Adaptive NIKE for Unbounded Parties

Shafik Nassar
UT Austin
shafik@cs.utexas.edu

Brent Waters
UT Austin and NTT Research
bwaters@cs.utexas.edu

June 13, 2026

Abstract

This paper presents the first construction of adaptively secure non-interactive key exchange (NIKE) for an unbounded number of parties. Prior unbounded protocols were restricted to static security, with no adaptively secure construction known even in the random oracle model.

Our main contribution is a NIKE scheme in the standard model, that supports an unbounded number of honest and malicious users, as well as unbounded party sizes, while tolerating a bounded number of dynamic user corruptions. The construction is based on sub-exponential indistinguishability obfuscation and sub-exponential fully-homomorphic encryption. A key technical contribution is a new application of what we call a function-extractable hash function. This is a variant of a function binding hash function that enables resilient extraction of properties from maliciously hashed digests.

As an additional contribution, we present a compiler in the random oracle model that upgrades any adaptive (unbounded) NIKE that does not support dynamic user corruptions at all into a fully adaptive (unbounded) NIKE that supports an unbounded number of dynamic corruptions. This compiler is completely generic, does not introduce any additional assumptions, and does not rely on sub-exponential hardness.

Contents

1 Introduction **3**
- 1.1 Our Results . . . 4
- 1.2 Technical Overview . . . 5

2 Preliminaries **9**

3 Function-Extractable Hash **12**
- 3.1 Function-Extractable Hash for Conjunction of Block Functions . . . 14

4 Multiparty Non-interactive Key Exchange **14**

5 Constructing Adaptive NIKE with Static Corruptions **17**
- 5.1 Security Analysis . . . 19

6 Unbounded Corruptions in the Random Oracle Model **28**
- 6.1 Multiparty Non-interactive Key Exchange in the Random Oracle Model . . . 28
- 6.2 Achieving Full Adaptive Security . . . 29

A Adaptive Corruptions from Static Corruptions **37**

B Constructing Adaptive NIKE Without a Trusted Setup **40**
- B.1 Additional Preliminaries: Signature Schemes . . . 41
- B.2 Trustless NIKE Construction . . . 43
- B.3 Security Analysis . . . 45

C Collision-Resistant Function-Extractable Hash **51**

D Constructing Function-Extractable Hash **52**

1 Introduction

Non-interactive key exchange (NIKE) is a fundamental cryptographic primitive that allows two or more parties to agree on a shared secret key without any interaction, beyond a public key that each party posts once-and-for-all on a repository, which can be reused across different invocations of the protocol. The concept was first introduced by Diffie and Hellman in their seminal work [DH76], in which they presented a protocol for the two-party case, and their protocol constitutes the backbone of secure communication today. Extending the Diffie-Hellman NIKE beyond two parties remained open for decades, until Joux [Jou00] gave a solution for three parties based on bilinear maps. With the prevalence of chat groups and conference calls in the current communication ecosystem, getting a NIKE protocol in the *multiparty* setting is of great interest. But even going beyond three seemed very difficult, and was a "major open problem" according to Boneh and Silverberg [BS02], who demonstrated how Joux's construction can be extended to the multiparty setting, assuming the existence of multilinear maps. A decade later, multilinear maps candidates were proposed [GGH13a, CLT13, GGH15] alongside their corresponding multiparty NIKE constructions. However, multilinear maps have fallen out of favor in the cryptographic community due to "zeroizing" attacks [GGH13a, CHL+15, HJ16, CLLT16] which, among other things, completely broke NIKE constructions based on multilinear maps.

Multiparty NIKE from iO. The landscape of multiparty NIKE changed dramatically with the emergence of indistinguishability obfuscation (iO) [BGI+01, GGH+13b] and the punctured programming paradigm [SW14]. While initial iO candidates were based on multilinear maps [GGH+13b], a long line of works concluded with constructions of iO from "well-founded" assumptions [JLS21, JLS22, RVV25]. Boneh and Zhandry [BZ14] gave the first multiparty NIKE construction from iO and one-way functions (OWFs), which roughly works as follows: the secret key sk for each user is a seed for some pseudorandom generator (PRG) G, and the corresponding public key is $\mathsf{pk} = G(\mathsf{sk})$. A trusted setup samples a key K of a punctured pseudorandom function (PPRF) F, and obfuscates the following program P: on input a set of public keys $U = \{\mathsf{pk}_i\}_{i\in[n]}$ and a secret key sk, the program $P(U, \mathsf{sk})$ outputs $F(K, U)$ if there exists some $\mathsf{pk} \in U$ such that $G(\mathsf{sk}) = \mathsf{pk}$, and outputs $\perp$ otherwise. It is easy to see that the Boneh-Zhandry construction is inherently limited to only bounded parties, since the program circuit size grows with its input size, which in this case constitutes the full description of the parties.

Ananth et al. [ABG+13] showed how to generalize the Boneh-Zhandry [BZ14] construction to the *unbounded* setting by hashing the public keys using a hash with succinct local openings (e.g., a Merkle hash [Mer89]). This strategy works with a Merkle hash only if we assume *differing-inputs obfuscation* [ABG+13, BCP14], a strong notion with serious implausibility results [BSW16, GGHW17]. Subsequently, Khurana, Rao and Sahai [KRS15] demonstrated that the unbounded NIKE framework [ABG+13] can work with iO by replacing the Merkle hash with a somewhere-statistically binding hash (SSB) [HW15]. Additionally, Alwen et al. [ABG+25] found another way to construct NIKE for unbounded parties using iO for Turing machines with unbounded inputs [JJ22] and succinct PPRFs.

Adaptive security. The main drawback of prior unbounded NIKE constructions [KRS15, ABG+25] from iO is that they only achieve static security[1], where the adversary commits to the set of users it wants to attack ahead of time. Khurana et al. [KRS15] concluded by saying "it would be interesting to design protocols that tolerate more active attacks by adversaries, for an unbounded number of parties". Indeed, achieving adaptive security appears to be of paramount importance in the context of multiparty NIKE, since users dynamically join the system and the target set of the adversary and whom it will deploy resources to compromise may realistically change depending on information it learns from prior key compromises.

The relationship between selective and adaptive security for multiparty NIKE is subtle. When the party bound n is constant, we could simply guess the target set of the adversary in order to upgrade static security to adaptive security. Since an efficient adversary can at most register a polynomial number of users q, we only incur an $O(q^n)$ loss, a polynomial. When the party bound n is polynomial, we incur an exponential loss, which can still be mitigated by relying on sub-exponential assumptions and using complexity leveraging. However, the situation is completely different in the unbounded setting, since q^n would be an unbounded exponential, meaning that even complexity leveraging cannot help.

[1] We point out that even by relying on differing inputs obfuscation, Ananth et al. [ABG+13] only prove static security.

State-of-the-art. Achieving adaptive security for unbounded NIKE remained elusive, even in the *random oracle model* (ROM). Namely, Hofheinz et al. [HJK+16] introduced universal samplers and showed how they can be used to construct adaptive NIKE, and they constructed universal samplers in the ROM from sub-exponentially secure iO. However, their construction only supports bounded parties. We point out a couple of additional works that get adaptive security without random oracles nor sub-exponential assumptions, but also only work in the bounded multiparty setting:

- Rao [Rao14] proposed a new primitive she called obliviously-patchable puncturable pseudorandom functions, and constructed adaptive multiparty NIKE from that primitive in conjunction with iO. However, the security of this new primitive is itself an interactive assumption that does not seem any easier to construct than adaptive multiparty NIKE.
- The work of Koppula, Waters and Zhandry [KWZ22] constructs adaptive bounded multiparty NIKE from polynomially-secure iO and standard polynomially-secure group-based assumptions. Their techniques do not seem to easily transfer to the unbounded-party setting, even if we allow sub-exponential security.

As such, there are currently no known constructions of adaptive unbounded-party NIKE in the standard model or the ROM, even if we allow sub-exponential assumptions.

