}$”; and
- $\mathsf{Leak}^*(\pi^{(0)}, \mathcal{K}_S, \mathcal{K}_R, \{x_{u,t^*}^{(0)}\}_{u\in\mathcal{H}_S}) = \mathsf{Leak}^*(\pi^{(1)}, \mathcal{K}_S, \mathcal{K}_R, \{x_{u,t^*}^{(1)}\}_{u\in\mathcal{H}_S})$.

The lemma below shows that we can remove the selective single-challenge restriction for free and upgrade the security to adaptive single-challenge, under a single-inversion, all-receiver-corrupting, static-corruption adversary. The lemma considers a NIAR scheme where each sender's plaintext message in every time step is a single bit, i.e., $\mathsf{len} = 1$. This assumption is without loss of generality, since we can always parallel-compose multiple NIAR schemes for $\mathsf{len} = 1$ to get a NIAR scheme for $\mathsf{len} > 1$. We need the $\mathsf{len} = 1$ assumption for the proof to work because we want to make sure the reduction's guess is correct with $1/\mathsf{poly}(\lambda)$ probability — see the proof for more details.

Lemma D.4 (Removing the selective restriction)**.** *Given any NIAR scheme which works for single-bit messages, and moreover satisfies selective single-challenge security w.r.t. a single inversion (Definition D.1) under a static, all-receiver-corrupting adversary, then it is also adaptive single-challenge secure w.r.t. a single inversion (Definition D.3) under a static, all-receiver-corrupting adversary.*

Proof. We consider a reduction $\mathcal{B}$ that interacts with an adaptive single-challenge adversary $\mathcal{A}$ and leverages it to break selective single-challenge security, w.r.t. inversion in both cases.

- At the start, $\mathcal{A}$ submits to $\mathcal{B}$ the terms n, len, the set of corrupt senders $\mathcal{K}_S$, and two permutations $\pi^{(0)}$ and $\pi^{(1)}$ that differ by a single inversion (recall also that $\mathcal{A}$ always corrupts all receivers). Let s, s' be the pair of senders whose destinations are swapped. By the admissibility rule on $\mathcal{A}$, s and s' must be honest.
- Let T be the maximum number of queries made by $\mathcal{A}$. $\mathcal{B}$ guesses at random $t^* \xleftarrow{\$} [T]$, guesses the plaintext messages $x_{s,t^*}^{(0)}$, $x_{s',t^*}^{(0)}$, and lets $x_{s,t^*}^{(1)} = x_{s',t^*}^{(0)}$, $x_{s',t^*}^{(1)} = x_{s,t^*}^{(0)}$.

- With its own challenger, $\mathcal{B}$ corrupts all receivers, and all senders except s and s'. It submits the same permutations $\pi^{(0)}$ and $\pi^{(1)}$, and the guessed t^*, and the guessed plaintext messages $x^{(0)}_{s,t^*}$, $x^{(0)}_{s',t^*}$, $x^{(1)}_{s,t^*}$, $x^{(1)}_{s',t^*}$.
- As a result, $\mathcal{B}$ obtains $\{\mathsf{ek}_u\}_{u\in[n]\setminus\{s,s'\}}$, $\{\mathsf{rk}_u\}_{u\in[n]}$, and tk from its own challenger. $\mathcal{B}$ now passes $\{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}$, $\{\mathsf{rk}_u\}_{u\in[n]}$, and tk to $\mathcal{A}$.
- In the following, if $\mathcal{A}$ ever submits a challenge query during some $t \neq t^*$, or it does not submit a challenge query during t^*, $\mathcal{B}$ aborts.
- During every time step $t \neq t^*$, whenever $\mathcal{A}$ submits $\{x^{(0)}_{u,t}\}_{u\in\mathcal{H}_S}$, $\{x^{(1)}_{u,t}\}_{u\in\mathcal{H}_S}$, and $\delta_t \in \{0,1\}$ (assuming $\mathcal{B}$ has not aborted), $\mathcal{B}$ submits $\{x^{(\delta_t)}_{u,t}\}_{u\in\{s,s'\}}$ and δ_t to its own challenger, and it gets back the encryptions $\mathsf{CT}_{s,t}$ and $\mathsf{CT}_{s',t}$. For any $u \in \mathcal{H}_S\setminus\{s,s'\}$, $\mathcal{B}$ simply uses ek_u to encrypt $x^{(\delta_t)}_{u,t}$ and obtains $\mathsf{CT}_{u,t}$. $\mathcal{B}$ returns to $\mathcal{A}$ the resulting $\{\mathsf{CT}_{u,t}\}_{u\in\mathcal{H}_S}$.
- For $t = t^*$, assuming the reduction $\mathcal{B}$ has not aborted, $\mathcal{A}$ must have specified t^* to be the challenge time step. Now, check if the challenge plaintexts $\{x^{(0)}_{u,t^*}, x^{(1)}_{u,t^*}\}_{u\in\mathcal{H}_S}$ $\mathcal{A}$ has submitted are consistent with the reduction $\mathcal{B}$'s guesses $x^{(0)}_{s,t^*}$, $x^{(0)}_{s',t^*}$, $x^{(1)}_{s,t^*}$, $x^{(1)}_{s',t^*}$. If not, the reduction $\mathcal{B}$ simply aborts.

  Now, $\mathcal{B}$ receives from its challenger CT_{s,t^*} and CT_{s',t^*}. For any $u \in \mathcal{H}_S\setminus\{s,s'\}$, by the admissibility rule on $\mathcal{A}$, it must be that $x^{(0)}_{u,t^*} = x^{(1)}_{u,t^*}$. $\mathcal{B}$ now uses the corresponding ek_u to compute an encryption of $x^{(0)}_{u,t^*}$ and let the result be CT_{u,t^*}. $\mathcal{B}$ now returns $\{\mathsf{CT}_{u,t^*}\}_{u\in\mathcal{H}_S}$ to $\mathcal{A}$.
- At the end, if $\mathcal{B}$ has not aborted, it outputs whatever $\mathcal{A}$ outputs.

Observe that the **Setup*** algorithm executed by $\mathcal{B}$'s challenger does not depend on the challenge time step t^* or the challenge plaintexts. Therefore, if $\mathcal{B}$'s challenger is in world $b = 0$, and if the reduction did not abort, then $\mathcal{A}$'s view is identically distributed as in $\mathsf{NIARStatic\text{-}AdSingleCh}^{0,\mathcal{A}}$ (subject to single inversion). Otherwise, if $\mathcal{B}$'s challenger is in world $b = 1$, and if the reduction did not abort, then $\mathcal{A}$'s view is identically distributed as in $\mathsf{NIARStatic\text{-}AdSingleCh}^{1,\mathcal{A}}$ (subject to single inversion). The lemma follows by observing as long as the message length is only one bit, the probability that the reduction $\mathcal{B}$ guesses correctly is $1/\mathsf{poly}(\lambda)$. □

D.3.2 Removing the Single Challenge Restriction

We now prove that we can further remove the single challenge restriction.

Lemma D.5 (Removing the single challenge restriction)**.** *Given any NIAR scheme that satisfies adaptive single-challenge security w.r.t. inversion (Definition D.3) subject to a static, all-receiver corrupting adversary, the same scheme also satisfies Definition A.1 subject to a static, single-inversion, all-receiver-corrupting adversary.*

Proof. Let T be the maximum number of queries made by the adversary. We consider the following sequence of hybrids indexed by $i \in \{0,1,\ldots,T\}$, and recall that the permutations $\pi^{(0)}, \pi^{(1)}$ submitted by the adversary $\mathcal{A}$ must differ by only one inversion where the swapped pair of senders is denoted s and s' below.

Hybrid experiment Hyb_i:

- $n, \mathsf{len}, \mathcal{K}_S, \mathcal{K}_R, \pi^{(0)}, \pi^{(1)} \leftarrow \mathcal{A}(1^\lambda)$;

- $(\{\mathsf{ek}_u\}_{u\in[n]\setminus\{s,s'\}}, \mathsf{ek}_s^{(0)}, \mathsf{ek}_{s'}^{(0)}, \mathsf{ek}_s^{(1)}, \mathsf{ek}_{s'}^{(1)}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}^*(1^\lambda, \mathsf{len}, n, \pi^{(0)}, \pi^{(1)})$;
- $\bot \leftarrow \mathcal{A}(\mathsf{tk}, \{\mathsf{ek}_u\}_{u\in\mathcal{K}_S}, \{\mathsf{rk}_u\}_{u\in\mathcal{K}_R})$;
- For $t = 1, 2, \ldots$:
  - $(\{x_{u,t}^{(0)}\}_{u\in\mathcal{H}_S}, \{x_{u,t}^{(1)}\}_{u\in\mathcal{H}_S}) \leftarrow \mathcal{A}(\bot)$;
  - if $t > i$: then for $u \in \mathcal{H}_S$, let $\mathsf{CT}_{u,t} := \mathbf{Enc}(\mathsf{ek}_u^{(0)}, x_{u,t}, t)$ where $\mathsf{ek}_u^{(0)} := \mathsf{ek}_u$ if $u \notin \{s, s'\}$;
  - else if $t \le i$, then for $u \in \mathcal{H}_S$, let $\mathsf{CT}_{u,t} := \mathbf{Enc}(\mathsf{ek}_u^{(1)}, x_{u,t}^{(1)}, t)$ where $\mathsf{ek}_u^{(1)} := \mathsf{ek}_u$ if $u \notin \{s, s'\}$;
  - $\bot \leftarrow \mathcal{A}(\{\mathsf{CT}_{u,t}\}_{u\in\mathcal{H}_S})$;

$\mathcal{A}$ is said to be admissible iff it is subject to the same admissibility rules as Definition A.1; moreover, we also assume that $\mathcal{A}$ respects the single-inversion, and all-receiver-corrupting[6] constraints. Since the marginal distribution of $\mathbf{Setup}^*$ is statistical close to the real-world $\mathbf{Setup}$ algorithm, it holds that $\mathcal{A}$'s view in Hyb_0 is statistically close to $\mathsf{NIARStatic}^{0,\mathcal{A}}$, and its view in Hyb_T is statistically close to $\mathsf{NIARStatic}^{1,\mathcal{A}}$. It suffices to prove that every adjacent pair of hybrids are computationally indistinguishable to the adversary. This can be shown through a straightforward reduction to the adaptive single-challenge security w.r.t. inversion (Definition D.3). □

D.3.3 Removing the Single Inversion Restriction

We next show how to remove the single inversion restriction on the adversary.

Lemma D.6 (Removing the single inversion restriction (static corruption))**.** *Given a NIAR scheme that satisfies Definition A.1 subject to a static, single-inversion, all-receiver-corrupting adversary, the same scheme also satisfies Definition A.1 subject to a static, all-receiver-corrupting adversary.*

Proof. Given any two permutations $\pi^{(0)}$ and $\pi^{(1)}$ submitted by $\mathcal{A}$, let $\overline{C}(\pi^{(0)}, \pi^{(1)})$ be the set of senders that have different destinations in $\pi^{(0)}$ and $\pi^{(1)}$ — by the admissibility rule on $\mathcal{A}$, it must be that $\overline{C}(\pi^{(0)}, \pi^{(1)})$ are all honest senders. We define a sequence of permutations denoted $\pi_0^*, \ldots, \pi_n^*$ where $\pi_0^* = \pi^{(0)}$, and for any $0 < i \le n$, π_i^* is almost the same as π_{i-1}^*, except that if $i \le |\overline{C}(\pi^{(0)}, \pi^{(1)})|$, then we additionally swap the destinations of the i-th honest sender in $\overline{C}(\pi^{(0)}, \pi^{(1)})$ denoted u_i^* and whoever is sending to $\pi^{(1)}(u_i^*)$ in π_{i-1}^* — by construction, the sender u_i^* is swapping destinations with another sender that must lie within the the set $\overline{C}(\pi^{(0)}, \pi^{(1)})$. Else if $i > |\overline{C}(\pi^{(0)}, \pi^{(1)})|$, then, $\pi_i^* = \pi_{i-1}^*$. By construction, in π_i^*, the first i honest senders in $\overline{C}(\pi^{(0)}, \pi^{(1)})$ have their correct destinations as in $\pi^{(1)}$, and thus $\pi_n^* = \pi^{(1)}$.

We now consider a sequence of hybrid experiments denoted Hyb_i where $i \in \{0, 1, \ldots, n\}$, in which a challenger interacts with an adversary $\mathcal{A}$ that has the same interface as a $\mathsf{NIARStatic}^{b,\mathcal{A}}$ adversary, and moreover, it always corrupts all receivers upfront. Namely, $\mathcal{A}$ submits $n, \mathsf{len}, \mathcal{K}_S, \mathcal{K}_R, \pi^{(0)}, \pi^{(1)}$ upfront where $\mathcal{K}_R$ is guaranteed to be $[n]$, and then in every time step t, it submits $\{x_{u,t}^{(0)}, x_{u,t}^{(1)}\}_{u\in\mathcal{H}_S}$. In Hyb_i, the challenger computes the responses to $\mathcal{A}$ as follows:

- It calls the $\mathbf{Setup}$ algorithm on the input len, n and the permutation π_i^* (which is uniquely determined given $\pi^{(0)}$ and $\pi^{(1)}$ submitted by $\mathcal{A}$),
- During each time step t, upon receiving $\{x_{u,t}^{(0)}, x_{u,t}^{(1)}\}_{u\in\mathcal{H}_S}$, do the following: for each $u \in \mathcal{H}_S$, let $x_{u,t}^* = x_{u',t}^{(1)}$ where $u' = {\pi^{(1)}}^{-1}(\pi_i^*(u))$; now compute $\mathsf{ct}_{u,t} = \mathbf{Enc}(\mathsf{ek}_u, x_{u,t}^*, t)$.

[6] The all-receiver-corrupting restriction is not important for this lemma, that is, the lemma and the proof still hold if we remove every occurrence of "all-receiver-corrupting".

By construction, and due to the admissibility rule on $\mathcal{A}$, Hyb_0 is the same as $\mathsf{NIARStatic}^{0,\mathcal{A}}$, and Hyb_n is the same as $\mathsf{NIARStatic}^{1,\mathcal{A}}$. It suffices to argue that the adversary's views in every pair of adjacent hybrids Hyb_i and Hyb_{i+1} are computationally indistinguishable. This can be achieved through a reduction to the static single-inversion security under all-corrupt-receivers.

If $\pi^*_{i+1} = \pi^*_i$, then by definition, Hyb_i and Hyb_{i+1} are identically. Henceforth we focus on the case when π^*_{i+1} and π^*_i differ by exactly one inversion. Consider a reduction $\mathcal{B}$ which receives $n, \mathsf{len}, \mathcal{K}_S, \mathcal{K}_R, \pi^{(0)}, \pi^{(1)}$ from $\mathcal{A}$ upfront where $\mathcal{K}_R$ is guaranteed to be $[n]$, $\mathcal{B}$ submits to its own challenger $n, \mathsf{len}, \mathcal{K}_S, \mathcal{K}_R, \pi^*_i, \pi^*_{i+1}$ and passes the responses to $\mathcal{A}$. In every time step t, $\mathcal{B}$ receives $\{x^{(0)}_{u,t}, x^{(1)}_{u,t}\}_{u\in\mathcal{H}_S}$ from $\mathcal{A}$. Now, for each $u \in \mathcal{H}_S$, let $x^*_{u,t} = x^{(1)}_{u',t}$ where $u' = {\pi^{(1)}}^{-1}(\pi^*_i(u))$, and let $y^*_{u,t} = x^{(1)}_{u',t}$ where $u' = {\pi^{(1)}}^{-1}(\pi^*_{i+1}(u))$. It submits to its own challenger the challenge plaintexts $\{x^*_{u,t}, y^*_{u,t}\}_{u\in\mathcal{H}_S}$, and passes the responses to $\mathcal{A}$. $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

If $\mathcal{B}$'s challenger is in world $b = 0$ and encrypts the plaintexts $\{x^*_{u,t}\}_{u\in\mathcal{H}_S}$, then $\mathcal{A}$'s view is identically distributed as in Hyb_i; else $\mathcal{B}$'s challenger is in world $b = 1$ and encrypts the plaintexts $\{y^*_{u,t}\}_{u\in\mathcal{H}_S}$, then $\mathcal{A}$'s view is identically distributed as in Hyb_{i+1}.

Finally, by construction, if $\mathcal{A}$ respects its admissibility rules, then $\mathcal{B}$ respects its admissibility rules as well. Moreover, as mentioned earlier, $\mathcal{B}$ respects the single-inversion constraint. Therefore, $\mathcal{B}$ can translate $\mathcal{A}$'s advantage in distinguishing Hyb_i and Hyb_{i+1} into its own advantage at breaking the single-inversion static-corruption security of NIAR (subject to all-corrupting receivers). □

D.3.4 Completing the Proof

The proof of Theorem D.2 follows directly by combining Lemmas D.4 to D.6.