1.1 Our Results

Adaptive Unbounded NIKE in the Standard Model

In this work, we construct the *first* adaptively secure unbounded-party NIKE in the standard model, without a random oracle nor a trusted setup. Our construction relies on sub-exponential iO and sub-exponential fully-homomorphic encryption (FHE), and supports unbounded party size, unbounded honest users and unbounded malicious users. In addition to adaptively choosing the target set, our model allows the adversary to *dynamically corrupt* honest users of its choosing. Our construction supports a bounded number of dynamic corruptions, in the sense that our parameters grow polynomially with the number of dynamic corruptions. Formally, our main theorem is stated below:

Theorem 1.1. *Assume there exists sub-exponentially secure indistinguishability obfuscation and sub-exponentially secure fully-homomorphic encryption. Then there exists an adaptively secure NIKE where the number of honest users, the number of malicious users and the party size are all unbounded. The only bound is on the number of* dynamically corrupted *users.*

Our main construction, which we present in Section 5, in fact utilizes a trusted setup, but Koppula et al. [KWZ22] show how to generically compile any NIKE with a trusted setup into a trustless NIKE by additionally assuming iO, which we assume in our construction anyway. The compiler of Koppula et al. [KWZ22] is quite involved, so in addition, we extend our construction in Appendix B in order to *directly* get a trustless, unbounded and adaptively secure NIKE without going through that compiler [KWZ22]. Along the way of getting our main construction, we extend the notion of function-binding hash which was introduced by Freitag, Waters and Wu [FWW23] to function-*extractable* hash, which gives an interesting new technique of using such hash functions with iO.

Unbounded Dynamic Corruptions in the ROM

As an additional contribution, we present a compiler in the ROM that upgrades any adaptive NIKE (for unbounded parties) to support *unbounded* dynamic corruptions:

Theorem 1.2. *Assume there exists an adaptively secure NIKE for unbounded parties, that does not support dynamic corruptions. Then there exists an adaptively secure NIKE for unbounded parties in the random oracle model that supports unbounded dynamic corruptions.*

We emphasize that Theorem 1.2 is generic, and does not assume any additional cryptographic primitives, does not use complexity leveraging, and does not require super-polynomial security. Therefore, it could be of independent

interest and can be potentially useful for NIKE constructions that do not rely on iO. As a corollary, we get the first adaptive NIKE in the ROM with unbounded party size and unbounded dynamic corruptions.[2]

Essentially, our work offers the following tradeoff for adaptively secure NIKE for unbounded parties: either support bounded dynamic corruptions in the standard model, or unbounded dynamic corruptions in the ROM. Supporting unbounded dynamic corruptions in the standard model without guessing the challenge set remains an interesting open question, even in the case of *bounded* multiparty NIKE [KWZ22].

1.2 Technical Overview

In this section, we give an overview of our main construction. We will construct a *succinct* multiparty NIKE, meaning that the length of the public parameters is essentially independent of the number of parties that the scheme supports, which suffices to get unbounded-party NIKE. Our main construction will initially have two caveats:

1. The construction utilizes a trusted setup. However, the trusted setup can be removed in multiple ways, as we discuss at the end of this section.
2. The construction will not support dynamic corruptions: this means that the challenger-generated keys cannot be corrupted by the adversary[3], but the adversary can still adaptively choose the challenge set. In Appendix A, we prove that we can generically upgrade any such NIKE to support a bounded number of dynamic corruptions.

Prior Work: Blueprint For Static Security

Our construction follows the approach of Ananth et al. [ABG+13] and Khurana et al. [KRS15], but with crucial differences that we highlight later. We start with a brief overview of the iO [KRS15] construction. The ingredients they use are:

1. An indistinguishability obfuscator $i\mathcal{O}$.
2. A length-doubling pseudorandom generator $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$.
3. A puncturable pseudorandom function (PPRF) F. Recall that a PPRF is a pseudorandom function equipped with a puncturing algorithm that takes a key K and a string x^*, and outputs a punctured key $K\{x^*\}$. The punctured key behaves identically to K on all $x \neq x^*$, and moreover the security guarantees that $F(K, x^*)$ remains pseudorandom even given $K\{x^*\}$.
4. A somewhere-statistically binding (SSB) hash [HW15]. We recall that an SSB is a succinct hash with succinct local openings, which can be verified using a Verify algorithm, but the hash key can be sampled in binding mode on a specific index $i \in [n]$. Such a hash key *statistically* guarantees that any digest can only be opened to one value on index i. Additionally, a binding key is computationally indistinguishable from a (standard) key.

Let n be a bound on the party size. The NIKE setup algorithm samples a hash key hk and a PPRF key K, and obfuscates the following program:

[2]The sub-exponential security in Theorem 1.1 is only required for supporting dynamic corruptions and constructing constrained PRFs. See Theorem 5.2 for more details. Theorem 1.2 does not need the underlying NIKE to support dynamic corruptions. Constrained PRFs can be constructed from sub-exponentially secure iO and OWFs in the standard model, which is why we did not need to explicitly state it in Theorem 1.1. However, there are constructions of constrained PRFs from polynomially secure iO and LWE [DKN+20]. As such, it is possible to base the ROM NIKE construction on *polynomial* security of the underlying primitives.

[3]This setting is close to the notion of semi-static security [BZ14], where the adversary a priori declares a set of users which cannot be corrupted, and the challenge has to be a subset of that set.

Constants: A PPRF key K, and a SSB hash key hk.
Inputs: A digest dig, an index $j \in [n]$, a public key pk_j, a secret sk_j, and an opening π_j.

1. If $\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, j, \mathsf{pk}_j, \pi_j) = 0$, outputs $\perp$.
2. If $G(\mathsf{sk}_j) \neq \mathsf{pk}_j$, outputs $\perp$.
3. Outputs $F(K, \mathsf{dig})$.

Figure 1: Program $\mathsf{crsProg}[K, \mathsf{hk}]$.

Denote the obfuscated program by P. The CRS consists of P and hk. To publish a public key, a user samples a secret key $\mathsf{sk} \xleftarrow{R} \{0,1\}^\lambda$ and publishes $\mathsf{pk} = G(\mathsf{sk})$. To generate a shared key for an ordered tuple of public keys $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_t)$, the user must hold the secret key sk_j for some pk_j in the tuple. Then, that user can hash $(\mathsf{dig}, \pi_1, \ldots, \pi_t) = \mathsf{Hash}(\mathsf{hk}, U)$ (we assume that the hashing algorithm outputs the local openings in addition to the digest), and output $P(\mathsf{dig}, j, \mathsf{pk}_j, \mathsf{sk}_j, \pi_j)$ as the shared key.

In the NIKE security game without dynamic corruptions, the adversary can make the following queries:

- **Honest user registration:** the challenger increments a counter i, samples $\mathsf{sk}_i \xleftarrow{R} \{0,1\}^\lambda$, and sends $\mathsf{pk}_i = G(\mathsf{sk}_i)$ to the adversary.
- **Malicious user registration:** the adversary sends a (potentially malicious) public key pk, the challenger increments the same counter i, and sets $\mathsf{pk}_i = \mathsf{pk}$.
- **Evaluation:** the adversary sends a set of indices $\mathcal{I} \subseteq [2^\lambda]$ and an index $j^* \in \mathcal{I}$ such that $\mathcal{I}$ consists of indices that all have associated public keys, and j^* specifically was generated by the challenger (i.e., the challenger knows its corresponding secret key). The challenger would typically use the secret key sk_{j^*} to generate the shared key, but since the challenger knows the PPRF key K, it can compute the shared key directly as follows: hashes $\mathsf{dig}^* = \mathsf{Hash}(\mathsf{hk}, U)$ where $U = (\mathsf{pk}_i)_{i \in \mathcal{I}}$ and outputs $F(K, \mathsf{dig}^*)$ as the shared key.

At any point, the adversary can make a challenge query $\mathcal{I}^* \subseteq [2^\lambda]$, to which the challenger responds with either the shared key $y_0 = F(K, \mathsf{dig}^*)$, where $\mathsf{dig}^* = \mathsf{Hash}(\mathsf{hk}, U^*)$ and $U^* = (\mathsf{pk}_i)_{i \in \mathcal{I}^*}$, or a random string y_1 in the codomain of the PPRF. The adversary is called admissible if $\mathcal{I}^*$ consists of challenger-generated users only, and it was never sent as an evaluation query before. Finally, the adversary breaks the NIKE security if it can guess whether it got y_0 or y_1.

Static security. In the static security game, the adversary has to commit to the challenge set $\mathcal{I}^*$ ahead of time. This makes the proof strategy straightforward: we want to get to a hybrid where the PPRF key K is punctured on the digest dig^* of the challenge set. To do so, we first use the PRG security in order to change all public keys of users in $\mathcal{I}^*$ to truly random strings in the codomain of the PRG. This is possible because the adversary never gets to see the corresponding secret keys[4], and the challenger does not need any secret key in order to answer **evaluation** queries.

Now, we can define a sequence of hybrids, where the obfuscated program refuses to evaluate $F(K, \mathsf{dig}^*)$ for a subset of (hash) indices. Specifically, in Hyb_k, the CRS program is changed as follows:

[4]Even if we allow corruption queries, an admissible adversary cannot corrupt a user in the challenge set.

Constants: A constrained PRF key K, a SSB hash key hk, a digest dig^*, an index k.
Inputs: A digest dig, an index $j \in [n]$, a public key pk_j, a secret sk_j, and an opening π_j.

1. If $\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, j, \mathsf{pk}_j, \pi_j) = 0$, outputs $\perp$.
2. If $G(\mathsf{sk}_j) \neq \mathsf{pk}_j$, outputs $\perp$.
3. If $\mathsf{dig} = \mathsf{dig}^*$ and $j \leq k$, outputs $\perp$.
4. Outputs $F(K, \mathsf{dig})$.

Figure 2: Program $\mathsf{crsProg}[K, \mathsf{hk}, \mathsf{dig}^*, k]$.

It is easy to see that by iO security, Hyb_0 is computationally indistinguishable from the real security game and Hyb_n is computationally indistinguishable from a game where we use a PPRF key that is punctured on dig^* (since the k then covers all possible indices). Therefore, the problem now only reduces to proving that Hyb_k and Hyb_{k+1} are computationally indistinguishable for all $k \in [0, n-1]$. This is done as follows: make the SSB key hk binding on index $k+1$, which goes unnoticed by SSB mode indistinguishability. Using SSB binding security, the digest dig^* can be only opened to the one (truly random) public key pk_{k+1} that was sampled by the challenger. Since the PRG is length doubling, this public key lies outside of the image of G with overwhelming probability, therefore there does not exist any string sk such that $G(\mathsf{sk}) = \mathsf{pk}_{k+1}$, therefore the program would output $\perp$ anyway, so we can just advance k to $k+1$. We then change hk back to non-binding, which gets us to Hyb_{k+1}.

This Work: Getting Adaptive Security

Constraining over puncturing. In the adaptive security game, the adversary reveals the challenger query much later than the setup in which the PRF key is sampled. Puncturing the PRF key at a single point is inherently "static", and requires the challenger to know the digest, and hence the challenge set, in advance. To overcome this problem, we use the notion of *constrained PRFs* (CPRF) [BW13, KPTZ13, BGI14]: a constrained PRF with an input length ℓ_{in} is just like a PRF, but is additionally equipped with a constraining algorithm that inputs a key K and a predicate $C : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$, and outputs a constrained key $K^{(C)}$. Correctness states that this key can be used to evaluate $F(K, x)$ for all x satisfying $C(x) = 1$. Security requires that $F(K, x)$ remains pseudorandom for any x satisfying $C(x) = 0$. Our NIKE construction uses the same program from Figure 1, but replaces the PPRF with a CPRF. In the security proof, if we reach an experiment where the CPRF key is constrained on *all* digests that are computed from challenger-generated keys exclusively, then we would be done: any admissible challenge set consists of challenger-generated keys exclusively, therefore the CPRF security implies that the shared key, computed as the evaluation of the CPRF on the corresponding digest, is pseudorandom if the adversary is only given the constrained key. We remark that Boneh and Zhandry [BZ14] use a CPRF to prove that their bounded NIKE construction is "semi-statically" secure, but their CPRF key has to grow at least with the size of the challenge set.

Labeling keys. For constraining to make sense, we need a way to distinguish challenger-generated public keys from whatever malicious or honest public keys the adversary can produce. Here, we borrow an idea from Koppula et al. [KWZ22], and use *pseudorandom deterministic encryption* (PDE)[5]: a PDE is a secret key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ in which the encryption algorithm is deterministic. Security requires that without the secret key, ciphertexts look pseudorandom. Strong correctness requires that under any encryption key ek, every ciphertext ct decrypts to a message i if and only if i encrypts to ct. This means that "invalid" ciphertexts decrypt to some special symbol $\perp$. This primitive can be built from PRFs [KPW17]. Now the first two steps in our hybrid argument consist of switching the challenger-generated keys to truly random (using PRG security), and then to pseudorandom deterministic encryptions: the public key of the ith challenger-generated user is simply the encryption of its index i, that is $\mathsf{pk}_i = \mathsf{Enc}(\mathsf{ek}, i)$, where ek is the encryption key. We refer to such keys as *labeled keys*. Since the key ek does

[5] In [KWZ22], they actually use a stronger notion of *puncturable PDE*. By structuring our hybrids correctly, we do not need any puncturing.

not appear anywhere in the view of the adversary, the second step can be reduced to the PDE security. Given the PDE key ek, we can check whether a public key pk is labeled simply by checking whether pk is a valid ciphertext

Everything, everywhere, all at once. We now have a way to identify a challenger-generated key. However, the problem now is that the obfuscated program only gets access to a hash of the public keys: the SSB hash (respectively, somewhere-*extractable* hash) allows statistical binding (respectively extraction) only on one location, whereas our goal is to decide whether *every* key in the hashed set is labeled. Indeed, if we have a succinct hash function with an extraction trapdoor that extracts whether all keys are labeled, then we can constrain our CPRF key to not work on any digest that only includes labeled keys. The idea is that if the PRG is sufficiently expanding (say, length-tripling), then all labeled keys are outside of the image of the PRG with overwhelming probability[6], therefore constraining the CPRF key does not really change the functionality of the obfuscated program, which allows us to appeal to iO security. For this idea to work with iO, we need a *resilient function extraction* security: for any digest, even one that is *maliciously* generated, if the trapdoor decides that all keys are labeled then it is statistically impossible to open *any* index to an unlabeled key.

Recently, many hash functions with "global" guarantees have been introduced and constructed from various assumptions [FWW23, BBK+23, NWW24, NWW25]. Unfortunately, none of the previous hash functions suffice for our use case out-of-the-box. Brakerski et al. [BBK+23] and Nassar et al. [NWW24, NWW25] construct hash functions that allow extracting some global properties, however, their constructions only support bit-fixing/zero-fixing predicates which are not expressive enough for our application. The function-binding hash (FBH) of Freitag et al. [FWW23] supports conjunction/disjunction of block functions, which is exactly the function family that we need in our context, and moreover, can be constructed from fully-homomorphic encryption. While FBH [FWW23] is not equipped with an extraction algorithm, we observe that we can easily add such an algorithm to the Freitag et al. [FWW23] construction. But the FBH definition does not guarantee anything about maliciously-generated digests, therefore the crucial question is whether this construction actually supports resilient function extraction. In Appendix D, we prove that this is indeed the case. We refer to hash functions with this property as *function-extractable hash* functions (FEH), and give the definition in Section 3. We believe that FEH can find additional applications, be it in conjunction with iO or not, similar to how somewhere-*extractable* hash was a useful and meaningful extension of SSB hash.

This concludes the overview of our main NIKE construction. The full construction and security proofs can be found in Section 5.

Security against dynamic corruptions. Dynamic corruptions throw a wrench in our previous analysis, since we cannot just make all challenger-generated keys labeled: if the adversary corrupts an honest user with public key pk, the challenger should be able to produce a secret key sk such that $\mathsf{pk} = G(\mathsf{sk})$. However, our analysis relies on the fact that such a secret does not exist! The solution is for the challenger to guess the indices of the corrupted users ahead of time, and sample those keys normally, i.e., not labeled. If the guess is correct, then the challenger will be able to answer the corruption queries, and at the same time, an admissible challenge set should not include any of the corrupted users, which means that we can use the CPRF security as before to argue that the challenge shared key is pseudorandom. Recall that an efficient adversary can register less than 2^λ honest users (for sufficiently large λ), therefore if we bound the number of dynamic corruptions by $c \in \mathbb{N}$, then the guess comes with a $2^{c\lambda}$ loss, which we can afford if we assume sub-exponential hardness. In fact, this transformation is generic and can turn any sub-exponentially secure adaptive NIKE *without* dynamic corruptions into an adaptive NIKE *with* bounded dynamic corruptions. The transformation is formally presented and proved in Appendix A.

Removing the trusted setup. Removing the trusted setup can be done generically (assuming iO) using a transformation introduced by Koppula et al. [KWZ22, Section 3]. However, we can also directly get a construction without a trusted setup by additionally employing signature schemes. This step can be viewed as adapting the Boneh-Zhandry technique for [BZ14, Section 4] to our unbounded construction. We defer the overview and the full construction to Appendix B.