E Proof of Selective Single-Challenge Security w.r.t. a Single Inversion

We now prove indistinguishability w.r.t. inversions against an adversary that additionally satisfies the selective single-challenge restriction. Formally, we have the following theorem.

Theorem E.1. *Suppose that* PRF *is a secure puncturable PRF,* Sig *is a secure deterministic SSU signature scheme, and* iO *is a secure indistinguishability obfuscation scheme. Then our NIAR construction in Section 6.1 satisfies selective single-challenge security w.r.t. inversion under static corruption (Definition D.1) subject to an all-receiver-corrupting adversary.*

In order to invoke Definition D.1, we next describe the **Setup*** algorithm. We will show in Claim E.2 that the **Setup*** algorithm satisfies the requirements that its marginal distribution is statistically close to the real setup.

Setup$^*(1^\lambda, \mathsf{len}, n, \pi^{(0)}, \pi^{(1)})$: on inputs the security parameter 1^λ, the individual message length len, the number of parties n, the permutations $\pi^{(0)}$ and $\pi^{(1)}$, does the following:

1. Set $\mathsf{tlen} = \log^2(\lambda)$.
2. **Sampling Routes**: Let the two senders that $\pi^{(0)}$ and $\pi^{(1)}$ differ on be s and s'. Run the $\mathsf{AssignRoutes}^*$ procedure (Appendix B.1) on inputs $(1^\lambda, n, \pi^{(0)}, \pi^{(1)})$. Abort if it outputs $\perp$. Else parse the output as $(\{\mathsf{rte}_u\}_{u\in[n]\setminus\{s,s'\}}, \{\mathsf{rte}_u^{(0)}, \mathsf{rte}_u^{(1)}\}_{u\in\{s,s'\}})$.
3. **Sampling Wire Keys**: Same as in **Setup** (Figure 3).
4. **Signing Routes**: For each sender $u \in [n] \setminus \{s, s'\}$ do the following:
   (a) Parse $\mathsf{rte}_u = (j_1, \ldots, j_L)$. Sign rte_u using route signing keys for each wire along rte_u, that is, for $\ell \in [L]$ compute $\mathsf{rsig}_\ell = \mathsf{Sig.Sign}(\mathsf{rpp}_{(\ell,j_\ell)}, \mathsf{rsk}_{(\ell,j_\ell)}, 1, \mathsf{rte}_u)$. // *Same way as in* **Setup** *(Figure 3).*
   (b) Set $\overline{\mathsf{rte}}_u = (\mathsf{rte}_u, \mathsf{rsig}_u = (\mathsf{rsig}_1, \ldots, \mathsf{rsig}_L))$. // *Same way as in* **Setup** *(Figure 3).*

   For each sender $u \in \{s, s'\}$ and for each $\beta \in \{0, 1\}$, do the following:
   (a) Parse $\mathsf{rte}_u^{(\beta)} = (j_1, \ldots, j_L)$. Sign $\mathsf{rte}_u^{(\beta)}$ using route signing keys for each wire along $\mathsf{rte}_u^{(\beta)}$, that is, for $\ell \in [L]$ compute $\mathsf{rsig}_\ell^{(\beta)} = \mathsf{Sig.Sign}(\mathsf{rpp}_{(\ell,j_\ell)}, \mathsf{rsk}_{(\ell,j_\ell)}, 1, \mathsf{rte}_u^{(\beta)})$. // *Same way as in* **Setup** *(Figure 3).*
   (b) Set $\overline{\mathsf{rte}}_u^{(\beta)} = (\mathsf{rte}_u^{(\beta)}, \mathsf{rsig}_u = (\mathsf{rsig}_1^{(\beta)}, \ldots, \mathsf{rsig}_L^{(\beta)}))$. // *Same way as in* **Setup** *(Figure 3).*
5. **Setting Routing Token**: Same as in **Setup** (Figure 3).
6. **Setting Sender Keys**: For each $u \in [n] \setminus \{s, s'\}$, set $\mathsf{ek}_u = (k_{(1,j_1)}, \mathsf{mpp}_{(1,j_1)}, \mathsf{msk}_{(1,j_1)}, \overline{\mathsf{rte}}_u)$. For each $u \in \{s, s'\}$ and $\beta \in \{0, 1\}$, set $\mathsf{ek}_u^{(\beta)} = (k_{(1,j_1)}, \mathsf{mpp}_{(1,j_1)}, \mathsf{msk}_{(1,j_1)}, \overline{\mathsf{rte}}_u^{(\beta)})$.
7. **Setting Receiver Keys**: For each $v \in [n]$, set $\mathsf{rk}_v = k_{(L,2v-1)}$.
8. Output $(\{\mathsf{ek}_u\}_{u\in[n]\setminus\{s,s'\}}, \{\mathsf{ek}_u^{(0)}, \mathsf{ek}_u^{(1)}\}_{u\in\{s,s'\}}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk})$.

Figure 4: The **Setup*** algorithm for selective single-challenge security w.r.t. inversion under static corruption as defined in Definition D.1.

Claim E.2. *If the routing network satisfies obliviousness as defined in Definition B.1, then the* **Setup*** *algorithm in Figure 4 satisfies the requirement that its marginal distribution is statistically close to the real setup. More formally,*

- *for either* $b \in \{0, 1\}$*, the terms* $\left(\{\mathsf{ek}_u\}_{u\in[n]\setminus\{s,s'\}}, (\mathsf{ek}_s^{(b)}, \mathsf{ek}_{s'}^{(b)}), \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}\right)$ *output by* **Setup**$^*(1^\lambda, \mathsf{len}, \pi^{(0)}, \pi^{(1)})$ *has negligibly small statistically distance from the output of the real* **Setup**$(1^\lambda, \mathsf{len}, n, \pi^{(b)})$.

Proof. For either $b \in \{0, 1\}$, the terms $\left(\{\mathsf{ek}_u\}_{u\in[n]\setminus\{s,s'\}}, (\mathsf{ek}_s^{(b)}, \mathsf{ek}_{s'}^{(b)}), \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}\right)$ output by **Setup*** differs from the output of the real **Setup** only in the following way: while **Setup** samples the

senders' routes using $\mathsf{AssignRoutes}$, $\mathbf{Setup}^*$ samples them using $\mathsf{AssignRoutes}^*$ algorithm. Therefore, for either $b \in \{0,1\}$, the indistinguishability of the terms $(\{\mathsf{ek}_u\}_{u\in[n]\setminus\{s,s'\}}, (\mathsf{ek}_s, \mathsf{ek}_{s'}), \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk})$ output by the real $\mathbf{Setup}(1^\lambda, \mathsf{len}, n, \pi^{(b)})$ and the terms $\left(\{\mathsf{ek}_u\}_{u\in[n]\setminus\{s,s'\}}, (\mathsf{ek}_s^{(b)}, \mathsf{ek}_{s'}^{(b)}), \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}\right)$ output by $\mathbf{Setup}^*$ follows from the obliviousness of the routing network (Definition B.1).

□

E.1 The Hybrids for Theorem E.1

We are now ready to prove Theorem E.1. We do this via the sequence of hybrids below. In this sequence, $\mathsf{Hyb}_0^{(0)}$ implements $\mathsf{NIARStatic\text{-}SelSingleCh\text{-}Inv}^{0,\mathcal{A}}$, and $\mathsf{Hyb}_0^{(1)}$ implements $\mathsf{NIARStatic\text{-}SelSingleCh\text{-}Inv}^{1,\mathcal{A}}$. Thus it suffices to prove indistinguishability between each successive pair of hybrids $\mathsf{Hyb}_i^{(b)}$ and $\mathsf{Hyb}_{i+1}^{(b)}$ for each i and $b \in \{0,1\}$, and that the final hybrid $\mathsf{Hyb}_9^{(b)}$ is wholly independent of b.

In the following, we say that a wire is "*corrupt*" or "*honest*" if it is on a path which originates with a *corrupt* or *honest* sender respectively, and we say that a wire is an "*inversion*" wire if it is on one of the paths that originate with the two *honest* senders s and s'. We say all wires apart from *corrupt* and *honest* wires to be "*filler*" wires. Also, we sometimes use the shorthand "wire (ℓ, j)" to refer to j^{th} wire in the ℓ^{th} layer.

E.1.1 Informal Hybrids

In the real world hybrid $\mathsf{Hyb}_0^{(b)}$, the adversary's view contains the sender keys of corrupt senders, the receiver keys of corrupt receivers, the routing token, non-challenge and challenge ciphertexts for honest senders. Let's understand which all of these terms contain information about the challenge bit b.

- As $\mathsf{AssignRoutes}^*$ assigns a single route to each corrupt sender, hence, the sender keys of corrupt senders are independent of b. All receiver keys are independent of b.
- The non-challenge ciphertexts are independent of b. It is only the challenge ciphertexts that are dependent on b as follows. The challenger provides the following challenge ciphertexts to the adversary.
  - for all $u \in \{s, s'\}$: $\mathsf{CT}_{u,t^*} = \mathbf{Enc}(\mathsf{ek}_u^{(b)}, x_{u,t^*}^{(b)}, t^*)$.
  - for all $u \in \mathcal{H}_S \setminus \{s, s'\}$: $\mathsf{CT}_{u,t^*} = \mathbf{Enc}(\mathsf{ek}_u, x_{u,t^*}^{(b)}, t^*)$.

  CT_{u,t^*} contain information about b only for $u \in \{s, s'\}$: Observe that for senders $u \in \mathcal{H}_S \setminus \{s, s'\}$, $\pi^{(0)}(u) = \pi^{(1)}(u)$. This means that the receiver remains the same across the two worlds for each such sender. And recall that we are proving security against an all-receiver-corrupting adversary. The admissibility criteria requires then that each corrupt receiver must receive the same message across the two worlds. This implies that $x_{u,t^*}^{(0)} = x_{u,t^*}^{(1)}$ for all $u \in \mathcal{H}_S \setminus \{s, s'\}$. Hence, it follows that CT_{u,t^*} is independent of b for all such honest senders u. Therefore, the adversary's view contains information about b in only the challenge ciphertexts for $u \in \{s, s'\}$. Removing this information about bit b is non-trivial and requires an intricate sequence of hybrids as discussed below.
- From the description of the circuit $\mathsf{Gate}_{(\ell,g)}$ in Figure 2, it seems that the routing token contains no information of b. But, observe that the obfuscated gates treat filler and non-filler

wires differently. More importantly, notice that its possible that a wire is an *inversion* wire when $b = 0$ and *filler* wire when $b = 1$ or vice versa. This is not the case for a *corrupt* wire or a *honest non-inversion* wire as $\mathsf{AssignRoutes}^*$ only assigns a single route for all senders $u \notin \{s, s'\}$. Hence, one of the goals of our sequence of hybrids is to arrive at the final hybrid in which the obfuscated gates will treat the *inversion* and *filler* wires exactly the same in the challenge round t^*.

Our sequence of hybrids is as follows.

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

Hybrids $\mathsf{Hyb}_1^{(b)}, \mathsf{Hyb}_2^{(b)}, \mathsf{Hyb}_3^{(b)}$: The foremost change we make is to ensure that for all the corrupt wires, the authenticated routes that the obfuscated gates obtain are as intended by $\mathbf{Setup}^*$. This change is accomplished in three steps.

- $\mathsf{Hyb}_1^{(b)}$: For each corrupt wire in each layer, puncture the route signing key such that it can only sign the expected route that passes through that wire. $\mathsf{Hyb}_1^{(b)}$ is identical to $\mathsf{Hyb}_0^{(b)}$ as none of the route signing keys are in the view of the adversary.
- $\mathsf{Hyb}_2^{(b)}$: For each corrupt wire in each layer, bind the route signing and verification key such that the verification key for a corrupt wire only accepts signature for the expected route that passes through that wire. Indistinguishability follows from indistinguishability of punctured and binding modes of SSU signatures.
- $\mathsf{Hyb}_3^{(b)}$: For each corrupt wire in each layer, hardwire the expected route and expected route signatures. Further, update the route authentication check to directly compare routes and route signatures against hardcoded values. Indistinguishability follows from statistical unforgeability of the SSU signatures at the binding points and iO security.

Hybrids $\mathsf{Hyb}_4^{(b)}$: For all filler/inversion wires of layer $\ell = 1$, puncture the message signing keys such that they can sign any message for non-challenge round $t \neq t^*$ and only the challenge message for the challenge round $t = t^*$. 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 E.6).

Hybrids $\mathsf{Hyb}_5^{(b)}$: For all filler/inversion wires of layer $\ell = 1$, bind the message signing and verification keys such that the verification key accepts signatures for any message for non-challenge round $t \neq t^*$ and only the challenge message for the challenge round $t = t^*$. Indistinguishability follows from indistinguishability of punctured and binding modes of SSU signatures.

Layered hybrids. Now that we have shown how to change the message signing and verification keys for all filler/inversion wires in layer $\ell = 1$, we show how to do the same for rest of the layers. This is done in a layer-by-layer fashion. Below, we describe how to do it for layer $\ell = 2$ through a sequence of hybrids $\mathsf{Hyb}_{6,1,1}^{(b)}, \mathsf{Hyb}_{6,1,2}^{(b)}, \mathsf{Hyb}_{6,1,3}^{(b)}, \mathsf{Hyb}_{6,1,4}^{(b)}, \mathsf{Hyb}_{6,1,5}^{(b)}$. Once we have reached $\mathsf{Hyb}_{6,1,5}^{(b)}$, similiar arguments can be made for layer $\ell = 3$ via hybrids $\mathsf{Hyb}_{6,2,1}^{(b)}, \mathsf{Hyb}_{6,2,2}^{(b)}, \mathsf{Hyb}_{6,2,3}^{(b)}, \mathsf{Hyb}_{6,2,4}^{(b)}, \mathsf{Hyb}_{6,2,5}^{(b)}$. These layer-by-layer changes carry on for rest of the layers till we finally arrive at hybrid $\mathsf{Hyb}_{6,L-1,5}^{(b)}$.

Hybrids $\mathsf{Hyb}_{6,1,1}^{(b)}, \mathsf{Hyb}_{6,1,2}^{(b)}, \mathsf{Hyb}_{6,1,3}^{(b)}, \mathsf{Hyb}_{6,1,4}^{(b)}, \mathsf{Hyb}_{6,1,5}^{(b)}$: The goal of this set of hybrids is to reach a distribution where the message signing and verification keys for all filler/inversion wires of layer $\ell = 2$ are in binding mode. We show how to reach this distribution via a sequence of steps.

- $\mathsf{Hyb}^{(b)}_{6,1,1}$: For all filler/inversion wires in layer $\ell = 1$, hardcode the expected message signatures and messages. Further, update the message authentication check to directly compare messages and message signatures against hardcoded values. Route authentication check remains the same as before.

- $\mathsf{Hyb}^{(b)}_{6,1,2}$: For all filler/inversion wires in layer $\ell = 1$, hardcode the expected input ciphertext and puncture the PRF keys. Further, replace message and route authentication checks with ciphertext comparison.

  At this point, observe that for all filler/inversion wires in layer $\ell = 2$, the unpunctured message signing keys can not be used to sign messages other than the challenge messages for the challenge round t^*.

- $\mathsf{Hyb}^{(b)}_{6,1,3}$: For all filler/inversion wires of layer $\ell = 2$, puncture the message signing keys such that they can sign any message for non-challenge round $t \neq t^*$ and only the challenge message for the challenge round $t = t^*$. These punctured message signing keys are hardcoded in the gates in the first layer $\ell = 1$. Importantly, changing these message signing keys does not change the gate's functionality because as noted in the previous hybrid, the unpunctured signing keys for layer $\ell = 2$ can not be used to sign messages other than the challenge messages for the challenge round t^* either. Consequently, the security of indistinguishability obfuscation can be invoked to transition from $\mathsf{Hyb}^{(b)}_{6,1,2}$ to $\mathsf{Hyb}^{(b)}_{6,1,3}$ (formal proof in Claim E.10).

- $\mathsf{Hyb}^{(b)}_{6,1,4}$: For all filler/inversion wires of layer $\ell = 2$, bind the message signing and verification keys such that the verification key accepts signatures for any message for non-challenge round $t \neq t^*$ and only the challenge message for the challenge round $t = t^*$. Indistinguishability follows from indistinguishability of punctured and binding modes of SSU signatures.

- $\mathsf{Hyb}^{(b)}_{6,1,5}$: For all filler/inversion wires of layer $\ell = 2$, hardcode the expected output ciphertext for the challenge round, and puncture the PRF keys. Further, update preparing the output ciphertext to directly use the hardcoded values.

Hybrid $\mathsf{Hyb}^{(b)}_7$: 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 inversion wire in the last layer has the destination that is corrupt, then, we do not make any change to the hardwired outgoing ciphertexts in the last layer. Further, the challenge ciphertexts given out to the adversary for $u \in \{s, s'\}$ are set in a manner consistent with the hardcoded input ciphertexts in layer $\ell = 1$.