[6]To make this argument go through, the labeled keys have to be PDE ciphertexts XORed with a random pad, but we ignore this point in the overview.

Unbounded Dynamic Corruptions in the Random Oracle

As an independent contribution, we present a compiler that takes any adaptive NIKE in the plain model, and upgrades it to support unbounded dynamic corruptions in the random oracle model (ROM). This gives a new avenue to achieving full adaptive security in the ROM. Notably, our compiler is completely generic, and does not introduce new assumptions or require super-polynomial hardness. This means that it can potentially be useful for future NIKE constructions that do not rely on iO.

Our transformation is similar in spirit to the two-key simulation technique of Gentry and Waters [GW09], which is used to achieve adaptively secure broadcast encryption in the standard model. However, the NIKE application causes the techniques to depart: we will use 2λ keys instead of 2, and we will additionally rely on a random oracle.

The transformation is simple: given a plain model NIKE Π_{NIKE}, the key publishing algorithm of the new ROM NIKE samples 2λ key pairs which we denote by $(\mathsf{sk}_{i,b}, \mathsf{pk}_{i,b})_{i\in[\lambda],b\in\{0,1\}}$. The public key consists of all 2λ sub-public keys $\mathsf{pk} = (\mathsf{pk}_{i,b})_{i\in[\lambda],b\in\{0,1\}}$. The secret consists of half the sub-secret keys: the algorithm samples a uniform selecting string $s = s_1 \cdots s_\lambda \xleftarrow{R} \{0,1\}^\lambda$, and the secret key is $\mathsf{sk} = s, (\mathsf{sk}_{i,s_i})_{i\in[\lambda]}$. The publishing algorithm does not use the random oracle.

Given a hash function $\mathcal{H}$ modeled as a random oracle, the key generation algorithm takes a set of public keys $U = \{\mathsf{pk}^{(1)}, \cdots, \mathsf{pk}^{(t)}\}$ (where $t = |U|$) and a corresponding secret key $\mathsf{sk}^{(j)} = s^{(j)}, (\mathsf{sk}^{(j)}_{i,s_i})_{i\in[\lambda]}$ for the jth public key. The algorithm queries the random oracle on U to decide which sub-keys are used. That is, let $R = \mathcal{H}(U)$ and parse $R = R^{(1)} \| \cdots \| R^{(t)}$, where each $R^{(k)} \in \{0,1\}^\lambda$. For each $k \in [t]$, the sub-public keys determined by the selection string $R^{(k)}$ are the ones participating in the key evaluation. That is, denote the set $V^{(k)} = \left\{\mathsf{pk}^{(k)}_{1,R^{(k)}_1}, \cdots, \mathsf{pk}^{(k)}_{\lambda,R^{(k)}_\lambda}\right\}$ and let V be the union of all $V^{(k)}$. Then the key generation algorithm invokes the underlying NIKE on the set V. Of course, this only succeeds if $R^{(j)}$ and $s^{(j)}$ happen to agree on at least one index: if such an index i' exists, then the evaluating user can use $\mathsf{sk}^{(j)}_{i',s_{i'}}$ as the secret key for the underlying NIKE key generation invocation. Luckily, this happens with probability $1 - 2^{-\lambda}$, leaving us with a negligible correctness error.

The security reduction to the underlying NIKE works as follows: whenever the adversary registers an honest user, the reduction samples the selection string uniformly, and samples the corresponding sub-keys itself, whereas the others are registered as honest users in the underlying NIKE. This allows the reduction to answer all key evaluation queries even without going through the underlying NIKE challenger (with all but negligible probability). The key challenge is that we want the challenge set to consist exclusively of challenger-generated sub-keys. To do so, the reduction guesses the index of the random oracle query corresponding to the challenge set, and programs that query to be answered with the negation of the corresponding selection strings. On one hand, if the guess is correct then this translates the high level challenge query to a valid low-level challenge query (since the non-selection sub-keys are generated by the underlying adversary). On the other hand, this allows the reduction to answer all evaluation queries (with overwhelming probability), by the same logic that gives us the negligible correctness error. When proving this formally, there are some subtleties we need to take care of, like: how does the reduction program the random oracle on the challenge set to depend on the selection strings, even though the selection strings determine how other key evaluation queries are answered? We go through a careful hybrid argument to prove that this is indeed possible. The construction and formal proofs can be found in Section 6.

2 Preliminaries

For any positive integer $n \in \mathbb{N}$, we denote $[n] := \{1, \ldots, n\}$. The security parameter will be denoted by λ. For a binary string $x \in \{0,1\}^*$, we denote the length of x by $|x|$. A function $\varepsilon(\lambda)$ is called negligible if for every positive constant c, we have $\varepsilon(\lambda) = o(\lambda^{-c})$. A function $f(\lambda)$ is polynomial if there exists a positive constant c such that $f(\lambda) = O(\lambda^c)$. We denote this by writing $f(\lambda) = \mathrm{poly}(\lambda)$. For any set S, when we write $x \xleftarrow{R} S$ we mean that x is sampled uniformly from S. Given two Boolean circuits $C_0, C_1 : \{0,1\}^n \to \{0,1\}^m$, we say that C_0 and C_1 are functionally equivalent if $C_0(x) = C_1(x)$ for all $x \in \{0,1\}^n$.

Definition 2.1 (Indistinguishability Obfuscation [BGI+12, GGH+13b])**.** Let $\mathcal{C} = \{C_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of polynomial-size circuits. An indistinguishability obfuscator $i\mathcal{O}$ for $\mathcal{C}$ is an efficient algorithm that takes as input the security

parameter λ, a circuit $C \in \mathcal{C}_\lambda$ and outputs a circuit C'. An $i\mathcal{O}$ scheme should satisfy the following properties:

- **Functionality-preserving:** For all security parameters $\lambda \in \mathbb{N}$, all $C \in \mathcal{C}_\lambda$, and all inputs x, we have that $C'(x) = C(x)$ where $C' \leftarrow i\mathcal{O}(1^\lambda, C)$.
- **Security:** We say that $i\mathcal{O}$ is ε-secure if for all polynomial time adversaries $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ and all functionally equivalent $C_0, C_1 \in \mathcal{C}_\lambda$

$$\left|\Pr[\mathcal{A}(\mathsf{st}, i\mathcal{O}(1^\lambda, C_0)) = 1] - \Pr[\mathcal{A}(\mathsf{st}, i\mathcal{O}(1^\lambda, C_1)) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda),$$

  We say that $i\mathcal{O}$ is polynomially secure if $\varepsilon(\lambda) = 1$.

Definition 2.2 (Pseudorandom Generator)**.** Let $\ell_{\mathsf{in}}(\lambda)$ and $\ell_{\mathsf{out}}(\lambda)$ be any polynomials such that $\ell_{\mathsf{in}}(\lambda) < \ell_{\mathsf{out}}(\lambda)$. A pseudorandom generator (PRG) on domain $\mathcal{X} = \left\{\{0,1\}^{\ell_{\mathsf{in}}(\lambda)}\right\}_{\lambda\in\mathbb{N}}$ and range $\mathcal{Y} = \left\{\{0,1\}^{\ell_{\mathsf{out}}(\lambda)}\right\}_{\lambda\in\mathbb{N}}$ is a deterministic polynomial time algorithm $G : \mathcal{X} \to \mathcal{Y}$. We say that G is a ε-secure PRG if for any efficient adversary $\mathcal{A}$ there exists a $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ the following holds:

$$\left|\Pr\left[\mathcal{A}(1^\lambda, G(s)) = 1 : s \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{in}}(\lambda)}\right] - \Pr\left[\mathcal{A}(1^\lambda, x) = 1 : x \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}\right]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda).$$

We say that G is polynomially secure if $\varepsilon(\lambda) = 1$.

By a simple hybrid argument, we can extend the definition to support any number of PRG samples. Specifically, if G is an ε-secure PRG then for any $k \in \mathbb{N}$ and any efficient adversary $\mathcal{A}$ there exists a $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ the following holds:

$$\left|\Pr\left[\mathcal{A}(1^\lambda, G(s_1), \ldots, G(s_k)) = 1\right] - \Pr\left[\mathcal{A}(1^\lambda, x_1, \ldots, x_k) = 1\right]\right| \leq k \cdot \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda),$$

where $s_i \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{in}}(\lambda)}$ and $x_i \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}$ for all $i \in [k]$.