Hybrids $\mathsf{Hyb}^{(b)}_8, \mathsf{Hyb}^{(b)}_9$: In $\mathsf{Hyb}^{(b)}_7$, 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 unbind and unpuncture all the message signing and verification keys.

Analysis of the final hybrid: In hybrid $\mathsf{Hyb}^{(b)}_9$, 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.

- Observe that the challenge ciphertext for $u \in \{s, s'\}$ for the challenge round t^* obtained by the adversary are random strings independent of challenge bit b.

- 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 (i.e., inversion wires with corrupt receiver as destination) have the same value across the two worlds by the admissibility criteria. Put in other words, for the challenge round t^*, the circuit description (Figure 15) treats filler and inversion 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 E.16.

E.1.2 Formal Hybrids

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

Hybrid $\mathsf{Hyb}_1^{(b)}$: This hybrid is identical to $\mathsf{Hyb}_0^{(b)}$ except that during the algorithm **Setup***, 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 $j_\ell \in \mathsf{rte}_u^*$ for some sender $u \in \mathcal{K}_S$, we compute the route signatures as follows:

$$(\mathsf{rsk}_{(\ell,i)}, \mathsf{rsk}'_{(\ell,i)}, \mathsf{rvk}_{(\ell,i)}, \mathsf{rpp}_{(\ell,i)}) \leftarrow \mathsf{Sig}.\mathbf{PuncturedSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}_{\mathsf{rte}}, 1, \mathsf{rte}_u^*) \ ,$$
$$\mathsf{rsig}_\ell^* = \mathsf{Sig}.\mathbf{PSign}(\mathsf{rpp}_{(\ell,i)}, \mathsf{rsk}'_{(\ell,i)}, 1, \mathsf{rte}_u^*) \ .$$

Hybrid $\mathsf{Hyb}_2^{(b)}$: This hybrid is identical to $\mathsf{Hyb}_1^{(b)}$ except that during the algorithm **Setup***, the challenger generates binding route signature keys for all the *corrupt* wires. More specifically, for each wire $(\ell, i = j_\ell)$, where $j_\ell \in \mathsf{rte}_u^*$ for some sender $u \in \mathcal{K}_S$, we compute the route signatures as follows:

$$(\mathsf{rsk}^*_{(\ell,i)}, \mathsf{rvk}^*_{(\ell,i)}, \mathsf{rpp}^*_{(\ell,i)}) \leftarrow \mathsf{Sig}.\mathbf{BindingSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}_{\mathsf{rte}}, 1, \mathsf{rte}_u^*) \ ,$$
$$\mathsf{rsig}_\ell^* = \mathsf{Sig}.\mathbf{PSign}(\mathsf{rpp}^*_{(\ell,i)}, \mathsf{rsk}^*_{(\ell,i)}, 1, \mathsf{rte}_u^*).$$

Then, we replace the gates $\mathsf{Gate}_{(\ell,g)}$ for all $\ell \in [L-1], g \in [G]$ as described in Figure 5.

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 public parameter $\mathsf{rpp}^*_{(\ell,i)}$ and route verification key $\mathsf{rvk}^*_{(\ell,i)}$ are hardcoded instead of $\mathsf{rpp}_{(\ell,i)}$ and $\mathsf{rvk}_{(\ell,i)}$.

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

Figure 5: 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 $\overline{\mathsf{rte}}_u^* = (\mathsf{rte}_u^* = (j_1^*, \ldots, j_L^*), \mathsf{rsig}_u^* = (\mathsf{rsig}_1^*, \ldots, \mathsf{rsig}_L^*))$ for some sender $u \in [n]$. Then, hardcode $(i, \overline{\mathsf{rte}}_u^*)$.

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 $\overline{\mathsf{rte}}_u$ as $(\mathsf{rte}_u = (j_1, \ldots, j_L), \mathsf{rsig}_u = (\mathsf{rsig}_1, \ldots, \mathsf{rsig}_L))$ and perform the following checks to authenticate the route rte_u: If $\mathsf{rte}_u \neq \mathsf{rte}_u^*$, 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 $\mathsf{msig}' = \mathsf{Sig}.\mathbf{Sign}(\mathsf{mpp}_{(\ell+1,j)}, \mathsf{msk}_{(\ell+1,j)}, t, (x, \overline{\mathsf{rte}}_u^*))$ and $\mathsf{CT}_{(\ell+1,j)} = (x, \overline{\mathsf{rte}}_u^*, \mathsf{msig}') \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,j)}, t)$.
    - Step iii is same as in $\mathsf{Hyb}_2^{(b)}$.
- Steps 2 and 3 are same as in $\mathsf{Hyb}_2^{(b)}$.

Figure 6: The circuit $\mathsf{Gate}_{(\ell,g)}$ in hybrid experiment $\mathsf{Hyb}_3^{(b)}$.

Hybrid $\mathsf{Hyb}_3^{(b)}$: In this hybrid, for the obfuscated gates generated by **Setup***, 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 6.

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

$$(\mathsf{msk}_{(1,i)}, \mathsf{msk}'_{(1,i)}, \mathsf{mvk}_{(1,i)}, \mathsf{mpp}_{(1,i)}) \leftarrow \mathsf{Sig}.\mathbf{PuncturedSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}_m, t^*, \tilde{x}^*).$$

Then, during **Enc** algorithm, whenever the challenger has to compute message signatures for the first layer, it computes them as follows:

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

Hybrid $\mathsf{Hyb}_5^{(b)}$: This hybrid is identical to $\mathsf{Hyb}_4^{(b)}$, except that during **Setup***, the challenger binds message signature keys for all *filler/inversion* wires $(1,i)$ in the first layer at the challenge round t^* and challenge plaintext $\tilde{x}^* = (x_{u,t^*}^{(b)}, \overline{\mathsf{rte}}_u^{(b)})$ (or $\tilde{x}^* = (\bot_{\mathsf{filler}}, \bot_{\mathsf{filler}})$ in case of *filler*), that is, for each such wire $(1,i)$,

$$(\mathsf{msk}^*_{(1,i)}, \mathsf{mvk}^*_{(1,i)}, \mathsf{mpp}^*_{(1,i)}) \leftarrow \mathsf{Sig}.\mathbf{BindingSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}_m, t^*, \tilde{x}^*).$$

Then, the challenger replaces the gates $\mathsf{Gate}_{(1,g)}$ for all $g \in [G]$ as described in Figure 7.

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 $\mathsf{Hyb}_3^{(b)}$ except that for each *filler/inversion* wire $i \in \mathsf{Input}_{(1,g)}$, the message public parameter $\mathsf{mpp}^*_{(1,i)}$ and the message verification key $\mathsf{mvk}^*_{(1,i)}$ are hardcoded instead of $\mathsf{mpp}_{(1,i)}, \mathsf{mvk}_{(1,i)}$.

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

Figure 7: The circuit $\mathsf{Gate}_{(\ell,g)}$ in hybrid experiment $\mathsf{Hyb}_5^{(b)}$.

Hybrid $\mathsf{Hyb}_{6,\ell,1}^{(b)}$ for each $\ell \in [L-1]$: This hybrid is identical to $\mathsf{Hyb}_{6,\ell-1,5}^{(b)}$ (or $\mathsf{Hyb}_5^{(b)}$ if $\ell = 1$), except that during **Setup***, for all the gates in layer ℓ, i.e., $\mathsf{Gate}_{(\ell,g)}$ for all $g \in [G]$, for all *filler/inversion* wires (ℓ,i) in the input layer ℓ, the challenger hardcodes the expected challenge message $x^*_{(\ell,i)} = x^{(b)}_{u,t^*}$ (or $x^*_{(\ell,1)} = \bot_{\mathsf{filler}}$ in case of *filler* wire), the expected route $\overline{\mathsf{rte}}^* = \overline{\mathsf{rte}}^*_u$ (or $\overline{\mathsf{rte}}^* = \bot_{\mathsf{filler}}$ in case of *filler* wire) and the message signature $\mathsf{msig}^*_{(\ell,i)} = \mathsf{Sig}.\mathbf{PSign}(\mathsf{mpp}^*_{(\ell,i)}, \mathsf{msk}^*_{(\ell,i)}, t^*, (x^*, \overline{\mathsf{rte}}^*))$ for the challenge round t^* and compares the message, route and message signature in the decrypted plaintext against 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 the hardcoded $x^*_{(\ell,i)}$ and $\overline{\mathsf{rte}}^*$ 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 8.

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/inversion* input wire $i \in \mathsf{Input}_{(\ell,g)}$, the expected challenge message $x^*_{(\ell,i)}$, expected route $\overline{\mathsf{rte}}^*$ 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* or is a *non-inversion honest wire*, 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 $(x, \overline{\mathsf{rte}}, \mathsf{msig}) = \mathsf{CT}_{(\ell,i)} \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell,i)}, t^*)$.
    - Step ii: If $x \neq x^*_{(\ell,i)}$ or $\overline{\mathsf{rte}} \neq \overline{\mathsf{rte}}^*$ 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 $\mathsf{msig}' = \mathsf{Sig}.\mathbf{Sign}(\mathsf{mpp}_{(\ell+1,j)}, \mathsf{msk}_{(\ell+1,j)}, t^*, (x^*_{(\ell,i)}, \overline{\mathsf{rte}}^*))$ and $\mathsf{CT}^*_{(\ell+1,j)} = (x^*_{(\ell,i)}, \overline{\mathsf{rte}}^*, \mathsf{msig}') \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,j)}, t)$.
    - Step iii: If $\ell = L-1$, compute $\mathsf{CT}^*_{(L,j)} = (x^*_{(\ell,i)}, \bot, \bot) \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(L,j)}, t)$.
- 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 $x = \bot_{\mathsf{filler}}$ and $\overline{\mathsf{rte}} = \bot_{\mathsf{filler}}$.
  - 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 8: 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 during $\mathbf{Setup}^*$, 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/inversion* input wires for the challenge round t^*. In other words, for each *filler/inversion* input wire (ℓ, i) that is on the route $\overline{\mathsf{rte}}^*$, do the following.

- The challenger hardcodes the expected input ciphertext $\overline{\mathsf{CT}}^*_{(\ell,i)} = (x^*_{(\ell,i)}, \overline{\mathsf{rte}}^*, \mathsf{msig}^*_{(\ell,i)}) \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell,i)}, t^*)$ 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 9.

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/inversion* 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 $\overline{\mathsf{rte}}^*$, 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* or is a *non-inversion honest wire*, 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 9: 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 during **Setup***, the challenger punctures the message signing keys for all *filler/inversion* wires $(\ell+1, i)$ on the path $\overline{\mathsf{rte}}^* = \overline{\mathsf{rte}}^*_u$ (or $\overline{\mathsf{rte}}^* = \bot_{\mathsf{filler}}$ in case of *filler* wire) at the challenge round t^* and challenge message $x^*_{(\ell+1,i)} = x^{(b)}_{u,t^*}$ (or $x^*_{(\ell+1,i)} = \bot_{\mathsf{filler}}$ in case of *filler* wire), that is, for each such wire $(\ell+1, i)$,

$$(\mathsf{msk}_{(\ell+1,i)}, \mathsf{msk}'_{(\ell+1,i)}, \mathsf{mvk}_{(\ell+1,i)}, \mathsf{mpp}_{(\ell+1,i)}) \leftarrow \mathsf{Sig}.\mathbf{PuncturedSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}_m, t^*, (x^*_{(\ell+1,i)}, \overline{\mathsf{rte}}^*_u)).$$

Consequently, the associated message signatures in the gates will be computed using $\mathsf{Sig}.\mathbf{PSign}$. Formally, the behavior of gate $\mathsf{Gate}_{(\ell,g)}$ for all $g \in [G]$ is changed as described in Figure 10.

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 during **Setup***, the challenger uses the binding setup to bind message signature keys for all *filler/inversion* wires $(\ell+1, i)$ on the path $\overline{\mathsf{rte}}^*$ at the challenge round t^* and challenge message $x^*_{(\ell,i)} = x^{(b)}_{u,t^*}$ (or $x^*_{(\ell,i)} = \bot_{\mathsf{filler}}$ in case of *filler* wire), that is, for each such wire $(1, i)$,

$$(\mathsf{msk}^*_{(\ell+1,i)}, \mathsf{mvk}^*_{(\ell+1,i)}, \mathsf{mpp}^*_{(\ell+1,i)}) \leftarrow \mathsf{Sig}.\mathbf{BindingSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}_m, t^*, (x^*_{(\ell,i)}, \overline{\mathsf{rte}}^*)).$$

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 11 and Figure 12.

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/inversion* 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* or is a *non-inversion honest wire*, 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 $\mathsf{msig}' = \mathsf{Sig}.\mathbf{PSign}(\mathsf{mpp}_{(\ell+1,j)}, \mathsf{msk}'_{(\ell+1,j)}, t^*, (x^{(b)}_{u,t^*}, \overline{\mathsf{rte}}^*_u))$ and $\mathsf{CT}^*_{(\ell+1,j)} = (x^{(b)}_{u,t^*}, \overline{\mathsf{rte}}^*_u, \mathsf{msig}') \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,j)}, t)$.
- 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 $x = \perp_{\mathsf{filler}}$ and $\overline{\mathsf{rte}} = \perp_{\mathsf{filler}}$.
  - Step (b): Compute $\mathsf{msig}' = \mathsf{Sig}.\mathbf{PSign}(\mathsf{mpp}_{(\ell+1,j)}, \mathsf{msk}'_{(\ell+1,j)}, t^*, (x, \overline{\mathsf{rte}}))$.
  - Step (c): Compute $\mathsf{CT}^*_{(\ell+1,j)} \leftarrow (x, \overline{\mathsf{rte}}, \mathsf{msig}') \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,j)}, t^*)$.
- Step 3 is same as in $\mathsf{Hyb}^{(b)}_{6,\ell,2}$.

Figure 10: 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/inversion* wire $i \in \mathsf{Output}_{(\ell,g)}$, message public parameter $\mathsf{mpp}^*_{(\ell+1,i)}$ and message signing key $\mathsf{msk}^*_{(\ell+1,i)}$ are hardcoded instead of $\mathsf{mpp}_{(\ell+1,i)}, \mathsf{msk}'_{(\ell+1,i)}$.

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

Figure 11: 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/inversion* wire $i \in \mathsf{Input}_{(\ell+1,g)}$, message public parameter $\mathsf{mpp}^*_{(\ell+1,i)}$ and message verification key $\mathsf{mvk}^*_{(\ell+1,i)}$ are hardcoded instead of $\mathsf{mpp}_{(\ell+1,i)}, \mathsf{mvk}_{(\ell+1,i)}$.

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

Figure 12: 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 during **Setup***, when generating all circuits for layer ℓ, for all *filler/inversion* wires $(\ell+1,i)$ on the path $\overline{\mathsf{rte}}^*$ the challenger hardcodes the expected output ciphertexts for the challenge round t^*: $\overline{\mathsf{CT}}^*_{(\ell+1,i)} = (x^{(b)}_{u,t^*}, \overline{\mathsf{rte}}^*_u, \mathsf{msig}^*_{(\ell+1,i)}) \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,i)}, t^*)$ if $\ell < L-1$, else $\overline{\mathsf{CT}}^*_{(L,i)} = (x^{(b)}_{u,t^*}, \bot, \bot) \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(L,i)}, t^*)$. 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 inversion 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 inversion wires 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 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. Same as $\mathsf{Hyb}^{(b)}_{6,\ell,4}$ except that for each *filler/inversion* 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* or is a *non-inversion honest wire*, compute as in $\mathsf{Hyb}^{(b)}_{6,\ell,5}$. Else:
   (a) If $\ell = 1$ and i is even, continue to next i. // *It 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/inversion* 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 13: 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 during **Setup***, the challenger changes the hardcoded ciphertexts as follows: In gates $\mathsf{Gate}_{(L-1,g)}$ for all $g \in [G]$, for all *inversion* 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 *inversion* output wire whose corresponding receiver is *honest*, or if it is *filler* wire $i \in \mathsf{Output}_{(L-1,g)}$, or if it *filler/inversion* 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 a random string of appropriate length. Further, the challenge ciphertexts given out to the adversary for $u \in \{s, s'\}$ are set in a manner consistent with the hardcoded input ciphertexts in layer $\ell = 1$.