The following definition of a *constrained pseudorandom function* is adapted from [BZ14] and is a generalization of *puncturable PRFs* [BW13, KPTZ13, BGI14]. In our application, it suffices to give only a single constrained key to the adversary. A more general definition where the adversary requests multiple constrained keys has also been considered.

Definition 2.3 (Constrained Pseudorandom Function [BW13, KPTZ13, BGI14])**.** A constrained pseudorandom function for a circuit class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ is a tuple of efficient algorithms $\Pi_{\mathsf{CPRF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Constrain})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}) \to K$: On input a security parameter λ, an input length ℓ_{in}, and an output length ℓ_{out}, the key generation algorithm outputs an evaluation key K.
- $\mathsf{Eval}(K, x) \to y$: On input an evaluation key K (which may be a master key or a constrained key) and a string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, outputs a string $y \in \{0,1\}^{\ell_{\mathsf{out}}} \cup \{\perp\}$.
- $\mathsf{Constrain}(K, C) \to K^{(C)}$: On input a key K, and a binary circuit $C : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ in the class $\mathcal{C}_\lambda$, outputs a constrained evaluation key $K^{(C)}$.

A constrained pseudorandom function should satisfy the following properties:

- **Constrained correctness:** For any $\lambda, \ell_{\mathsf{in}}, \ell_{\mathsf{out}} \in \mathbb{N}$, any Boolean circuit $C : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}$ in $\mathcal{C}_\lambda$, any K in the support of $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$, and any $K^{(C)}$ in the support of $\mathsf{Constrain}(K, C)$, we have

$$\mathsf{Eval}(K^{(C)}, x) = \begin{cases} \mathsf{Eval}(K, x) & C(x) = 1 \\ \perp & C(x) = 0 \end{cases}$$

- **Selective constrained pseudorandomness:** For any adversary $\mathcal{A}$, define the constrained security game with the parameter b as follows:

1. On input a security parameter 1^λ, the adversary $\mathcal{A}$ chooses an input length $1^{\ell_{\text{in}}}$, an output length $1^{\ell_{\text{out}}}$, and a binary circuit $C : \{0,1\}^{\ell_{\text{in}}} \to \{0,1\}$, and sends $(1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}}, C)$ to the challenger.
2. The challenger samples a key $K \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\text{in}}}, 1^{\ell_{\text{out}}})$ and a constrained key $K^{(C)} \leftarrow \mathsf{Constrain}(K, C)$, and sends $K^{(C)}$ to $\mathcal{A}$.
3. The adversary can make evaluation queries x, and the challenger responds with $\mathsf{Eval}(K, x)$.
4. The adversary submits a challenge x^* satisfying the two following requirements:
   - x^* was not queried before.
   - $C(x^*) = 0$, meaning that the constrained key $K^{(C)}$ cannot evaluate x^*.

   The challenger computes $y_0 = \mathsf{Eval}(K, x^*)$, samples $y_1 \xleftarrow{R} \{0,1\}^{\ell_{\text{out}}}$ and sends y_b to $\mathcal{A}$.
5. The adversary outputs a guess b', which is the output of the experiment.

We say that Π_{CPRF} satisfies ε-constrained pseudorandomness if for every efficient adversary $\mathcal{A}$ there exists a $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda).$$

If $\varepsilon(\lambda) = 1$, then Π_{CPRF} satisfies polynomial constrained pseudorandomness.

Constrained PRFs are known in the standard model from sub-exponentially secure iO. In the random oracle model, they can be constructed from polynomially secure iO.

Theorem 2.4 ([BZ14])**.** *Assume sub-exponentially secure iO and sub-exponentially secure OWFs exist. Then there exists a constrained PRF.*

Theorem 2.5 ([DKN+20])**.** *Assume polynomially secure iO and the polynomial security of the LWE assumption. Then there exists a constrained PRF.*

Next, we describe the notion of a *pseudorandom deterministic encryption* (PDE), which can be constructed from pseudorandom functions. We note that a stronger *puncturable* version of PDE due to [KPW17] (which can be constructed from puncturable PRFs) was used in [KWZ22] to construct adaptively secure NIKE for bounded parties. For our construction, the unpunctured notion suffices.

Definition 2.6 (Pseudorandom Deterministic Encryption)**.** A pseudorandom deterministic encryption (PDE) with message space $\{0,1\}^\lambda$ and ciphertexts of length $\ell := \ell(\lambda)$ is a tuple of efficient algorithms $\Pi_{\mathsf{PDE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda) \to \mathsf{ek}$: On input a security parameter λ, the setup algorithm outputs a key ek.
- $\mathsf{Enc}(\mathsf{ek}, m) \to \mathsf{ct}$: On input a key ek and a message $m \in \{0,1\}^\lambda$, the encryption algorithm outputs a ciphertext $\mathsf{ct} \in \{0,1\}^\ell$. This algorithm is deterministic.
- $\mathsf{Dec}(\mathsf{ek}, \mathsf{ct}) \to m$: On input a key ek and a ciphertext $\mathsf{ct} \in \{0,1\}^\ell$, outputs a message $m \in \{0,1\}^\lambda$ or $\perp$.

In addition, Π_{PDE} must satisfy the following properties:

- **Strong Correctness:** For all $\lambda \in \mathbb{N}$, all keys in the support of $\mathsf{Gen}(1^\lambda)$, all $m \in \{0,1\}^\lambda$ and all ciphertexts $\mathsf{ct} \in \{0,1\}^\ell$:
$$\mathsf{Dec}(\mathsf{ek}, \mathsf{ct}) = m \Leftrightarrow \mathsf{Enc}(\mathsf{ek}, m) = \mathsf{ct}$$
- **Ciphertext pseudorandomness:** We define the following security game which is parameterized by a bit b:
  1. The challenger samples the key $\mathsf{ek} \leftarrow \mathsf{Gen}(1^\lambda)$.
  2. The adversary can make *unique* encryption queries. For each query m, the challenger responds with ct which is computed as follows:

- If $b = 0$, the challenger samples $\mathsf{ct} \xleftarrow{R} \{0,1\}^{\ell}$.
- If $b = 1$, the challenger sets $\mathsf{ct} = \mathsf{Enc}(\mathsf{ek}, m)$.

3. The adversary outputs a guess $b' \in \{0,1\}$ which is the output of the experiment.

We say that Π_{PDE} satisfies ε-ciphertext pseudorandomness if for every efficient adversary $\mathcal{A}$ there exists a $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda).$$

If $\varepsilon(\lambda) = 1$, then Π_{PDE} satisfies polynomial ciphertext pseudorandomness.

Fact 2.7 ([KPW17])**.** Assume polynomially secure OWFs exist. Then there exists a PDE scheme.

3 Function-Extractable Hash

In this section, we strengthen the definition of *function-binding hash* [FWW23]. Roughly speaking, a function-binding hash (FBH) is a hash with succinct local openings, where the hash key can bind on a specific function f. Statistical function binding guarantees the following statistical property: if the honestly-generated digest of some string x (with respect to the binding key) can be opened on position i to value $\bar{x}_i$, then there must exist an extension $\bar{x}$ of the value $\bar{x}_i$ such that $f(x) = f(\bar{x})$. We emphasize that this statistical property is required to hold only for honestly-generated digests.

We extend the definition to function-*extractable* hash (FEH) by first adding an extraction algorithm that allows for extracting the value $f(x)$ from the digest on x. Next, we require that for *any* digest, if we extract the value y and open position i to $\bar{x}_i$, then there must exist an extension $\bar{x}$ such that $f(\bar{x}) = y$. This time, we emphasize that the property should hold even for maliciously-generated digests. We call this property *resilient function extraction.*