Hybrid $\mathsf{Hyb}^{(b)}_8$: This hybrid is same as $\mathsf{Hyb}^{(b)}_7$ except that during **Setup***, the challenger unbinds all the message signature tuples that were previously binding: for all the message signature tuples for all *filler/inversion* wires (ℓ, i) on the path $\overline{\mathsf{rte}}^*$ in all the layers, the challenger uses punctured setup instead of binding setup at the challenge round t^* and challenge message $x^*_{(L,i)} = x^{(b)}_{u,t^*}$ (or $x^*_{(L,i)} = \bot_{\mathsf{filler}}$ in case of *filler* wire), that is, for each such wire (ℓ, i),

$$(\mathsf{msk}_{(\ell,i)}, \mathsf{msk}'_{(\ell,i)}, \mathsf{mvk}_{(\ell,i)}, \mathsf{mpp}_{(\ell,i)}) \leftarrow \mathsf{Sig}.\mathbf{PuncturedSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}_m, t^*, (x^*_{(L,i)}, \overline{\mathsf{rte}}^*)).$$

Hybrid $\mathsf{Hyb}^{(b)}_9$: This hybrid is same as $\mathsf{Hyb}^{(b)}_8$ except that during **Setup***, the challenger unpunctures all the message signing keys that were previously punctured: for all the message signature key pairs for all *filler/inversion* wires (ℓ, i) on the path $\overline{\mathsf{rte}}^*$ in all the layers, the challenger uses setup algorithm and does not puncture anymore at the challenge round t^* and challenge message $x^*_{(L,i)}$,

that is, for each such wire (ℓ, i),

$$(\mathsf{msk}_{(\ell,i)}, \mathsf{mvk}_{(\ell,i)}, \mathsf{mpp}_{(\ell,i)}) \leftarrow \mathsf{Sig}.\mathbf{Setup}(1^\lambda, \mathsf{tlen}, \mathsf{len}_m).$$

Then, whenever the challenger has to compute a message signature during **Enc** algorithm or inside a gate, it computes as: $\mathsf{msig}_{(\ell,i)} = \mathsf{Sig}.\mathbf{Sign}(\mathsf{mpp}_{(\ell,i)}, \mathsf{msk}_{(\ell,i)}, \cdot, \cdot)$.

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 14 and Figure 15.

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 public parameter $\mathsf{mpp}_{(\ell,i)}$, the message verification key $\mathsf{mvk}_{(\ell,i)}$, the route public parameter $\mathsf{rpp}^*_{(\ell,i)}$, the route verification key $\mathsf{rvk}^*_{(\ell,i)}$, the expected route $\overline{\mathsf{rte}}^*_u$.
  - if *filler/inversion*: the PRF key $k^*_{(\ell,i)}$, the message public parameter $\mathsf{mpp}_{(\ell,i)}$, the message verification key $\mathsf{mvk}_{(\ell,i)}$, the route public parameter $\mathsf{rpp}_{(\ell,i)}$, 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 public parameter $\mathsf{mpp}_{(\ell+1,i)}$. the message signing key $\mathsf{msk}_{(\ell+1,i)}$.
  - if *filler/inversion*: the PRF key $k^*_{(\ell+1,i)}$, the message public parameter $\mathsf{mpp}_{(\ell+1,i)}$, the message signing key $\mathsf{msk}_{(\ell+1,i)}$, the expected challenge ciphertext $\overline{\mathsf{CT}}^*_{(\ell+1,i)}$.
- The challenge round t^*.

Figure 14: 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 is even, continue to next i. // *It is a filler wire and is ignored.*
   (b) **Decrypt and authenticate the message/route:**
   If it is a *filler/inversion* wire and $t = t^*$: If $\mathsf{CT}^*_{(\ell,i)} \neq \overline{\mathsf{CT}}^*_{(\ell,i)}$, then abort.
   If it is a (*corrupt* wire or is a *non-inversion honest wire*) or (*any* wire and $t \neq t^*$):
      i. Compute the plaintext $(x, \overline{\mathsf{rte}}, \mathsf{msig}) = \mathsf{CT}_{(\ell,i)} \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell,i)}, t)$.
      ii. If msig is not a valid signature of $(x, \overline{\mathsf{rte}})$ w.r.t. $\mathsf{mpp}_{(\ell,i)}, \mathsf{mvk}_{(\ell,i)}$ and round t, then abort.
      iii. If $x = \bot_{\mathsf{filler}}$ and $\overline{\mathsf{rte}} = \bot_{\mathsf{filler}}$, go to the next i. // *It is a filler wire and is ignored.*
      iv. Parse $\overline{\mathsf{rte}}$ as $(\mathsf{rte} = (j_1, \ldots, j_L), \mathsf{rsig} = (\mathsf{rsig}_1, \ldots, \mathsf{rsig}_L))$ and perform the following checks to authenticate the route $\overline{\mathsf{rte}}$:
      If it is a *corrupt* wire: If $\mathsf{rte} \neq \mathsf{rte}^*_u$, abort. If $\mathsf{rsig}_\ell \neq \mathsf{rsig}^*_\ell$, abort.
      If it is an *honest* wire and $t \neq t^*$: Parse $\overline{\mathsf{rte}}$ as $(\mathsf{rte} = (j_1, \ldots, j_L), \mathsf{rsig} = (\mathsf{rsig}_1, \ldots, \mathsf{rsig}_L))$ and perform the following checks to authenticate the route $\overline{\mathsf{rte}}$: 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. $\mathsf{rpp}_{(\ell,i)}$ and $\mathsf{rvk}_{(\ell,i)}$, 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 $\mathsf{msig}' = \mathsf{Sig}.\mathbf{Sign}(\mathsf{mpp}_{(\ell+1,j)}, \mathsf{msk}_{(\ell+1,j)}, t, (x, \overline{\mathsf{rte}}^*_u))$ and $\mathsf{CT}_{(\ell+1,j)} = (x, \overline{\mathsf{rte}}^*_u, \mathsf{msig}') \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,j)}, t)$.
      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 $\mathsf{msig}' = \mathsf{Sig}.\mathbf{Sign}(\mathsf{mpp}_{(\ell+1,j)}, \mathsf{msk}_{(\ell+1,j)}, t, (x, \overline{\mathsf{rte}}))$ and $\mathsf{CT}_{(\ell+1,j)} \leftarrow (x, \overline{\mathsf{rte}}, \mathsf{msig}') \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,j)}, t)$.
      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/inversion* ciphertexts as $\mathsf{CT}^*_{(\ell+1,j)} = \overline{\mathsf{CT}}^*_{(\ell+1,j)}$.
   If $t \neq t^*$, compute *filler* ciphertexts:
   (a) Set $x = \bot_{\mathsf{filler}}$ and $\overline{\mathsf{rte}} = \bot_{\mathsf{filler}}$.
   (b) Compute $\mathsf{msig}' = \mathsf{Sig}.\mathbf{Sign}(\mathsf{mpp}_{(\ell+1,j)}, \mathsf{msk}_{(\ell+1,j)}, t, (x, \overline{\mathsf{rte}}))$.
   (c) Compute $\mathsf{CT}_{(\ell+1,j)} \leftarrow (x, \overline{\mathsf{rte}}, \mathsf{msig}') \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,j)}, t)$.
3. Output $\{\mathsf{CT}_{(\ell+1,i)} : i \in \mathsf{Output}_{(\ell,g)}\}$.

Figure 15: The circuit $\mathsf{Gate}_{(\ell,g)}$ in hybrid experiment $\mathsf{Hyb}^{(b)}_9$: procedure.

E.2 Proofs of Indistinguishability

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

Claim E.3. *For $b \in \{0,1\}$, assuming correctness of the SSU signatures scheme, 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 5 that the input/output behaviour of these two algorithms is identical for all unpunctured points. As these signatures are generated by the challenger for some unpunctured points, hence, adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$ are identical. □

Claim E.4. *For $b \in \{0,1\}$, assuming the SSU signature scheme satisfies computational indistinguishability of punctured and binding setups, 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 signature key tuple $(\mathsf{rsk}'_{(\ell,i)}, \mathsf{rvk}_{(\ell,i)}, \mathsf{rpp}_{(\ell,i)})$ is used where the signing key is punctured and the verification key and public parameter are unpunctured. Whereas in hybrid $\mathsf{Hyb}_2^{(b)}$, for all the *corrupt* wires, the route signature key tuple $(\mathsf{rsk}^*_{(\ell,i)}, \mathsf{rvk}^*_{(\ell,i)}, \mathsf{rpp}^*_{(\ell,i)})$ is used where all the keys are binding and are generated using binding 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 signature key tuple $(\mathsf{rsk}^*_{w_j}, \mathsf{rvk}^*_{w_j}, \mathsf{rpp}^*_{w_j})$ is used if $j \leq i$, else the pair $(\mathsf{rsk}'_{w_j}, \mathsf{rvk}_{w_j}, \mathsf{rpp}_{w_j})$ 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 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 signature key tuple $(\mathsf{rsk}'_{w_i}, \mathsf{rvk}_{w_i}, \mathsf{rpp}_{w_i})$ is used, whereas in $\mathsf{H}_{1,i}^{(b)}$, the route signature key tuple $(\mathsf{rsk}^*_{w_i}, \mathsf{rvk}^*_{w_i}, \mathsf{rpp}^*_{w_i})$ is used. Then, we can create a simple reduction from the computational indistinguishability of the two hybrids to the computational indistinguishability of punctured and binding setup of SSU signature scheme which states that the following two distibutions are computationally indistinguishable.

1. Let $(\mathsf{rsk}_{w_i}, \mathsf{rsk}'_{w_i}, \mathsf{rvk}_{w_i}, \mathsf{rpp}_{w_i}) \leftarrow \mathsf{Sig}.\mathbf{PuncturedSetup}(1^\lambda, 0, \mathsf{len}_{\mathsf{rte}}, 1, \mathsf{rte}^*_u)$, and output $(\mathsf{rsk}'_{w_i}, \mathsf{rvk}_{w_i}, \mathsf{rpp}_{w_i})$.
2. Let $(\mathsf{rsk}^*_{w_i}, \mathsf{rvk}^*_{w_i}, \mathsf{rpp}^*_{w_i}) \leftarrow \mathsf{Sig}.\mathbf{BindingSetup}(1^\lambda, 0, \mathsf{len}_{\mathsf{rte}}, 1, \mathsf{rte}^*_u)$, and output $(\mathsf{rsk}^*_{w_i}, \mathsf{rvk}^*_{w_i}, \mathsf{rpp}^*_{w_i})$.

□

Claim E.5. *For $b \in \{0,1\}$, assuming that the SSU signature scheme is deterministic and statistically unforgeable at the binding point 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 $\overline{\mathsf{rte}}_u = (\mathsf{rte}_u = (j_1, \ldots, j_L), \mathsf{rsig}_u = (\mathsf{rsig}_1, \ldots, \mathsf{rsig}_L))$ for some user $u \in \mathcal{K}_S$), the route authentication in hybrid $\mathsf{Hyb}_2^{(b)}$ is performed by checking

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

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

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

where the expected route for this wire $\overline{\mathsf{rte}}^*_u = (\mathsf{rte}^*_u = (j^*_1, \ldots, j^*_L), \mathsf{rsig}^* = (\mathsf{rsig}^*_1, \ldots, \mathsf{rsig}^*_L))$ 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 $\mathsf{rpp}^*_{(\ell,i)}$ and $\mathsf{rvk}^*_{(\ell,i)}$ are binding route public parameter and binding route verification key and satisfy statistical unforgeability at round 1 and value $(j^*_1, \ldots, j^*_L)$ as defined in Definition 5.1. 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 E.6. *For $b \in \{0,1\}$, assuming correctness of the SSU signatures scheme, 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/inversion* 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 $\tilde{x}^* = (x^{(b)}_{u,t^*}, \overline{\mathsf{rte}}^{(b)}_u)$ in case of *inversion* wire for some $u \in \mathcal{H}_S$ or $\tilde{x}^* = (\bot_{\mathsf{filler}}, \bot_{\mathsf{filler}})$ in case of *filler* wire. But, notice that none of these message signing keys are in the view of the adversary. The only difference then is the ciphertexts that the adversary obtains from the challenger for the *inversion* wires in the first layer. In $\mathsf{Hyb}_3^{(b)}$, the **Enc** algorithm internally uses $\mathsf{Sig}.\mathbf{Sign}$ to compute the message signatures, whereas in $\mathsf{Hyb}_4^{(b)}$, the **Enc** algorithm internally uses $\mathsf{Sig}.\mathbf{PSign}$ to compute the message signatures. It follows from the correctness of the SSU signature scheme as defined in Section 5 that the input/output behaviour of these two algorithms is identical for all unpunctured points. As these signatures are generated by the challenger for some unpunctured points, hence, it follows that the adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_3^{(b)}$ and $\mathsf{Hyb}_4^{(b)}$ are identical.

□

Claim E.7. *For $b \in \{0,1\}$, assuming the SSU signature scheme satisfies computational indistinguishability of punctured and binding setups, 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/inversion* wires $(1,i)$, the message signature key tuple $(\mathsf{msk}'_{(1,i)}, \mathsf{mvk}_{(1,i)}, \mathsf{mpp}_{(1,i)})$ is used where the signing key is punctured and the verification key and public parameter are unpunctured. Whereas in hybrid $\mathsf{Hyb}_5^{(b)}$, for all the *filler/inversion* wires, the message signature key tuple $(\mathsf{msk}^*_{(1,i)}, \mathsf{mvk}^*_{(1,i)}, \mathsf{mpp}^*_{(1,i)})$ is used where all the keys are binding and are generated using binding setup. Suppose that there are $\eta < 2n$ *filler/inversion* wires in the first layer $\ell = 1$. For the sake of simplicity, call these η wires to be $w_1, \ldots, w_\eta$. 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 $\mathsf{H}_{4,0}^{(b)}, \ldots, \mathsf{H}_{4,\eta}^{(b)}$ where in $\mathsf{H}_{4,i}$, for wire w_j, the message signature key tuple $(\mathsf{msk}^*_{w_j}, \mathsf{mvk}^*_{w_j}, \mathsf{mpp}^*_{w_j})$ is used if $j \leq i$, else the tuple $(\mathsf{msk}'_{w_j}, \mathsf{mvk}_{w_j}, \mathsf{mpp}_{w_j})$ 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 signature key tuple for wire w_i. If wire w_i is an *inversion* wire, then, the challenge plaintext is $\tilde{x}^* = (x^{(b)}_{u,t^*}, \overline{\mathsf{rte}}^{(b)}_u)$ for some user $u \in \mathcal{H}_S$. Else, if it is a *filler* wire, then, the challenge plaintext is $\tilde{x}^* = (\bot_{\mathsf{filler}}, \bot_{\mathsf{filler}})$. In $\mathsf{H}_{4,i-1}^{(b)}$, the message signature key tuple $(\mathsf{msk}'_{w_i}, \mathsf{mvk}_{w_i}, \mathsf{mpp}_{w_i})$ is used, whereas in $\mathsf{H}_{4,i}^{(b)}$, the message signature key tuple $(\mathsf{msk}^*_{w_i}, \mathsf{mvk}^*_{w_i}, \mathsf{mpp}^*_{w_i})$ is used. In both the above hybrids the binding 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 punctured and binding setups of SSU signature scheme which states that the following two distibutions are computationally indistinguishable.

1. Let $(\mathsf{msk}_{w_i}, \mathsf{msk}'_{w_i}, \mathsf{mvk}_{w_i}, \mathsf{mvk}_{w_i}) \leftarrow \mathsf{Sig}.\mathbf{PuncturedSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}_m, t^*, \tilde{x}^*)$, and output $(\mathsf{msk}'_{w_i}, \mathsf{mvk}_{w_i}, \mathsf{mvk}_{w_i})$.
2. Let $(\mathsf{msk}^*_{w_i}, \mathsf{mvk}^*_{w_i}, \mathsf{mpp}^*_{w_i}) \leftarrow \mathsf{Sig}.\mathbf{BindingSetup}(1^\lambda, \mathsf{tlen}, \mathsf{len}_m, t^*, \tilde{x}^*)$, and output $(\mathsf{msk}^*_{w_i}, \mathsf{mvk}^*_{w_i}, \mathsf{mpp}^*_{w_i})$.

□

Claim E.8. *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 binding 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/inversion* 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 $\overline{\mathsf{rte}}$ is equal to the hardcoded route $\overline{\mathsf{rte}}^*$.

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

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 $\mathsf{Hyb}'_{2n} = \mathsf{Hyb}^{(b)}_{6,\ell,1}$. 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 $(x_{(\ell,\alpha)}, \overline{\mathsf{rte}}, \mathsf{msig}_{(\ell,\alpha)})$ with respect to round t^* which was not generated honestly. It is clear that in Hyb'_α, because the *filler/inversion* message $x^*_{(\ell,i)}$, route $\overline{\mathsf{rte}}^*$ 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 key tuple $(\mathsf{msk}^*_{(\ell,\alpha)}, \mathsf{mvk}^*_{(\ell,\alpha)}, \mathsf{mpp}^*_{(\ell,\alpha)})$ which was generated by $\mathsf{Sig}.\mathbf{BindingSetup}$, the only message that is accepted by $\mathsf{Sig}.\mathbf{Vf}$ is $(x^*_{(\ell,i)}, \overline{\mathsf{rte}}^*)$, 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 E.9. *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 inversion/filler wire keys are punctured at t^*, and during round t^* inversion/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 $\alpha \in \{1, \ldots, 2n\}$.

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 $i < \alpha$, if this is an inversion/filler wire, hardcode punctured key $k^*_{(\ell,i)}$. Treat input inversion/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 inversion/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 $\mathsf{Hyb}'_{2n} = \mathsf{Hyb}^{(b)}_{6,\ell,2}$. 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 E.10. *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 inversion/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 $\alpha \in \{1, \ldots, 2n\}$.