Definition 3.1 (Function-Extractable Hash)**.** Let $s(\cdot)$, $d(\cdot)$ and $\ell_{\mathsf{out}}(\cdot)$ be polynomials. Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ be an input domain and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ be a codomain where each $y \in \mathcal{Y}_\lambda$ can be represented using $\ell_{\mathsf{out}}(\lambda)$ bits, and let $\mathcal{F} = \{\mathcal{F}_\lambda\}_\lambda$ be a class of functions, where $\mathcal{F}_\lambda$ is a collection of functions $f : \mathcal{X}_\lambda^* \to \mathcal{Y}_\lambda$ each implementable by a circuit of size $s(\lambda) \cdot \mathsf{poly}(n)$ and depth $d(\lambda) + O(\log n)$, where n is the number of input blocks to f. A *function-extractable hash* (FEH) for $\mathcal{F}$ is a tuple of polynomial time algorithms $\mathsf{FEH} = (\mathsf{Setup}, \mathsf{SetupBinding}, \mathsf{Hash}, \mathsf{Verify}, \mathsf{Extract})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, n) \to \mathsf{hk}$: On input a security parameter λ, a number of blocks n (in binary), outputs a hash key hk. We implicitly assume that hk includes λ and n.
- $\mathsf{SetupBinding}(1^\lambda, n, f) \to (\mathsf{hk}, \mathsf{td})$: On input a security parameter λ, a number of blocks n (in binary), a function $f \in \mathcal{F}_\lambda$, outputs a hash key hk and a secret trapdoor td.
- $\mathsf{Hash}(\mathsf{hk}, x) \to (\mathsf{dig}, \pi_1, \ldots, \pi_n)$: On input a hash key hk and a string $x \in \mathcal{X}_\lambda^n$, the hash algorithm outputs a digest dig and n openings $\pi_1, \ldots, \pi_n$. This algorithm is deterministic.
- $\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i, \sigma, \pi) \to b$: On input a hash key hk, a digest dig, an index $i \in [n]$, a symbol $\sigma \in \mathcal{X}_\lambda$, and an opening π, the verification algorithm outputs a bit $b \in \{0,1\}$. The verification algorithm is deterministic.
- $\mathsf{Extract}(\mathsf{td}, \mathsf{dig}) \to y$: On input a trapdoor td and a digest dig, the extraction algorithm outputs a $y \in \mathcal{Y}_\lambda$. This algorithm is deterministic.

We require that Π_{FEH} satisfies the following properties:

- **Efficiency:** For all parameters $\lambda, n \in \mathbb{N}$, all hk in the support of $\mathsf{Setup}(1^\lambda, n)$, all inputs $x \in \mathcal{X}_\lambda^n$:
  - **Efficient setup:** The algorithm $\mathsf{Setup}(1^\lambda, n)$ runs in time at most $\mathsf{poly}(\lambda, \log n)$. Note that this implies hk has size at most $\mathsf{poly}(\lambda, \log n)$.

- **Succinct digest:** The digest dig output by $\mathsf{Hash}(\mathsf{hk}, x)$ has size at most $\mathsf{poly}(\lambda, \log n)$.
- **Succinct openings:** Each opening π_i output by $\mathsf{Hash}(\mathsf{hk}, x)$ has size at most $\mathsf{poly}(\lambda, \log n)$.
- **Efficient verification:** Since we have the size bounds $|\mathsf{hk}|, |\mathsf{dig}|$ and $|\pi|$, and Verify is a polynomial time algorithm, then we get that it runs in time $\mathsf{poly}(\lambda, \log n)$.

- **Correctness:** For all $\lambda, n \in \mathbb{N}$, every $x = x_1 \cdots x_n \in \mathcal{X}_\lambda^n$, and every $i \in [n]$:

$$\Pr\left[\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i, x_i, \pi_i) = 1 \ : \ \begin{array}{c} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, n) \\ (\mathsf{dig}, \pi_1, \ldots, \pi_n) = \mathsf{Hash}(\mathsf{hk}, x) \end{array} \right] = 1.$$

- **Extraction correctness:** For all $\lambda, n \in \mathbb{N}$, every $x = x_1 \cdots x_n \in \mathcal{X}_\lambda^n$, and every $f \in \mathcal{F}_\lambda$:

$$\Pr\left[f(x) = \mathsf{Extract}(\mathsf{td}, \mathsf{dig}) : \ \begin{array}{c} (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{SetupBinding}(1^\lambda, n, f) \\ (\mathsf{dig}, \pi_1, \ldots, \pi_n) \leftarrow \mathsf{Hash}(\mathsf{hk}, x) \end{array} \right] = 1.$$

We additionally require the following security properties:

- **Mode indistinguishability:** For a bit $b \in \{0, 1\}$ and an adversary $\mathcal{A}$, we define the mode indistinguishability game $\mathsf{ExptMI}_{\mathcal{A}}(\lambda, b)$ as follows:
  1. On input 1^λ, the adversary $\mathcal{A}$ sends $n \in \mathbb{N}$ and $f \in \mathcal{F}_\lambda$ to the challenger.
  2. The challenger samples the hash keys $\mathsf{hk}_0 \leftarrow \mathsf{Setup}(1^\lambda, n)$ and $(\mathsf{hk}_1, \mathsf{td}) \leftarrow \mathsf{SetupBinding}(1^\lambda, n, f)$. The challenger gives hk_b to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit b' which is the output of the experiment.

  We say that Π_{FEH} satisfies ε-mode indistinguishability if for every efficient adversary $\mathcal{A}$ there exists a $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda).$$

  If $\varepsilon(\lambda) = 1$, then Π_{FEH} satisfies polynomial mode indistinguishability.

- **Resilient function extraction:** We say that Π_{FEH} satisfies resilient function extraction if for all $\lambda, n \in \mathbb{N}$, all functions $f \in \mathcal{F}_\lambda$, and all $(\mathsf{hk}, \mathsf{td})$ in the support of $\mathsf{SetupBinding}(1^\lambda, n, f)$, as well as any (potentially adversarially computed) digest dig, index $i \in [n]$, block σ and opening π, the following holds:

$$\text{If } \mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i, \sigma, \pi) = 1 \text{ then there exists } x \in \mathcal{X}_\lambda^n \text{ such that } x_i = \sigma \ \wedge \ f(x) = \mathsf{Extract}(\mathsf{td}, \mathsf{dig})$$

Note that resilient function extraction is a worst-case (statistical) guarantee that holds with probability 1, whereas mode indistinguishability is parameterized by ε. When we say that Π_{FEH} is *ε-secure*, we mean that it satisfies ε-mode indistinguishability (and resilient function extraction).

Function-extraction and collision-resistance. In our specific use case, it will be helpful to assume that Π_{FEH} is also collision-resistant. We formally define what it means for a function-extractable hash to be collision-resistant:

Definition 3.2. Let $\Pi_{\mathsf{FEH}} = (\mathsf{Setup}, \mathsf{SetupBinding}, \mathsf{Hash}, \mathsf{Verify}, \mathsf{Extract})$ be a function-extractable hash. We say that Π_{FEH} is ε-collision-resistant if $(\mathsf{Setup}, \mathsf{Hash})$ is an ε-collision-resistant hashas per Definition C.1.

As it turns out, we can generically compose any function-extractable hash with a collision-resistant hash, by simply appending the CRH digest to the FEH digest, in order to get a function-extractable hash that is collision-resistant.

Proposition 3.3. *Assume there exists an $\varepsilon_{\mathsf{CRH}}$-secure collision-resistant hash and an $\varepsilon_{\mathsf{FEH}}$-secure function-extractable hash. Then there exists an $\varepsilon_{\mathsf{FEH}}$-secure function-extractable hash that is $\varepsilon_{\mathsf{CRH}}$-collision-resistant.*

For completeness, we give a proof of Proposition 3.3 in Appendix C.

Remark 3.4. We note that it is not always necessary to go through the composition route of Proposition 3.3 in order to get collision-resistance for function-extractable hash. For example, sometimes the functionality that Π_{FEH} supports readily implies collision-resistance like in the example of somewhere-extractable hash [HW15]. Other times, the construction itself supports collision-resistance (even if the functionality does not imply it). We *believe* that our construction, which is really just the function-binding hash construction from [FWW23] with the added extraction algorithm and resilient security property, is already collision-resistant. However, we do not attempt to prove this because the above approach is simple and sufficient.

3.1 Function-Extractable Hash for Conjunction of Block Functions

In this work, we require function-extractable hash for conjunction of block functions [7]. We start by defining this class of functions:

Definition 3.5 (Conjunction of Block Functions)**.** Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ be an input domain. We say a function $f_g : \mathcal{X}_\lambda^* \to \{0,1\}$ is a conjunction of block functions if we can express it as

$$f_g(x_1, \ldots, x_n) = \bigwedge_{i\in[n]} g(x_i),$$

where the function $g : \mathcal{X}_\lambda \to \{0,1\}$ is a predicate over the alphabet $\mathcal{X}_\lambda$. We say that $\mathcal{F} = \{\mathcal{F}_\lambda\}_\lambda$ is the class of conjunctions of block functions over the alphabet $\mathcal{X}_\lambda$ with block size $s(\lambda)$ and block depth $d(\lambda)$ if $\mathcal{F}_\lambda$ contains all conjunctions of block functions f_g such that the predicate g can be computed using a circuit of size $s(\lambda)$ and depth $d(\lambda)$.

Function-extractable hash from homomorphic encryption. It turns out that the construction of function-binding hash of [FWW23] from *leveled homomorphic encryption* naturally extends to function-extractable hash. The digest in the [FWW23] construction consists of an encryption of the function evaluated on the hashed string. We can simply extract the evaluation if we decrypt the digest. Resilient function extraction can be proved using the decryption correctness of the encryption and a simple inductive argument. Formally:

Theorem 3.6. *Assume there exists a leveled homomorphic encryption scheme. Then there exists a function-extractable hash for conjunction of block functions.*

For completeness, we recall the construction of [FWW23] and provide the proof in Appendix D. This provides a construction of function-extractable hash from the hardness of lattice problems [BV14, BGV14, Bra12, GSW13], or from indistinguishability obfuscation and rerandomizable encryption [CLTV15]. Rerandomizable encryption can be based on a variety of group based assumptions like Decisional Diffie-Hellman [Gam84] or Quadratic Residuosity [GM82].

4 Multiparty Non-interactive Key Exchange

We begin by defining succinct multiparty non-interactive key exchange (NIKE) with a trusted setup. A trustless NIKE can be thought of as a NIKE in which the CRS is an empty string. We require the NIKE to allow an unbounded number of users to publish their public keys (up to 2^λ). We will require that the sizes of the CRS, key pairs and the shared key are all succinct in the set size that the NIKE supports, i.e., if the NIKE supports shared key evaluations of up to n users at a time, then the parameters should grow with $\mathrm{polylog}(n)$. Finally, we allow the NIKE setup to take in a bound c on the number of dynamic honest user corruptions it can support, and allow the parameters to grow polynomially with c.

[7] The construction can be easily extended to disjunction of block functions as well.

Definition 4.1 (Succinct Multiparty Non-interactive Key Exchange)**.** A succinct multiparty non-interactive key exchange scheme (NIKE) is a tuple of polynomial time algorithms (Setup, Publish, KeyGen) with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^c, n) \to \mathsf{crs}$: On input a security parameter λ, a bound on the number of corruptions c, and a bound on the set size n (in binary), the setup algorithm outputs common reference string crs.
- $\mathsf{Publish}(1^\lambda, 1^c) \to (\mathsf{pk}, \mathsf{sk})$: On input a security parameter λ and a bound on the number of corruptions c, the key publishing algorithm outputs a key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{sk}, U) \to \mathsf{sh}$: On input crs, a secret key sk and a set (without repetitions) of public keys U of size at most n, the key generation algorithm outputs a shared key sh. This algorithm is deterministic and can run in time $\mathsf{poly}(\lambda, c, n)$ even if $|U|$ is much smaller than n.

A NIKE scheme is called **trustless** if there is no Setup algorithm, and KeyGen does not take a common reference string.

The algorithms must satisfy the following properties:

- **Perfect correctness:** For all $\lambda, n, c \in \mathbb{N}$, all $t \in [n]$, any crs in the support of $\mathsf{Setup}(1^\lambda, 1^c, n)$, any $j, j' \in [t]$, any two honest key pairs $(\mathsf{pk}_j, \mathsf{sk}_j)$ and $(\mathsf{pk}_{j'}, \mathsf{sk}_{j'})$ in the support of $\mathsf{Publish}(1^\lambda, 1^c)$, and any (not necessarily honest) set of keys $\{\mathsf{pk}_i\}_{i \in [t] \setminus \{j, j'\}}$, it holds that

  $$\mathsf{KeyGen}(\mathsf{crs}, \mathsf{sk}_j, U) = \mathsf{KeyGen}(\mathsf{crs}, \mathsf{sk}_{j'}, U)$$

  where $U = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_t\}$.[8]
- **Succinctness:** For all $\lambda, c, n \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^c, n)$ and all $(\mathsf{pk}, \mathsf{sk})$ in the support of $\mathsf{Publish}(1^\lambda, 1^c)$, the following holds:
  - **CRS succinctness:** $|\mathsf{crs}| = \mathsf{poly}(\lambda, c, \log n)$.
  - **Key pair succinctness:** $|\mathsf{pk}| + |\mathsf{sk}| = \mathsf{poly}(\lambda, c, \log n)$. By our syntax, this follows immediately since Publish is an efficient algorithm that only gets 1^λ and 1^c as input.
  - **Shared key succinctness:** for any honestly-generated key pairs $(\mathsf{pk}_1, \mathsf{sk}_1), \ldots, (\mathsf{pk}_n, \mathsf{sk}_n)$ and any index $i \in [n]$, the shared key $\mathsf{sh} = \mathsf{KeyGen}(\mathsf{crs}, \mathsf{sk}_i, \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\})$ has size at most $\mathsf{poly}(\lambda, c, \log n)$.

Next, we define two notions of security: adaptive security with dynamic corruptions, and adaptive security with static corruptions. In both notions, the adversary is allowed to adaptively pick its challenge; the difference is whether the adversary can dynamically corrupt honest users or not.

Definition 4.2 (Adaptive security with dynamic corruptions)**.** Let $\Pi_{\mathsf{NIKE}} = (\mathsf{Setup}, \mathsf{Publish}, \mathsf{KeyGen})$ be a succinct NIKE scheme. For any adversary $\mathcal{A}$, we define the following security experiment parameterized by a bit $b \in \{0, 1\}$:

1. On input 1^λ, adversary $\mathcal{A}$ sends $1^c, n$ to the challenger. The challenger initializes two empty tables T_{users} and T_{evals} and a counter $i \leftarrow 0$, and sends $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^c, n)$ to $\mathcal{A}$.
2. $\mathcal{A}$ can make the following queries:
   - **Honest user registration:** The challenger increments $i \leftarrow i + 1$, runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Publish}(1^\lambda, 1^c)$, adds the entry $(i, \mathsf{pk}, \mathsf{sk}, \mathsf{uncorrupted})$ to the table T_{users}, and sends pk to $\mathcal{A}$.
   - **Malicious user registration:** The adversary sends a public key pk. If there is *no* entry in T_{users} with the same public key, the challenger increments $i \leftarrow i + 1$, and adds the entry $(i, \mathsf{pk}, \perp, \mathsf{corrupted})$ to the table T_{users}. Otherwise, we say that the adversary is not admissible, and the experiment outputs a random bit $b' \xleftarrow{R} \{0, 1\}$.

[8] We can think of perfect correctness as a strong notion of *adversarial correctness*, where two honest players in a set *always* agree on the shared key, even if the other players in the set are chosen maliciously. We remark that our plain model constructions - as well as many others - satisfy this strong and natural notion.

- **Evaluation:** The adversary sends a set of indices $\mathcal{I} \subseteq [2^\lambda]$ without repetitions, of size at most n, and an index $j^* \in \mathcal{I}$. For each $j \in \mathcal{I}$, the challenger looks for an entry $(j, \mathsf{pk}_j, \cdot, \cdot)$ in T_{users}. For j^*, the challenger looks for an entry with a secret key $(j^*, \mathsf{pk}_{j^*}, \mathsf{sk}_{j^*}, \cdot)$. If any entry is not found or $\mathsf{sk}_{j^*} = \bot$, then we say that the adversary is not admissible, and the experiment outputs a random bit $b' \xleftarrow{R} \{0,1\}$. The challenger then computes the set $U = \left\{\mathsf{pk}_j\right\}_{j \in \mathcal{I}}$ and responds with $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{sk}_{j^*}, U)$. Additionally, the challenger adds $\mathcal{I}$ to the table T_{evals}.
- **Honest user corruption:** The adversary sends an index i'. The challenger looks in T_{users} for an entry $(i', \mathsf{pk}, \mathsf{sk}, \mathsf{uncorrupted})$, updates the entry to $(i', \mathsf{pk}, \mathsf{sk}, \mathsf{corrupted})$ and sends sk to $\mathcal{A}$. If the entry is not found, then we say that the adversary is not admissible, and the experiment outputs a random bit $b' \xleftarrow{R} \{0,1\}$.

The adversary is allowed to make up to c **honest user corruption** queries.