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 $i < \alpha$, if wire i is an inversion/filler wire, 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 $\mathsf{Hyb}'_{2n} = \mathsf{Hyb}^{(b)}_{6,\ell,3}$. 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 binding message. □

Claim E.11. *For $b \in \{0,1\}$ and $\ell \in [L-1]$, assuming the SSU signature scheme satisfies computational indistinguishability of punctured and binding setups, 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 signature key tuples for inversion/filler wires $(\ell+1,i)$, $i \in [2n]$ are all generated using $\mathsf{Sig}.\mathbf{BindingSetup}$, whereas in $\mathsf{Hyb}^{(b)}_{6,\ell,3}$ they are generated using $\mathsf{Sig}.\mathbf{PuncturedSetup}$.

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

We define Hyb'_α to generate message signature key tuples $(\mathsf{msk}_{(\ell+1,i)}, \mathsf{mvk}_{(\ell+1,i)}, \mathsf{mpp}_{(\ell+1,i)})$ using $\mathsf{Sig}.\mathbf{BindingSetup}$ for all inversion/filler $i < \alpha$, and to generate $(\mathsf{msk}_{(\ell+1,i)}, \mathsf{mvk}_{(\ell+1,i)}, \mathsf{mpp}_{(\ell+1,i)})$ using $\mathsf{Sig}.\mathbf{PuncturedSetup}$ for all $i \geq \alpha$. It is clear that $\mathsf{Hyb}'_1 = \mathsf{Hyb}^{(b)}_{6,\ell,3}$ and that $\mathsf{Hyb}'_{2n} = \mathsf{Hyb}^{(b)}_{6,\ell,4}$. 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 key tuple $(\mathsf{msk}_{(\ell+1,\alpha-1)}, \mathsf{mvk}_{(\ell+1,\alpha-1)}, \mathsf{mpp}_{(\ell+1,\alpha-1)})$. As such, a simple reduction to computational indistinguishability of the punctured and binding setups of the SSU signature scheme shows this indistinguishability. This proves the claim.

□

Claim E.12. *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 inversion/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 $\overline{\mathsf{CT}}^*_{(\ell+1,i)} = (x^{(b)}_{u,t^*}, \overline{\mathsf{rte}}^*_u, \mathsf{msig}^*_{(\ell+1,i)}) \oplus \mathsf{PRF}.\mathbf{Eval}(k_{(\ell+1,i)}, t^*)$ 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 $\alpha \in \{1,\ldots,2n\}$.

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/inversion 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 $\mathsf{Hyb}'_{2n} = \mathsf{Hyb}^{(b)}_{6,\ell,5}$. 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/inversion* wire in layer $\ell+1$ because of the hardwired routes for *corrupt* wires in hybrid $\mathsf{Hyb}^{(b)}_3$ (Figure 6).

- During round t^*, the plaintext $(x, \overline{\mathsf{rte}}, \mathsf{msig}')$ encrypted by $\mathsf{Gate}_{(\ell,g)}$ to form $\mathsf{CT}_{(\ell+1,\alpha-1)}$ in $\mathsf{Hyb}'_{\alpha-1}$ has values x and $\overline{\mathsf{rte}}$ 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 E.13. *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 (i) how are the hardcoded ciphertexts for the challenge round t^* for all the *filler/inversion* wires in all the layers (with the exception of *inversion* output wires directed towards corrupt receivers in the last layer) computed by the challenger, and (ii) the challenge ciphertexts for $u \in \{s, s'\}$ given to the adversary for the challenge round t^*. 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 $\mathsf{msig}^*_{(\ell,i)} = \mathsf{Sig}.\mathbf{PSign}(\mathsf{mpp}^*_{(\ell,i)}, \mathsf{msk}^*_{(\ell,i)}, t^*, \tilde{x}^*)$ and $y^* = \mathsf{PRF}.\mathbf{Eval}(k_{(\ell,i)}, t^*)$. If it is an *inversion* wire corresponding to route $\overline{\mathsf{rte}}^{(b)}_u$ for some $u \in \mathcal{H}_S$, then, $\tilde{x}^* = (x^{(b)}_{u,t^*}, \overline{\mathsf{rte}}^{(b)}_u)$. Else, if it is a *filler* wire, then, $\tilde{x}^* = (\bot_{\mathsf{filler}}, \bot_{\mathsf{filler}})$. On the other hand, in $\mathsf{Hyb}^{(b)}_{7}$, for a *filler/inversion* wire (ℓ, i), $\overline{\mathsf{CT}}^*_{(\ell,i)} = \mathit{random}$. Further, in both hybrids $\mathsf{Hyb}^{(b)}_{6,L-1,5}$ and $\mathsf{Hyb}^{(b)}_{7}$, the challenge ciphertexts for $u \in \{s, s'\}$ given to the adversary for the challenge round t^* are set consistent with the hardcoded incoming ciphertexts in the obfuscated gates in layer $\ell = 1$ in those respective hybrids. 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/inversion* wire (ℓ, i), we want to change from $\overline{\mathsf{CT}}^*_{(\ell,i)} = (\tilde{x}^*, \mathsf{msig}^*_{(\ell,i)}) \oplus y^*$ to $\overline{\mathsf{CT}}^*_{(\ell,i)} = \mathit{random}$, where $y^* = \mathsf{PRF}.\mathbf{Eval}(k_{(\ell,i)}, t^*)$. We first invoke the psuedorandomness of PRF at punctured points to change to $y^* = \mathit{random}'$. Then, we can change from $\overline{\mathsf{CT}}^*_{(\ell,i)} = (\tilde{x}^*, \mathsf{msig}^*_{(\ell,i)}) \oplus \mathit{random}'$ to $\overline{\mathsf{CT}}^*_{(\ell,i)} = \mathit{random}$ as both are identically distributed. This is exactly what we want in $\mathsf{Hyb}^{(b)}_{7}$.

□

Claim E.14. *For $b \in \{0,1\}$, assuming the SSU signature scheme satisfies computational indistinguishability of punctured and binding setups, adversary $\mathcal{A}$'s views in $\mathsf{Hyb}^{(b)}_{7}$ and $\mathsf{Hyb}^{(b)}_{8}$ are computationally indistinguishable.*

Proof. Similar to Claim E.7, except that here the message verification keys and message public parameters are changed from binding to unbinding for all the *filler/inversion* wires (ℓ, i) in all the layers.

□

Claim E.15. *For $b \in \{0,1\}$, assuming the SSU signatures scheme is correct, and 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/inversion* 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. This difference reflects in how message signatures are computed in all the circuit

$\mathsf{Gate}_{(\ell,g)}$ and by the **Enc** algorithm for ciphertexts corresponding to *inversion* wires in layer 1. For ciphertexts corresponding to *inversion* wires in layer 1, identical distibution can be proven similar to Claim E.6 assuming correctness of SSU signatures.

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 $\mathsf{Hyb}_9^{(b)}$, $\mathsf{msk}_{(\ell,i)}$ 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 E.16. *Adversary $\mathcal{A}$'s views in $\mathsf{Hyb}_9^{(0)}$ and $\mathsf{Hyb}_9^{(1)}$ are identical.*

Proof. Note that the ciphertexts received by $\mathcal{A}$ in round t^* on behalf of s and s' are random values independent of b, and the other ciphertexts received by $\mathcal{A}$ during that round are independent of b by the definition of the hybrids. Recall the formal description of circuits $\mathsf{Gate}_{(\ell,g)}$ described in Figure 14 and Figure 15. It suffices to argue that the router token received by $\mathcal{A}$ in $\mathsf{Hyb}_9^{(b)}$ 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 $i \in \mathsf{Input}_{(\ell,i)}$, route information $\overline{\mathsf{rte}}^*_u$, binding route public parameter $\mathsf{rpp}^*_{(\ell,i)}$ and binding route verification $\mathsf{rvk}^*_{(\ell,i)}$ are hardwired, where the route public parameter and route verification key are binding at route $\overline{\mathsf{rte}}^*_u$. The admissibility criteria requires that for each $u \in \mathcal{K}_S$, $\pi^{(0)}(u) = \pi^{(1)}(u)$. Hence, the hardcoded routes for corrupt senders have no information about the challenge bit b. It also follows then that the binding route public parameters and the binding verification keys for corrupt senders do not contain any information about the challenge bit b.

- For each *filler/inversion* 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/inversion* 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 *inversion* 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 inversion 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. □

F From Static Corruption to Adaptive Corruption

F.1 A Compiler for Upgrading from Static to Adaptive Corruption

In this section, we present a generic compiler that lifts any NIAR scheme that satisfies static-corruption security to a NIAR scheme with *full* security under adaptive corruptions and adaptive queries (Definition 3.1).

Starting from a static corruption NIAR scheme denoted NIAR', we will use an extra pseudorandom function PRF to encrypt the plaintext first, before encrypting it again with NIAR', thus creating *two layers* of encryption. For the security proof, we will need the PRF to be secure against *selective-opening attacks* (Definition B.7), which is implied by the standard security for pseudorandom functions as shown by Abraham et al. [ACD+19]. We formally describe the compiler below.

A compiler from static corruption to adaptive corruption

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

- For all $i \in [n]$: $k_i \leftarrow \mathsf{PRF}.\mathbf{Setup}(1^\lambda)$.
- Let $(\{\mathsf{ek}'_u\}_{u\in[n]}, \{\mathsf{rk}'_u\}_{u\in[n]}, \mathsf{tk}') \leftarrow \mathsf{NIAR}'.\mathbf{Setup}(1^\lambda, \mathsf{len}, n, \pi)$.
- For each sender $u \in [n]$, let $\mathsf{ek}_u := (k_{\pi(u)}, \mathsf{ek}'_u)$; for each receiver $u \in [n]$, let $\mathsf{rk}_u := (k_u, \mathsf{rk}'_u)$; and let $\mathsf{tk} = \mathsf{tk}'$. Output $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk})$.

Enc$(\mathsf{ek}_u, \mathbf{x}_u, t)$**:** *// Parse* $\mathsf{ek}_u := (k'_u, \mathsf{ek}'_u)$

- Output $\mathsf{ct}_{u,t} \leftarrow \mathsf{NIAR}'.\mathbf{Enc}(\mathsf{ek}'_u, \mathsf{PRF}.\mathbf{Eval}(k'_u, t) \oplus \mathbf{x}_u, t)$.

Rte$(\mathsf{tk}, t, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$**:**

- Let $(\mathsf{ct}'_1, \ldots, \mathsf{ct}'_n) \leftarrow \mathsf{NIAR}'.\mathbf{Rte}(\mathsf{tk}, t, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ and output $(\mathsf{ct}'_1, \ldots, \mathsf{ct}'_n)$.

Dec$(\mathsf{rk}_u, \mathsf{ct}'_u, t)$**:** *// Parse* $\mathsf{rk}_u := (k_u, \mathsf{rk}'_u)$

- Let $\mathbf{y} = \mathsf{PRF}.\mathbf{Eval}(k_u, t) \oplus \mathsf{NIAR}'.\mathbf{Dec}(\mathsf{rk}'_u, \mathsf{ct}'_u, t)$, and output $\mathbf{y}$.

Figure 16: A compiler from static corruption to adaptive corruption

Theorem F.1 (Static to adaptive corruption compiler)**.** *Suppose that* NIAR' *satisfies Definition A.1 subject to a static-corruption, single-inversion, and all-receiver-corrupting adverasary, and suppose that* PRF *is a selective-opening secure pseudorandom function (i.e. Definition B.7). Then,* NIAR *satisfies full security (i.e., Definition 3.1).*

Proof roadmap. In the remainder of this section, we prove Theorem F.1. The proof goes through multiple intermediate steps.

1. We start from a NIAR secure w.r.t. inversion under a static-corruption, all-receiver-corrupting adversary. We prove that the same scheme is also secure w.r.t. inversion under an *adaptive*-corruption, all-receiver-corrupting adversary. (see Appendix F.2 and Lemma F.2).
2. Next, given a NIAR scheme secure w.r.t. inversion under an adaptive-corruption, all-receiver-corrupting adversary, we remove the "inversion" restriction, and prove that it is also secure for an arbitrary pair of permutations under an adaptive-corruption, all-receiver-corrupting adversary (see Appendix F.3 and Lemma F.3).
3. Finally, given a NIAR scheme that is secure under an adaptive-corruption, all-receiver-corrupting adversary (for an arbitrary pair of permutations), we show that the above static-to-adaptive compiler gives a full NIAR scheme secure under Definition 3.1 (see Appendix F.4 and Lemma F.4).

F.2 Adaptive Corruption Under Single-Inversion and All-Receiver-Corruption

Lemma F.2. *If a NIAR scheme satisfies security under a static-corruption, single-inversion, and all-receiver-corrupting adversary, then it also satisfies security under a single-inversion adversary who corrupts all receivers upfront, and corrupts senders adaptively.*

Proof. Consider a non-uniform p.p.t. admissible adversary $\mathcal{A}$ who corrupts all receivers upfront and corrupts the senders adaptively. $\mathcal{A}$ is trying to distinguish its views in experiments $\mathsf{NIARFull}^{0,\mathcal{A}}(1^\lambda)$ and $\mathsf{NIARFull}^{1,\mathcal{A}}(1^\lambda)$ subject to single inversion. Then, we construct a p.p.t. admissible adversary $\mathcal{B}$ that plays a game with its own challenger $\mathcal{C}$ and leverages $\mathcal{A}$ to distinguish between $\mathsf{NIARStatic}^{0,\mathcal{B}}(1^\lambda)$ and $\mathsf{NIARStatic}^{1,\mathcal{B}}(1^\lambda)$ subject to single inversion and all-receiver-corruption. The description of $\mathcal{B}$ is as follows.

1. $\mathcal{B}$ gets $(n, \mathsf{len}, \pi^{(0)}, \pi^{(1)})$ from $\mathcal{A}(1^\lambda)$. Let s_1, s_2 be the two senders involved in the inversion. For its game with $\mathcal{C}$, it chooses to corrupt the senders $\mathcal{K}'_S = [n] \setminus \{s_1, s_2\}$, the receivers $\mathcal{K}'_R = [n]$, and it sends $(n, \mathsf{len}, \mathcal{K}'_S, \mathcal{K}'_R, \pi^{(0)}, \pi^{(1)})$ to $\mathcal{C}$. Then, $\mathcal{C}$ computes $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{len}, n, \pi^{(b)})$ and gives the terms $(\mathsf{tk}, \{\mathsf{ek}_u\}_{u\in\mathcal{K}'_S}, \{\mathsf{rk}_u\}_{u\in[n]})$ to $\mathcal{B}$. $\mathcal{B}$ now passes $(\mathsf{tk}, \{\mathsf{rk}_u\}_{u\in[n]})$ to $\mathcal{A}$ since $\mathcal{A}$ always corrupts all receivers upfront.
2. In each time step t:
   - Upon receiving an encryption query $\{x^{(0)}_{u,t}, x^{(1)}_{u,t}\}_{u\in\mathcal{H}_S}$ from $\mathcal{A}$ where $\mathcal{H}_S$ is the set of senders $\mathcal{A}$ has not corrupted yet, $\mathcal{B}$ sends the encryption query $\{x^{(0)}_{u,t}, x^{(1)}_{u,t}\}_{u\in\{s_1,s_2\}}$ to $\mathcal{C}$, and gets back ciphertexts $\{\mathsf{ct}_{u,t}\}_{u\in\{s_1,s_2\}}$. For every $u \in \mathcal{H}_S\setminus\{s_1, s_2\}$, by the admissibility rule on $\mathcal{A}$, it must be that $x^{(0)}_{u,t} = x^{(1)}_{u,t}$; thus, $\mathcal{B}$ computes $\mathsf{ct}_{u,t} = \mathbf{Enc}(\mathsf{ek}_u, x^{(0)}_{u,t}, t)$ on its own since it knows ek_u. Now, $\mathcal{B}$ sends $\{\mathsf{ct}_{u,t}\}_{u\in\mathcal{H}_S}$ to $\mathcal{A}$.
   - Upon receiving a corruption query to corrupt some sender $u \in [n]$, by the admissibility rule on $\mathcal{A}$, u cannot be s_1 or s_2. Therefore, $\mathcal{B}$ returns ek_u to $\mathcal{A}$, as well as all random coins used in earlier encryption queries involving u.
3. $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Observe that if $\mathcal{A}$ is admissible and respects single inversion, then, $\mathcal{B}$ is also admissible and respects single inversion. By construction, $\mathcal{B}$ respects all receiver corruption. Further, for $b \in \{0, 1\}$, when $\mathcal{C}$ uses the challenge bit b, $\mathcal{A}$'s view is identical to $\mathsf{NIARFull}^{b,\mathcal{A}}(1^\lambda)$. Therefore, if $\mathcal{A}$ has non-negligible advantage in distinguishing $\mathsf{NIARFull}^{0,\mathcal{A}}(1^\lambda)$ and $\mathsf{NIARFull}^{1,\mathcal{A}}(1^\lambda)$, then $\mathcal{B}$ has non-negligible advantage in distinguishing $\mathsf{NIARStatic}^{0,\mathcal{B}}$ and $\mathsf{NIARStatic}^{1,\mathcal{B}}$. □

F.3 Removing the Single Inversion Restriction

We next remove the single inversion restriction. The proof is very similar to that of Lemma D.6 where we removed the single inversion restriction but for the static corruption scenario.

Lemma F.3 (Removing the single inversion restriction (adaptive corruption))**.** *Suppose we are given a NIAR scheme that satisfies security subject to a single-inversion adversary who always corrupts all receivers upfront, and corrupts senders adaptively. Then, the same scheme is also secure for an arbitrary pair of permutations, subject to an adversary who always corrupts all receivers upfront, and corrupts senders adaptively.*

Proof. The proof is almost identical to that of Lemma D.6, except that now, to prove each pair of adjacent hybrids indistinguishable, the reduction $\mathcal{B}$ also has to answer $\mathcal{A}$'s adaptive sender corruption queries. $\mathcal{B}$ can simply forward such queries to its own challenger and forward back the answers. By $\mathcal{A}$'s admissibility rule, $\mathcal{A}$ can never corrupt a sender that is in $\overline{C}(\pi^{(0)}, \pi^{(1)})$. Therefore, in the adaptive corruption case, it still holds that if $\mathcal{A}$ is admissible, then $\mathcal{B}$ is admissible too. □

F.4 Removing the All Receiver Corruption Restriction

Finally, given a NIAR scheme that is secure under an adversary who corrupts all receivers upfront but can adaptively corrupt senders, we can remove the all-receiver-corruption restriction with the compiler described in F.1.

Lemma F.4 (Removing the all receiver corruption restriction)**.** *Suppose that* NIAR' *satisfies security under an adversary who corrupts all receivers upfront but can adaptively corrupt senders. Then, the compiled* NIAR *scheme described in F.1 satisfies full NIAR security (Definition 3.1).*

Proof. We first define what is a conspicuously honest receiver.

Conspicuously honest receiver. Consider the experiment $\mathsf{NIARFull}^{b,\mathcal{A}}$. Recall that $\mathcal{A}$ submits two permutations $\pi^{(0)}$ and $\pi^{(1)}$ upfront. Consider some receiver j, and let s_0 and s_1 be the corresponding senders to j in $\pi^{(0)}$ and $\pi^{(1)}$, respectively, that is, for $b \in \{0,1\}$, $\pi^{(b)}(s_b) = j$. Suppose during some time step t, the adversary $\mathcal{A}$ submits an encryption query for the plaintext vectors $\{x^{(0)}_{u,t}, x^{(1)}_{u,t}\}_{u \in \mathcal{H}_S}$ where $\mathcal{H}_S$ denotes the so-far honest senders, and it turns out that $x^{(0)}_{s_0,t} \neq x^{(1)}_{s_1,t}$, i.e., j is receiving two different messages in the two worlds. Then, at this moment, we know that the receiver j can never be corrupt by $\mathcal{A}$ due to the admissibility rules on $\mathcal{A}$. Henceforth, such a receiver is called a conspicuously honest receiver. Observe also the corresponding senders s_0 and s_1 (which could be the same) must be conspicuously honest too, by the admissibility rule on $\mathcal{A}$.

Consider the following hybrid experiment indexed by a bit $b \in \{0,1\}$.

Experiment Hyb_b**.** Hyb_b is almost the same as $\mathsf{NIARFull}^b$, except with the following modifications: whenever $\mathcal{A}$ submits an encryption query for the plaintext vectors $\{x^{(0)}_{u,t}, x^{(1)}_{u,t}\}_{u \in \mathcal{H}_S}$ where $\mathcal{H}_S$ denotes the so-far honest senders, the challenger responds with the following ciphertexts for each $u \in \mathcal{H}_S$:

- if $\pi^{(b)}(u)$ is conspicuously honest, then compute $\mathsf{ct}_{u,t} = \mathsf{NIAR}'.\mathbf{Enc}(\mathsf{ek}_u, r, t)$ for a randomly chosen r of appropriate length;
- else compute $\mathsf{ct}_{u,t} = \mathsf{NIAR}.\mathbf{Enc}(\mathsf{ek}_u, x^{(b)}_{u,t}, t)$ honestly.

Claim F.5. *Suppose that the PRF satisfies selective opening security, and let* $b \in \{0,1\}$*. Then, for the compiled* NIAR *scheme, it holds that for any non-uniform p.p.t. admissible adversary* $\mathcal{A}$*, its views in* Hyb_b *and* $\mathsf{NIARFull}^b$ *are computationally indistinguishable.*

Proof. We prove it for $b = 0$, since the case when $b = 1$ is symmetric. Given an efficient $\mathsf{NIARFull}$ adversary $\mathcal{A}$, we will construct an efficient reduction $\mathcal{B}$ that breaks the selective-opening security of the PRF.

- At the beginning $\mathcal{B}$ creates n PRF instances with its own challenger $\mathcal{C}$, one corresponding to each receiver. Except for the PRF keys which $\mathcal{B}$ does not know at the beginning, $\mathcal{B}$ samples all other terms of the NIAR scheme using the honest algorithms.
- Whenever $\mathcal{A}$ makes a corruption query for a receiver $u \in [n]$, $\mathcal{B}$ corrupts the corresponding PRF key with its own challenger $\mathcal{C}$. It then responds to $\mathcal{A}$ with u's secret key in the NIAR scheme.

- Whenever $\mathcal{A}$ makes a corruption query for a sender $u \in [n]$, $\mathcal{B}$ corrupts the corresponding PRF key for the receiver $\pi^{(0)}(u)$ with its own challenger $\mathcal{C}$. It then responds to $\mathcal{A}$ with u's secret key in the NIAR scheme. Additionally, it returns the random coins consumed by u's Enc algorithm in all previous time steps.

- Whenever $\mathcal{A}$ makes an encryption query for a time step t with the challenge plaintexts $\{x^{(0)}_{u,t}, x^{(1)}_{u,t}\}_{u\in\mathcal{H}_S}$, $\mathcal{B}$ computes the answers in the following way: for every $u \in \mathcal{H}_S$,

  - if $\pi^{(0)}(u)$ is conspicuously honest, then $\mathcal{B}$ submits to $\mathcal{C}$ a challenge query for the PRF instance corresponding to receiver $\pi^{(0)}(u)$, with the challenge message t, and it obtains the response c^*;
  - else it submits an evaluation query to $\mathcal{C}$ for the the PRF instance corresponding to receiver $\pi^{(0)}(u)$, with challenge message t and obtains the responses c^*;
  - $\mathcal{B}$ computes $\mathsf{ct}_{u,t} = \mathsf{NIAR}'.\mathbf{Enc}(\mathsf{ek}'_u, c^* \oplus x^{(0)}_{u,t}, t)$.

Clearly, if the selective-opening PRF challenger $\mathcal{C}$ always encrypts the real challenge message, then $\mathcal{A}$'s view is the same as in $\mathsf{NIARFull}^0$. Otherwise, if $\mathcal{C}$ always returns random strings for challenge queries, then $\mathcal{A}$'s view is the same as in Hyb_0. As mentioned earlier, if a receiver is conspicuously honest, then the senders paired with it in either $\pi^{(0)}$ or $\pi^{(1)}$ must be conspicuously honest too, meaning that an admissible adversary $\mathcal{A}$ will never corrupt it. Therefore, $\mathcal{B}$ is admissible w.r.t. to its own security game. Summarizing the above, $\mathcal{B}$ can translate $\mathcal{A}$'s advantage in distinguishing $\mathsf{NIARFull}^0$ and Hyb_0 into its own advantage in breaking the selective-opening PRF security.

□

Claim F.6. *Suppose that the underlying* NIAR' *scheme satisfies Definition 3.1 subject to an adversary that corrupts all receivers upfront and can adaptively corrupt senders. Then, for any non-uniform p.p.t. admissible adversary* $\mathcal{A}$*, its views in* Hyb_0 *and* Hyb_1 *are computationally indistinguishable.*

Proof. In Hyb_b where $b \in \{0, 1\}$, in each time step, to answer the adversary $\mathcal{A}$'s encryption query $\{x^{(0)}_{u,t}, x^{(1)}_{u,t}\}_{u\in\mathcal{H}_S}$ for the so-far honest set $\mathcal{H}_S$, the challenge does the following: for every $u \in \mathcal{H}_S$,

- either call $\mathsf{NIAR}'.\mathbf{Enc}(\mathsf{ek}_u, x^{(b)}_{u,t} \oplus \mathsf{PRF}.\mathbf{Eval}(k_{\pi^{(b)}(u)}, t), t)$ to encrypt the *inner-message* $x^{(b)}_{u,t} \oplus \mathsf{PRF}.\mathbf{Eval}(k_{\pi^{(b)}(u)}, t)$, where $k_{\pi^{(b)}(u)}$ denotes the PRF key of the receiver paired with sender u, which is also sender u's PRF key.
- or call $\mathsf{NIAR}'.\mathbf{Enc}(\mathsf{ek}_u, r, t)$ to encrypt some random inner-message r.

If $\mathcal{A}$ respects its admissibilty rules, then for each receiver u, then the following must hold:

1. For every eventually corrupt receiver u, let s_0 and s_1 be its corresponding senders in $\pi^{(0)}$ and $\pi^{(1)}$, respectively; then it must be that for every time step t, the inner-messages $x^{(0)}_{s_0,t} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t)$, and $x^{(1)}_{s_1,t} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t)$ (to be passed to $\mathsf{NIAR}'.\mathbf{Enc}$ in Hyb_0 and Hyb_1 respectively) are the same;
2. For every eventually honest receiver u,
   - For any t before u becomes conspicuously honest, the inner-messages $x^{(0)}_{s_0,t} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t)$ and $x^{(1)}_{s_1,t} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t)$ are the same;
   - For any t during or after the round in which u becomes conspicuously honest, the inner-messages to be passed to $\mathsf{NIAR}'.\mathbf{Enc}$ in Hyb_0 and Hyb_1 are both random inner-messages.

Now, imagine that the challenger has some randomness tape $\Gamma[\cdot,\cdot]$ which can be viewed as a two dimensional array. The randomness tape is sampled at the beginning of the experiment Hyb_b where $b \in \{0,1\}$. Later, whenever the challenger needs to sample a random inner-message for some sender whose destination is u in $\pi^{(b)}$, it will read the random coins $\Gamma[u,t]$ off the randomness tape where t denotes the current time.

It suffices to prove that *conditioned on any fixed choice of the randomness tape* Γ, $\mathcal{A}$'s views in Hyb_0 and Hyb_1 are computationally indistinguishable. This can be accomplished through a straightforward reduction to the security of the underlying NIAR' scheme. Basically, consider a reduction $\mathcal{B}$ that interacts with its own challenger as well as $\mathcal{A}$.

- $\mathcal{B}$ passes the terms $(n, \mathsf{len}, \pi^{(0)}, \pi^{(1)})$ sent by $\mathcal{A}$ directly to its own challenger, and gets back tk. It passes tk to $\mathcal{A}$.
- Further, $\mathcal{B}$ chooses the PRF key k_u for every receiver u.
- $\mathcal{B}$ corrupts all receivers upfront with its own challenger, and receives $\{\mathsf{rk}'_u\}_{u\in[n]}$.
- Whenever $\mathcal{A}$ submits an encryption query during some time step t, $\mathcal{B}$ computes the two corresponding inner-messages and pass them to its own challenger. It receives some ciphertexts from its own challenger and forwards them to $\mathcal{A}$.
- Whenever $\mathcal{A}$ makes a corruption query for some receiver u, $\mathcal{B}$ sends to $\mathcal{A}$ its key (k_u, rk'_u).
- Whenever $\mathcal{A}$ makes a corruption query for some sender, $\mathcal{B}$ forwards the query to its own challenger and passes the answer back to $\mathcal{A}$. Moreover, it reveals the corresponding player's PRF key to $\mathcal{A}$. If $\mathcal{A}$ is admissible, then a corrupt sender must have the same receiver in both $\pi^{(0)}$ and $\pi^{(1)}$, so $\mathcal{B}$ can always identify a unique PRF key to return to $\mathcal{A}$.
- $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Now, if $\mathcal{B}$'s challenger is in world $b = 0$, then $\mathcal{A}$'s view is identically distributed as Hyb_0. Else if $\mathcal{B}$'s challenger is in world $b = 1$, then $\mathcal{A}$'s view is identically distributed as Hyb_1. Further, for a fixed randomness tape, $\mathcal{B}$ is admissible as long as $\mathcal{A}$ is admissible. □

Completing the proof of Lemma F.4. Combining Claims I.8 and I.9, we have that $\mathsf{NIARFull}^0 \approx_c \mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1 \approx_c \mathsf{NIARFull}^1$ where $\approx_c$ denotes computational indistinguishability. This completes the proof of Lemma F.4. □

F.5 Completing the Proof of Theorem F.1

Theorem F.1 follows directly by combining Lemmas F.2 to F.4.

G Impossibility of Simulation Security Under Adaptive Corruption

So far in our paper, we have used indistinguishability-based security definitions. Shi and Wu [SW21] showed that under static corruption, indistinguishability-based security is equivalent to simulation-based security. In this section, we introduce a natural simulation-based security notion for adaptive corruption, and somewhat surprisingly at first sight, we show that the same equivalence of indistinguishability- and simulation-based security no longer holds under adaptive corruption. We show this by first proving an impossibility result for simulation security under adaptive corruption.

G.1 NIAR Full Simulation Security

We first define a real-world experiment in which the adversary can 1) adaptively corrupt senders or receivers; and 2) ask the challenger to encrypt on behalf of the currently honest senders in any time step. We use the notation $\mathsf{Cor}(\cdot)$ to denote a corruption oracle. Whenever Cor receives a specified sender or receiver to corrupt, it returns the newly corrupted player's secret key, as well as all the randomness the player has used in all past time steps[7]; it then updates the honest sender and receiver sets $\mathcal{H}_S$ and $\mathcal{H}_R$, respectively.

Real-world experiment $\mathsf{Real}^{\mathcal{A}}(1^\lambda)$.

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

Next, we define an ideal world experiment which involves a **stateful** simulator Sim. The simulator's job is to simulate the setup as well as responses to the adversary's encryption and corruption queries, based on only the permitted leakage which includes the following information: 1) all corrupt senders' destinations; 2) for every corrupt sender u, the challenge message submitted by the adversary on behalf of u in every round before u became corrupt; 3) for every corrupt receiver, the message it receives during every round in which its corresponding sender remains honest.

Ideal-world experiment $\mathsf{Ideal}^{\mathsf{Sim},\mathcal{A}}(1^\lambda)$.

1. $(n, \mathsf{len}, \pi) \leftarrow \mathcal{A}(1^\lambda)$.
2. $\mathcal{H}_S = [n]$, $\mathcal{H}_R = [n]$.
3. $\mathsf{tk} \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{len}, n)$, and $\mathcal{A}$ receives tk.
4. For $t = 1, 2, \ldots$:
   - Repeat the following for zero to multiple times until $\mathcal{A}$ eventually outputs $\{x_{u,t}\}_{u\in\mathcal{H}_S}$.
     - $\mathcal{A}$ sends to Sim a player to corrupt, in the form of either $(\texttt{sender}, u)$ or $(\texttt{receiver}, u)$;
     - if $\mathcal{A}$ output $(\texttt{sender}, u)$ in the previous step, Sim additionally receives the newly sender's destination as well as $\{x_{u,t'}\}_{t'<t}$;
     - else if $\mathcal{A}$ output $(\texttt{receiver}, u)$, Sim additionally receives $\{x_{\pi^{-1}(u),t'}\}_{t'<t''}$ where t'' is the first time step in which sender $\pi^{-1}(u)$'s became corrupt or $t'' = t$ if sender $\pi^{-1}(u)$ remains honest;
     - Sim returns to $\mathcal{A}$ the newly corrupted player's simulated secret key, as well as the simulated random coins consumed by the player in all previous rounds;
     - update $\mathcal{H}_S$ and $\mathcal{H}_R$ accordingly;

[7] In our scheme, the receiver does not consume randomness during decryption. In this case, $\mathsf{Cor}(\cdot)$ only needs to return the random coins used by a newly corrupted sender in all previous time steps.

- Sim receives $\{(\pi(u), x_{u,t})\}_{u\in\pi(\mathcal{H}_S)\cap\mathcal{K}_R}$, where $\mathcal{K}_R = [n]\backslash\mathcal{H}_R$, i.e., the plaintexts received by every currently corrupt receiver that is paired with an honest sender. Sim now outputs $\{\mathsf{CT}_{u,t}\}_{u\in\mathcal{H}_S}$.

Definition G.1 (NIAR full simulation security)**.** We say that a NIAR scheme satisfies full simulation security, iff for any non-uniform p.p.t. adversary $\mathcal{A}$, there exists a p.p.t. simulator Sim and a negligible function $\mathsf{negl}(\cdot)$, such that $\mathcal{A}$ cannot distinguish $\mathsf{Real}^{\mathcal{A}}(1^\lambda)$ and $\mathsf{Ideal}^{\mathsf{Sim},\mathcal{A}}(1^\lambda)$ except with $\mathsf{negl}(\lambda)$ probability.

G.2 Impossibility of Simulation Security Under Adaptive Corruption

Theorem G.2. *There does not exist any NIAR scheme that supports an unbounded (i.e., a-priori unknown) number of time steps while satisfying full simulation security.*

Proof. The proof is very similar to Nielsen's result [Nie02]. For the same of reaching a contradiction, suppose there is a scheme denoted NIAR that supports an unbounded number of time steps while satisfying full simulation security. Since decryption must enjoy perfect correctness, without loss of generality, we may assume that the receiver's decryption algorithm is deterministic in NIAR. We show that there exists an encoding scheme that leverages NIAR to compress a long random string. Let ℓ be a sufficiently large natural number, and let $\rho \xleftarrow{\$} \{0,1\}^\ell$ be a long random string.

To encode the string ρ with a common reference string r independent of ρ, we perform the following:

- First, initialize Sim with the random coins r.
- Next , call Sim with 1^λ for some sufficiently large λ, $\mathsf{len} = 1$, and $n = 1$, which outputs tk.
- Next, for $t = 1, 2, \ldots, \ell$, invoke Sim with no permitted leakage and receive a simulated ciphertext CT_t.
- Finally, corrupt the only receiver and invoke Sim, providing it with the leaked messages ρ. Sim outputs a receiver key rk.
- The resulting encoding is defined as $(\mathsf{tk}, \mathsf{rk})$.

To decode a codeword of the form $(\mathsf{tk}, \mathsf{rk})$ with the same common reference string r independent of the message, perform the following:

- Initialize Sim with the same random string r.
- Call Sim with 1^λ, $\mathsf{len} = 1$, and $n = 1$.
- Now, for $t = 1, \ldots, \ell$, we perform the following: 1) call Sim to produce CT_t; and 2) call the $\mathbf{Rte}$ algorithm which uses tk and transforms CT_t to CT'_t.
- Finally, we use rk to decrypt every CT'_t, and output the decrypted bits.

Due to Definition G.1 and the fact that the receiver's decryption algorithm is polynomially bounded, it must be that the above decoding algorithm is correct except with negligible probability. The theorem follows by making ℓ a constant factor larger than the total length of the codeword $(\mathsf{tk}, \mathsf{rk})$, by Shannon's source coding theorem [Sha48] formalized in Proposition B.6. □

H Upgrade to Adaptive Corruption for Non-Interactive Differentially Anonymous Router

In this section, we consider NIDAR schemes with receiver-insider protection. They were introduced by the recent work of Bünz, Hu, Matsuo and Shi [BHMS21]. A NIDAR scheme has exactly the same syntax as the NIAR scheme, except that it has a more relaxed security notion called (ϵ, δ)-computational differential anonymity (CDA). We show that the same static-to-adaptive-corruption compiler described in Appendix F.1 can be applied to NIDAR schemes.

First, we review the security definitions under static and adaptive corruptions, respectively — the definitions are the same as Bünz, Hu, Matsuo and Shi [BHMS21], focusing on the *receiver-insider-protection* setting.

NIDAR security under static corruption. The NIDAR security experiment, denoted $\mathsf{NIDARStatic}^{b,\mathcal{A}}(1^\lambda)$, is the same as $\mathsf{NIARStatic}^{b,\mathcal{A}}(1^\lambda)$, except that we now change the admissibility rule on the adversary to additionally require that the *adversary must submit two permutations that are separated by a single inversion.*

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

$$\Pr[\mathsf{NIDARStatic}^{0,\mathcal{A}}(1^\lambda) = 1] \leq e^{\epsilon(\lambda)} \times \Pr[\mathsf{NIDARStatic}^{1,\mathcal{A}}(1^\lambda) = 1] + \delta(\lambda).$$

NIDAR full security. The $\mathsf{NIDARFull}^{b,\mathcal{A}}(1^\lambda)$ experiment is the same as the $\mathsf{NIARFull}^{b,\mathcal{A}}(1^\lambda)$ game as defined in Section 3.2, except that we now modify the admissibility rule to additionally require that *the adversary must submit two permutations that are separated by a single inversion.*

Definition H.2 (Full computational differential anonymity)**.** Let $\epsilon > 0$ and $\delta \in (0, 1)$ be functions of the security parameter λ. We say that a NIDAR scheme satisfies full (ϵ, δ)-computational differential anonymity, iff for any non-uniform p.p.t. admissible $\mathcal{A}$, for every $\lambda \in \mathbb{N}$, it holds that

$$\Pr[\mathsf{NIDARFull}^{0,\mathcal{A}}(1^\lambda) = 1] \leq e^{\epsilon(\lambda)} \times \Pr[\mathsf{NIDARFull}^{1,\mathcal{A}}(1^\lambda) = 1] + \delta(\lambda).$$

Static-to-adaptive corruption upgrade for NIDAR. The same compiler in in Appendix F.1 works for NIDAR as well. The proof is essentially the same as in Appendix F, except that now we replace NIAR security under static/adaptive corruption with (ϵ, δ)-CDA security under static/adaptive corruption, respectively.

I NIAR Sender-insider Protection: From Static Corruption to Adaptive Corruption

Bunn, Kushilevitz, and Ostrovsky [BKO22] defined the dual version of our security notion for NIAR, that is, they consider sender-insider-protection rather than receiver-insider-protection. In the sender-insider-protection setting, we assume that the adversary knows the corrupt receivers' respective senders; however, it does not know which honest receivers corrupt senders are sending to. In this section, we show that the same static-to-adaptive corruption compiler in Appendix F.1 works for sender-insider protection NIAR schemes as well.

I.1 Definition: NIAR Security under Sender Insider Protection

We first review the security definitions for the sender-insider-protection setting. We will start with the adaptive corruption version, since the static corruption can be obtained by additionally requiring that the adversary declare all corrupt players upfront.

In comparison with the receiver-insider definitions, the security game remains the same, and the main change is that the admissibility rules are now the dual of the previous rules as highlighted below in blue.

Admissibility rules for sender-insider protection. We say that $\mathcal{A}$ is *admissible* iff with probability 1, the following holds where $\mathcal{H}_S$ and $\mathcal{H}_R$ refer to the *eventually* honest sender and receiver set, respectively, and define $\mathcal{K}_R = [n] \setminus \mathcal{H}_R, \mathcal{K}_S = [n] \setminus \mathcal{H}_S$ to be the *eventually* corrupt sender and receiver sets, respectively:

1. For all eventually corrupt receivers $v \in \mathcal{K}_R$, ${\pi^{(0)}}^{-1}(v) = {\pi^{(1)}}^{-1}(v)$.
2. For any eventually corrupt sender $u \in \mathcal{K}_S$, for any t in which u was not corrupt yet, $x_{u,t}^{(0)} = x_{u,t}^{(1)}$. In other words, here we require that in the two alternate worlds $b = 0$ or $b = 1$, every eventually corrupt sender should be sending the same message in all rounds before it was corrupted.
3. For all rounds t, and for any $v \in \mathcal{K}_R \cap \pi^{(0)}(\mathcal{H}_S) = \mathcal{K}_R \cap \pi^{(1)}(\mathcal{H}_S)$, $x_{u,t}^{(0)} = x_{u,t}^{(1)}$ where $u = (\pi^{(0)})^{-1}(v) = (\pi^{(1)})^{-1}(v)$. In other words, here we require that in the two alternate worlds $b = 0$ or 1, every eventually corrupt receiver receiving from an eventually honest sender must receive the same message in all rounds.

Definition I.1 (NIAR full security with sender-insider protection)**.** We say that a NIAR scheme is $\mathsf{fully\ secure}$ with sender-insider protection iff for any non-uniform p.p.t. *admissible* $\mathcal{A}$, its views in the two experiments $\mathsf{NIARFull}^{0,\mathcal{A}}(1^\lambda)$ and $\mathsf{NIARFull}^{1,\mathcal{A}}(1^\lambda)$ are computationally indistinguishable.

Definition I.2 (Security with sender-insider protection against static corruptions)**.** We say that a NIAR scheme satisfies security with sender-insider protection against static corruption, iff Definition I.1 is satisfied for any admissible p.p.t. adversary that always declares the corrupted players upfront.

I.2 Upgrade from Static to Adaptive Corruption for Sender-Insider Protection

We now show that the same compiler in Appendix F.1 works for the sender-insider protection setting as well. More formally, we shall prove the following theorem.

Theorem I.3 (Static to adaptive corruption compiler)**.** *Suppose that* NIAR' *satisfies Definition I.2 subject to a single-inversion, and all-sender-corrupting adversary, and suppose that* PRF *is a selective-opening secure pseudorandom function (i.e. Definition B.7). Then, the compiled* NIAR *construction shown in Figure 16 satisfies full security with sender-insider protection (i.e., Definition I.1).*

We first give a proof roadmap below, and then present the detailed proofs in Appendices I.2.1 to I.2.3.

Proof roadmap. In the remainder of this section, we prove Theorem I.3. The proof goes through multiple intermediate steps.

1. We start from a NIAR secure with sender-insider protection w.r.t. inversion under a static-corruption, all-sender-corrupting adversary. We prove that the same scheme is also secure with sender-insider protection w.r.t. inversion under an *adaptive*-corruption, all-sender-corrupting adversary (see Appendix I.2.1 and Lemma I.4).

2. Next, given a NIAR scheme secure with sender-insider protection w.r.t. inversion under an adaptive-corruption, all-sender-corrupting adversary, we remove the "inversion" restriction, and prove that it is also secure with sender-insider protection for an arbitrary pair of permutations under an adaptive-corruption, all-sender-corrupting adversary (see Appendix I.2.2 and Lemma I.5).

3. Finally, given a NIAR scheme that is secure with sender-insider protection under an adaptive-corruption, all-sender-corrupting adversary (for an arbitrary pair of permutations), we show that the static-to-adaptive compiler in Figure 16 gives a NIAR scheme secure under Definition I.1 (see Appendix I.2.3 and Lemma I.6).

I.2.1 Adaptive Corruption Under Single-Inversion and All-Sender-Corruption

Lemma I.4. *If a NIAR scheme satisfies security with sender-insider protection under a static-corruption, single-inversion, and all-sender-corrupting adversary, then it also satisfies security with sender-insider protection under a single-inversion adversary who corrupts all senders upfront, and corrupts receivers adaptively.*

Proof. Consider a non-uniform p.p.t. admissible adversary $\mathcal{A}$ who corrupts all senders upfront and corrupts the receivers adaptively. $\mathcal{A}$ is trying to distinguish its views in experiments $\mathsf{NIARFull}^{0,\mathcal{A}}(1^\lambda)$ and $\mathsf{NIARFull}^{1,\mathcal{A}}(1^\lambda)$ subject to single inversion. Then, we construct a p.p.t. admissible adversary $\mathcal{B}$ that plays a game with its own challenger $\mathcal{C}$ and leverages $\mathcal{A}$ to distinguish between $\mathsf{NIARStatic}^{0,\mathcal{B}}(1^\lambda)$ and $\mathsf{NIARStatic}^{1,\mathcal{B}}(1^\lambda)$ subject to single inversion and all-sender-corruption. The description of $\mathcal{B}$ is as follows.

1. $\mathcal{B}$ gets $(n, \mathsf{len}, \pi^{(0)}, \pi^{(1)})$ from $\mathcal{A}(1^\lambda)$. Let s_1, s_2 be the two senders involved in the inversion. Let the corresponding receivers involved in the inversion be r_1, r_2. For its game with $\mathcal{C}$, it chooses to corrupt the senders $\mathcal{K}'_S = [n]$, the receivers $\mathcal{K}'_R = [n] \setminus \{r_1, r_2\}$, and it sends $(n, \mathsf{len}, \mathcal{K}'_S, \mathcal{K}'_R, \pi^{(0)}, \pi^{(1)})$ to $\mathcal{C}$. Then, $\mathcal{C}$ computes $(\{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in[n]}, \mathsf{tk}) \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{len}, n, \pi^{(b)})$ and gives the terms $(\mathsf{tk}, \{\mathsf{ek}_u\}_{u\in[n]}, \{\mathsf{rk}_u\}_{u\in\mathcal{K}'_R})$ to $\mathcal{B}$. $\mathcal{B}$ now passes $(\mathsf{tk}, \{\mathsf{ek}_u\}_{u\in[n]})$ to $\mathcal{A}$ since $\mathcal{A}$ always corrupts all senders upfront.
2. In each time step t:
   - $\mathcal{A}$ makes no encryption queries as $\mathcal{H}_S = \emptyset$.
   - Upon receiving a corruption query to corrupt some receiver $u \in [n]$, by the admissibility rule on $\mathcal{A}$, u cannot be r_1 or r_2. Therefore, $\mathcal{B}$ returns rk_u to $\mathcal{A}$.
3. $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Observe that if $\mathcal{A}$ is admissible and respects single inversion, then, $\mathcal{B}$ is also admissible and respects single inversion. By construction, $\mathcal{B}$ respects all-sender-corruption. Further, for $b \in \{0, 1\}$, when $\mathcal{C}$ uses the challenge bit b, $\mathcal{A}$'s view is identical to $\mathsf{NIARFull}^{b,\mathcal{A}}(1^\lambda)$. Therefore, if $\mathcal{A}$ has non-negligible advantage in distinguishing $\mathsf{NIARFull}^{0,\mathcal{A}}(1^\lambda)$ and $\mathsf{NIARFull}^{1,\mathcal{A}}(1^\lambda)$, then $\mathcal{B}$ has non-negligible advantage in distinguishing $\mathsf{NIARStatic}^{0,\mathcal{B}}$ and $\mathsf{NIARStatic}^{1,\mathcal{B}}$. □

I.2.2 Removing the Single Inversion Restriction

We next remove the single inversion restriction. The proof is very similar to that of Lemma D.6 where we removed the single inversion restriction but for the static corruption in receiver-insider protection scenario.

Lemma I.5 (Removing the single inversion restriction (adaptive corruption))**.** *Suppose we are given a NIAR scheme that satisfies security with sender-insider protection subject to a single-inversion adversary who always corrupts all senders upfront, and corrupts receivers adaptively. Then, the same scheme is also secure with sender-insider protection for an arbitrary pair of permutations, subject to an adversary who always corrupts all senders upfront, and corrupts receivers adaptively.*

Proof. Given any two permutations $\pi^{(0)}$ and $\pi^{(1)}$ submitted by $\mathcal{A}$, let $\overline{C}(\pi^{(0)}, \pi^{(1)})$ be the set of receivers that have different senders in $\pi^{(0)}$ and $\pi^{(1)}$ — by the admissibility rule on $\mathcal{A}$, it must be that $\overline{C}(\pi^{(0)}, \pi^{(1)})$ are all honest receivers.

We prove for the simpler case receivers are corrupted statically instead of adaptively. The proof for corrupting receivers adaptively will be straightforward as to answer the adversary $\mathcal{A}$'s adaptive receiver corruption queries, the reduction $\mathcal{B}$ can simply forward those to its own challenger and forward back the answers. By $\mathcal{A}$'s admissibility rule, $\mathcal{A}$ can never corrupt a receiver that is in $\overline{C}(\pi^{(0)}, \pi^{(1)})$. Therefore, in the adaptive corruption case, it will still hold that if $\mathcal{A}$ is admissible, then $\mathcal{B}$ is admissible too.

We define a sequence of permutations denoted $\pi^*_0, \ldots, \pi^*_n$ where $\pi^*_0 = \pi^{(0)}$, and for any $0 < i \le n$, π^*_i is almost the same as π^*_{i-1}, except that if $i \le |\overline{C}(\pi^{(0)}, \pi^{(1)})|$, then we additionally swap the senders of the i-th honest receiver in $\overline{C}(\pi^{(0)}, \pi^{(1)})$ denoted v^*_i and whoever is receiving from ${\pi^{(1)}}^{-1}(v^*_i)$ in π^*_{i-1} — by construction, the receiver v^*_i is swapping senders with another receiver that must lie within the the set $\overline{C}(\pi^{(0)}, \pi^{(1)})$. Else if $i > |\overline{C}(\pi^{(0)}, \pi^{(1)})|$, then, $\pi^*_i = \pi^*_{i-1}$. By construction, in π^*_i, the first i honest receivers in $\overline{C}(\pi^{(0)}, \pi^{(1)})$ have their correct senders as in $\pi^{(1)}$, and thus $\pi^*_n = \pi^{(1)}$.

We now consider a sequence of hybrid experiments denoted Hyb_i where $i \in \{0, 1, \ldots, n\}$, in which a challenger interacts with an adversary $\mathcal{A}$ that has the same interface as a $\mathsf{NIARStatic}^{b,\mathcal{A}}$ adversary, and moreover, it always corrupts all senders upfront. Namely, $\mathcal{A}$ submits $n, \mathsf{len}, \mathcal{K}_S, \mathcal{K}_R, \pi^{(0)}, \pi^{(1)}$ upfront where $\mathcal{K}_S$ is guaranteed to be $[n]$, and then in every time step t, it submits $\{x^{(0)}_{u,t}, x^{(1)}_{u,t}\}_{u \in \mathcal{H}_S}$. In Hyb_i, the challenger computes the responses to $\mathcal{A}$ as follows:

- It calls the **Setup** algorithm on the input len, n and the permutation π^*_i (which is uniquely determined given $\pi^{(0)}$ and $\pi^{(1)}$ submitted by $\mathcal{A}$),
- During each time step t, there are no encryption queries to handle as $\mathcal{H}_S = \emptyset$.

By construction, and due to the admissibility rule on $\mathcal{A}$, Hyb_0 is the same as $\mathsf{NIARStatic}^{0,\mathcal{A}}$, and Hyb_n is the same as $\mathsf{NIARStatic}^{1,\mathcal{A}}$. It suffices to argue that the adversary's views in every pair of adjacent hybrids Hyb_i and Hyb_{i+1} are computationally indistinguishable. This can be achieved through a reduction to the static single-inversion security under all-corrupt-senders.

If $\pi^*_{i+1} = \pi^*_i$, then by definition, Hyb_i and Hyb_{i+1} are identically. Henceforth we focus on the case when π^*_{i+1} and π^*_i differ by exactly one inversion. Consider a reduction $\mathcal{B}$ which receives $n, \mathsf{len}, \mathcal{K}_S, \mathcal{K}_R, \pi^{(0)}, \pi^{(1)}$ from $\mathcal{A}$ upfront where $\mathcal{K}_S$ is guaranteed to be $[n]$, $\mathcal{B}$ submits to its own challenger $n, \mathsf{len}, \mathcal{K}_S, \mathcal{K}_R, \pi^*_i, \pi^*_{i+1}$ and passes the responses to $\mathcal{A}$. In every time step t, $\mathcal{A}$ submits no encryption queries as $\mathcal{H}_S = \emptyset$.

If $\mathcal{B}$'s challenger is in world $b = 0$, then $\mathcal{A}$'s view is identically distributed as in Hyb_i; else $\mathcal{B}$'s challenger is in world $b = 1$, then $\mathcal{A}$'s view is identically distributed as in Hyb_{i+1}.

Finally, by construction, if $\mathcal{A}$ respects its admissibility rules, then $\mathcal{B}$ respects its admissibility rules as well. Moreover, as mentioned earlier, $\mathcal{B}$ respects the single-inversion constraint. Therefore, $\mathcal{B}$ can translate $\mathcal{A}$'s advantage in distinguishing Hyb_i and Hyb_{i+1} into its own advantage at breaking the single-inversion static-corruption security of NIAR (subject to all-corrupting senders). □

I.2.3 Removing the All Sender Corruption Restriction

Finally, given a NIAR scheme that is secure with sender-insider protection under an adversary who corrupts all senders upfront but can adaptively corrupt receivers, we can remove the all-sender-corruption restriction with the compiler described in F.1.

Lemma I.6 (Removing the all sender corruption restriction)**.** *Suppose that* NIAR' *satisfies security with sender-insider protection under an adversary who corrupts all senders upfront but can adaptively corrupt receivers. Then, the compiled* NIAR *scheme described in F.1 satisfies full NIAR security with sender-insider protection (Definition I.1).*

Proof. We first define what is a conspicuously honest sender.

Conspicuously honest sender. Consider the experiment $\mathsf{NIARFull}^{b,\mathcal{A}}$. Recall that $\mathcal{A}$ submits two permutations $\pi^{(0)}$ and $\pi^{(1)}$ upfront. Consider some sender j, and let r_0 and r_1 be the corresponding receivers of j in $\pi^{(0)}$ and $\pi^{(1)}$, respectively, that is, for $b \in \{0,1\}$, $\pi^{(b)}(j) = r_b$. Suppose during some time step t, the adversary $\mathcal{A}$ submits an encryption query for the plaintext vectors $\{x^{(0)}_{u,t}, x^{(1)}_{u,t}\}_{u \in \mathcal{H}_S}$ where $\mathcal{H}_S$ denotes the so-far honest senders, and it turns out that $x^{(0)}_{j,t} \neq x^{(1)}_{j,t}$, i.e., j is sending two different messages in the two worlds. Then, at this moment, we know that the sender j can never be corrupt by $\mathcal{A}$ due to the admissibility rules on $\mathcal{A}$. Henceforth, such a sender is called a conspicuously honest sender. Observe also the corresponding receivers r_0 and r_1 (which could be the same) must be conspicuously honest too, by the admissibility rule on $\mathcal{A}$.

Consider the following hybrid experiments indexed by a bit $b \in \{0,1\}$.

Experiment Hyb'_b**.** Hyb'_b is almost the same as $\mathsf{NIARFull}^b$, except with the following modifications: for all $u \in [n]$, the u^{th} PRF key is associated with the u^{th} sender instead of u^{th} receiver. Consequently,

- For all $u \in [n]$, $\mathsf{ek}_u := (k_u, \mathsf{ek}'_u)$ and $\mathsf{rk}_u := (k_{\pi^{(b)^{-1}}(u)}, \mathsf{rk}'_u)$.
- Whenever $\mathcal{A}$ submits an encryption query for the plaintext vectors $\{x^{(0)}_{u,t}, x^{(1)}_{u,t}\}_{u \in \mathcal{H}_S}$ where $\mathcal{H}_S$ denotes the so-far honest senders, the challenger responds with the following ciphertexts for each $u \in \mathcal{H}_S$: $\mathsf{ct}_{u,t} = \mathsf{NIAR}'.\mathbf{Enc}(\mathsf{ek}'_u, x^{(b)}_{u,t} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t), t)$.

Experiment Hyb_b**.** Hyb_b is almost the same as Hyb'_b, except with the following modifications: whenever $\mathcal{A}$ submits an encryption query for the plaintext vectors $\{x^{(0)}_{u,t}, x^{(1)}_{u,t}\}_{u \in \mathcal{H}_S}$ where $\mathcal{H}_S$ denotes the so-far honest senders, the challenger responds with the following ciphertexts for each $u \in \mathcal{H}_S$:

- if u is conspicuously honest, then compute $\mathsf{ct}_{u,t} = \mathsf{NIAR}'.\mathbf{Enc}(\mathsf{ek}_u, r, t)$ for a randomly chosen r of appropriate length;
- else compute $\mathsf{ct}_{u,t} = \mathsf{NIAR}'.\mathbf{Enc}(\mathsf{ek}'_u, x^{(b)}_{u,t} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t), t)$ honestly.

Claim I.7. *For all* $b \in \{0,1\}$, $\mathsf{NIARFull}^b$ *and* Hyb'_b *are identically distributed.*

Proof. The distribution of all the n PRF keys is identical, hence, one can conclude that the change from $\mathsf{NIARFull}^b$ to Hyb'_b merely involves change of variable names. Thus, the two distributions are identical. □

Claim I.8. *Suppose that the PRF satisfies selective opening security, and let $b \in \{0,1\}$. Then, for the compiled* NIAR *scheme, it holds that for any non-uniform p.p.t. admissible adversary $\mathcal{A}$, its views in Hyb_b and Hyb'_b are computationally indistinguishable.*

Proof. We prove it for $b = 0$, since the case when $b = 1$ is symmetric. Given an efficient NIARFull adversary $\mathcal{A}$, we will construct an efficient reduction $\mathcal{B}$ that breaks the selective-opening security of the PRF.

- At the beginning $\mathcal{B}$ creates n PRF instances with its own challenger $\mathcal{C}$, one corresponding to each sender. Except for the PRF keys which $\mathcal{B}$ does not know at the beginning, $\mathcal{B}$ samples all other terms of the NIAR scheme using the honest algorithms.
- Whenever $\mathcal{A}$ makes a corruption query for a receiver $u \in [n]$, $\mathcal{B}$ corrupts the corresponding PRF key for the sender $\pi^{(0)^{-1}}(u)$ with its own challenger $\mathcal{C}$. It then responds to $\mathcal{A}$ with u's secret key in the NIAR scheme.
- Whenever $\mathcal{A}$ makes a corruption query for a sender $u \in [n]$, $\mathcal{B}$ corrupts the corresponding PRF key for the sender u with its own challenger $\mathcal{C}$. It then responds to $\mathcal{A}$ with u's secret key in the NIAR scheme. Additionally, it returns the random coins consumed by u's Enc algorithm in all previous time steps.
- Whenever $\mathcal{A}$ makes an encryption query for a time step t with the challenge plaintexts $\{x^{(0)}_{u,t}, x^{(1)}_{u,t}\}_{u\in\mathcal{H}_S}$, $\mathcal{B}$ computes the answers in the following way: for every $u \in \mathcal{H}_S$,
  - if u is conspicuously honest, then $\mathcal{B}$ submits to $\mathcal{C}$ a challenge query for the PRF instance corresponding to sender u, with the challenge message t, and it obtains the response c^*;
  - else it submits an evaluation query to $\mathcal{C}$ for the the PRF instance corresponding to sender u, with challenge message t and obtains the responses c^*;
  - $\mathcal{B}$ computes $\mathsf{ct}_{u,t} = \mathsf{NIAR}'.\mathbf{Enc}(\mathsf{ek}'_u, c^* \oplus x^{(0)}_{u,t}, t)$.

Clearly, if the selective-opening PRF challenger $\mathcal{C}$ always encrypts the real challenge message, then $\mathcal{A}$'s view is the same as in Hyb'_0. Otherwise, if $\mathcal{C}$ always returns random strings for challenge queries, then $\mathcal{A}$'s view is the same as in Hyb_0. As mentioned earlier, if a sender is conspicuously honest, then an admissible adversary $\mathcal{A}$ will never corrupt it. Therefore, $\mathcal{B}$ is admissible w.r.t. to its own security game. Summarizing the above, $\mathcal{B}$ can translate $\mathcal{A}$'s advantage in distinguishing Hyb'_0 and Hyb_0 into its own advantage in breaking the selective-opening PRF security.

□

Claim I.9. *Suppose that the underlying* NIAR′ *scheme satisfies Definition I.1 subject to an adversary that corrupts all senders upfront and can adaptively corrupt receivers. Then, for any non-uniform p.p.t. admissible adversary $\mathcal{A}$, its views in Hyb_0 and Hyb_1 are computationally indistinguishable.*

Proof. In Hyb_b where $b \in \{0,1\}$, in each time step, to answer the adversary $\mathcal{A}$'s encryption query $\{x^{(0)}_{u,t}, x^{(1)}_{u,t}\}_{u\in\mathcal{H}_S}$ for the so-far honest set $\mathcal{H}_S$, the challenger does the following: for every $u \in \mathcal{H}_S$,

- either call $\mathsf{NIAR}'.\mathbf{Enc}(\mathsf{ek}_u, x_{u,t}^{(b)} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t), t)$ to encrypt the *inner-message* $x_{u,t}^{(b)} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t)$, where k_u denotes the PRF key of the sender u, which is also receiver $\pi^{(b)}(u)$'s PRF key.
- or call $\mathsf{NIAR}'.\mathbf{Enc}(\mathsf{ek}_u, r, t)$ to encrypt some random inner-message r.

If $\mathcal{A}$ respects its admissibilty rules, then for each sender u, then the following must hold:

1. For every eventually corrupt sender u, it must be that for every time step t, the inner-messages $x_{u,t}^{(0)} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t)$, and $x_{u,t}^{(1)} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t)$ (to be passed to $\mathsf{NIAR}'.\mathbf{Enc}$ in Hyb_0 and Hyb_1 respectively) are the same;
2. For every eventually honest sender u,
   - For any t before u becomes conspicuously honest, the inner-messages $x_{u,t}^{(0)} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t)$ and $x_{u,t}^{(1)} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t)$ are the same;
   - For any t during or after the round in which u becomes conspicuously honest, the inner-messages to be passed to $\mathsf{NIAR}'.\mathbf{Enc}$ in Hyb_0 and Hyb_1 are both random inner-messages.

Now, imagine that the challenger has some randomness tape $\Gamma[\cdot, \cdot]$ which can be viewed as a two dimensional array. The randomness tape is sampled at the beginning of the experiment Hyb_b where $b \in \{0, 1\}$. Later, whenever the challenger needs to sample a random inner-message for some sender u, it will read the random coins $\Gamma[u, t]$ off the randomness tape where t denotes the current time.

It suffices to prove that *conditioned on any fixed choice of the randomness tape* Γ, $\mathcal{A}$'s views in Hyb_0 and Hyb_1 are computationally indistinguishable. This can be accomplished through a straightforward reduction to the security of the underlying NIAR' scheme. Basically, consider a reduction $\mathcal{B}$ that interacts with its own challenger as well as $\mathcal{A}$.

- $\mathcal{B}$ passes the terms $(n, \mathsf{len}, \pi^{(0)}, \pi^{(1)})$ sent by $\mathcal{A}$ directly to its own challenger, and gets back tk. It passes tk to $\mathcal{A}$.
- Further, $\mathcal{B}$ chooses the PRF key k_u for every sender u.
- $\mathcal{B}$ corrupts all senders upfront with its own challenger, and receives $\{\mathsf{ek}'_u\}_{u \in [n]}$.
- Whenever $\mathcal{A}$ submits an encryption query $\{x_{u,t}^{(0)}, x_{u,t}^{(1)}\}_{u \in \mathcal{H}_S}$ during some time step t, $\mathcal{B}$ can't query its challenger as it has corrupted all the senders. So, it perfectly simulates the ciphertexts for all $u \in \mathcal{H}_S$ without knowing the challenger's choice b as follows:
  - If u is not conspicuously honest, then, $x_{u,t}^{(0)} = x_{u,t}^{(1)}$. Therefore, the two inner-messages $x_{u,t}^{(0)} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t) = x_{u,t}^{(1)} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t)$ for $b = 0$ and $b = 1$ are the same. Therefore, $\mathcal{B}$ can perfectly simulate the ciphertext as $\mathsf{ct}_{u,t} = \mathsf{NIAR}'.\mathbf{Enc}(\mathsf{ek}'_u, x_{u,t}^{(0)} \oplus \mathsf{PRF}.\mathbf{Eval}(k_u, t), t)$. (Note that here the PRF keys correspond to senders and not receivers. Consequently, $\mathcal{B}$ can simulate this step perfectly. If the PRF keys were associated with receivers, then, $\mathcal{B}$ would not know which of the two PRF keys to use at this step.)
  - If u is conspicuously honest, then, the two inner-messages for $\mathsf{NIAR}'.\mathbf{Enc}$ are both random strings. Therefore, $\mathcal{B}$ reads the random coins $r = \Gamma[u, t]$ from its randomness tape Γ and perfectly simulates the ciphertext as $\mathsf{ct}_{u,t} = \mathsf{NIAR}'.\mathbf{Enc}(\mathsf{ek}'_u, r, t)$.

  $\mathcal{B}$ sends the ciphertexts $\{\mathsf{ct}_{u,t}\}_{u \in \mathcal{H}_S}$ computed above to $\mathcal{A}$.
- Whenever $\mathcal{A}$ makes a corruption query for some sender u, $\mathcal{B}$ sends to $\mathcal{A}$ its key (k_u, ek'_u).

- Whenever $\mathcal{A}$ makes a corruption query for some receiver u, $\mathcal{B}$ forwards the query to its own challenger and passes the answer back to $\mathcal{A}$. Moreover, it reveals the corresponding player's PRF key to $\mathcal{A}$. If $\mathcal{A}$ is admissible, then a corrupt receiver must have the same sender in both $\pi^{(0)}$ and $\pi^{(1)}$, so $\mathcal{B}$ can always identify a unique PRF key to return to $\mathcal{A}$.
- $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Now, if $\mathcal{B}$'s challenger is in world $b = 0$, then $\mathcal{A}$'s view is identically distributed as Hyb_0. Else if $\mathcal{B}$'s challenger is in world $b = 1$, then $\mathcal{A}$'s view is identically distributed as Hyb_1. Further, for a fixed randomness tape, $\mathcal{B}$ is admissible as long as $\mathcal{A}$ is admissible. □

Completing the proof of Lemma I.6. Combining Claims I.7 to I.9, we have that $\mathsf{NIARFull}^0 \equiv \mathsf{Hyb}'_0 \approx_c \mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1 \approx_c \mathsf{Hyb}'_1 \equiv \mathsf{NIARFull}^1$ where $\approx_c$ denotes computational indistinguishability and $\equiv$ denotes distributionally equivalent. This completes the proof of Lemma I.6. □

I.2.4 Completing the Proof of Theorem I.3

Theorem I.3 follows directly by combining Lemmas I.4 to I.6.