3. At any point, the adversary sends the challenge set (without repetitions) $\mathcal{I}^*$ of size at most n, such that for each $j \in \mathcal{I}^*$ there exists an entry $(j, \mathsf{pk}_j, \mathsf{sk}_j, \mathsf{uncorrupted})$ in T_{users}. The challenger then computes the set $U^* = \left\{\mathsf{pk}_j\right\}_{j \in \mathcal{I}^*}$ and sends y_b, where $y_0 = \mathsf{KeyGen}(\mathsf{crs}, \mathsf{sk}_{\mathsf{idx}_1}, U^*)$ (where idx_1 is the first index in $\mathcal{I}^*$) and y_1 is uniform over the range of KeyGen. If $\mathcal{I}^*$ is in the table T_{evals}, or there exists $j \in \mathcal{I}^*$ such that $(j, \mathsf{pk}_j, \mathsf{sk}_j, \mathsf{uncorrupted})$ is not in T_{users}, then we say that the adversary is not admissible, and the experiment outputs a random bit $b' \xleftarrow{R} \{0,1\}$.
4. The adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

We say that Π_{NIKE} satisfies adaptive security with dynamic corruptions if for every efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

Definition 4.3 (Adaptive security with static corruptions)**.** Let $\Pi_{\mathsf{NIKE}} = (\mathsf{Setup}, \mathsf{Publish}, \mathsf{KeyGen})$ be a succinct NIKE scheme. The adaptive security game with static corruptions is identical to the adaptive security game with dynamic corruptions, except that the adversary sets $c = 0$, namely, no **honest user corruption** queries are allowed[9]. We say that Π_{NIKE} satisfies ε-adaptive security with static corruptions if for every efficient adversary $\mathcal{A}$ there exists a $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda).$$

If $\varepsilon(\lambda) = 1$, then Π_{NIKE} satisfies polynomial adaptive security with static corruptions.

Remark 4.4. We note that Koppula et al. [KWZ22, Section 3] proved that adaptive security *without* **malicious user registration** queries and **evaluation** queries can be generically lifted to support such queries, assuming iO. As we will see, our scheme readily supports such queries so we elect to keep them as part of the definition.

Static corruptions to adaptive corruptions. It turns out that using complexity leveraging, we can prove that if a (succinct) NIKE scheme is sub-exponentially adaptively secure against static corruptions, then it is also adaptively secure against dynamic corruptions. Formally:

Theorem 4.5 (Security against dynamic corruptions)**.** *Let $\delta \in (0,1)$ and Π_{NIKE} be a (succinct) NIKE scheme that is $2^{-\lambda^\delta}$-adaptively secure against* static *corruptions, where λ is the security parameter and n is a bound on the set size, and where the CRS, shared key and key pair all have size polynomial in $\lambda, \log n$. Then for any $c \in \mathbb{N}$, there exists a (succinct) NIKE scheme that is adaptively secure against c dynamic corruptions with polynomial hardness, where the CRS, shared key and key pair all have size polynomial in $\lambda, c, \log n$.*

We provide the proof in Appendix A.

[9] We call this security against static corruptions because it is equivalent to a game where the adversary a priori commits to the set of honest users that it wants to corrupt.

Succinctness to unboundedness. We argue that the succinctness notion we require in Definition 4.1 implies a NIKE for unbounded parties (up to 2^λ parties). The idea is simple: given a NIKE $\Pi_{\mathsf{NIKE}} = (\mathsf{Setup}, \mathsf{Publish}, \mathsf{KeyGen})$ satisfying Definition 4.1 and a bound on the number of corruptions $c \in \mathbb{N}$, we run λ instances of Π_{NIKE}:

- In the setup, we sample $\mathsf{crs}_i \leftarrow \mathsf{Setup}(1^\lambda, 1^c, 2^i)$ for each $i \in [\lambda]$, then set $\mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_\lambda)$. This procedure runs in time $\mathsf{poly}(\lambda, c)$.
- To publish a key pair, it suffices to sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Publish}(1^\lambda, 1^c)$, since the publishing algorithm is independent of n. This procedure runs in time $\mathsf{poly}(\lambda, c)$.
- To generate a shared key for a set U of size $n \in (2^{i-1}, 2^i]$ with the secret key sk, we use crs_i and compute $\mathsf{sh} = \mathsf{KeyGen}(\mathsf{crs}_i, \mathsf{sk}, U)$.

Therefore in this work, we focus on constructing succinct NIKE.

5 Constructing Adaptive NIKE with Static Corruptions

In this section, we present our construction of adaptively secure NIKE against static corruptions. Since no dynamic corruptions are allowed, we omit the parameter c from the input to Setup.

Construction 5.1 (NIKE construction)**.** Our construction uses the following ingredients:

- An indistinguishability obfuscator $i\mathcal{O}$ for the class of circuits that implement the algorithm in Figure 3 as per Definition 2.1.
- A length tripling PRG $G : \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$ as per Definition 2.2. We assume without loss of generality that the all-zero string $\mathbf{0} = 0^{3\lambda}$ is not in the image of G. Looking ahead, $\mathbf{0}$ will be used by KeyGen as the padding block, and this assumption guarantees that no padding block has a G-preimage.
- A pseudorandom deterministic encryption scheme $\Pi_{\mathsf{PDE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as per Definition 2.6 with ciphertexts of length 3λ. Assume that the decryption algorithm PDE.Dec can be implemented by a circuit of size $s(\lambda)$ and depth $d(\lambda)$. This ingredient only appears in the security analysis, not in the construction itself.
- A function-extractable hash $\Pi_{\mathsf{FEH}} = (\mathsf{Setup}, \mathsf{SetupBinding}, \mathsf{Hash}, \mathsf{Verify}, \mathsf{Extract})$ as per Definition 3.1 over the domain $\mathcal{X}_\lambda = \{0,1\}^{3\lambda}$. We assume that Π_{FEH} supports conjunction of block functions of size $s(\lambda)$ and depth $d(\lambda)$. Denote the digest length of Π_{FEH} by $\ell_{\mathsf{FEH}} := \ell_{\mathsf{FEH}}(\lambda, n)$. And let $\mathcal{C}_\lambda = \left\{\mathsf{FEH.Extract}(\mathsf{td}, \cdot) : \{0,1\}^{\ell_{\mathsf{FEH}}} \to \{0,1\}\right\}_{\mathsf{td}}$ be the class of circuits that implements the extraction algorithm.
- A constrained PRF scheme $\Pi_{\mathsf{CPRF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Constrain})$ as per Definition 2.3 for a class of circuits $\mathcal{C}_\lambda$ that can implements the extraction algorithm of the function-extractable hash.

Throughout this construction, we assume that any set (without repetitions) U is represented by the *lexicographically ordered* tuple $(\mathsf{pk}_1, \ldots, \mathsf{pk}_t)$, where $U = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_t\}$. The algorithms are defined as follows:

- $\mathsf{Setup}(1^\lambda, n)$:
  1. Samples a PRF key $K \leftarrow \mathsf{CPRF.Setup}(1^\lambda, 1^{\ell_{\mathsf{FEH}}}, 1^\lambda)$.
  2. Samples a FEH key $\mathsf{hk} \leftarrow \mathsf{FEH.Setup}(1^\lambda, n)$.
  3. Samples $P \leftarrow i\mathcal{O}(\mathsf{crsProg}[K, \mathsf{hk}])$ where crsProg is described in Figure 3.
  4. Outputs $\mathsf{crs} = (\mathsf{hk}, P)$.

Constants: A constrained PRF key K (which could be a master key or a constrained key), and a FEH hash key hk.
Inputs: A digest dig, an index $j \in [n]$, a public key pk_j, a secret sk_j, and an opening π_j.

1. If $\mathsf{FEH.Verify}(\mathsf{hk}, \mathsf{dig}, j, \mathsf{pk}_j, \pi_j) = 0$, outputs $\perp$.
2. If $G(\mathsf{sk}_j) \neq \mathsf{pk}_j$, outputs $\perp$.
3. Outputs $\mathsf{CPRF.Eval}(K, \mathsf{dig})$.

Figure 3: Program $\mathsf{crsProg}[K, \mathsf{hk}]$.

- $\mathsf{Publish}(1^\lambda)$: samples a uniform $\mathsf{sk} \xleftarrow{R} \{0,1\}^\lambda$, sets $\mathsf{pk} = G(\mathsf{sk})$, and outputs $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{sk}, U)$:
  1. Parses $\mathsf{crs} = (\mathsf{hk}, P)$.
  2. Parses $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_t)$, where $t = |U|$. If $\exists i$ such that $\mathsf{pk}_i = 0^{3\lambda}$, outputs $\perp$.
  3. Looks for $j \in [t]$ such that $\mathsf{pk}_j = G(\mathsf{sk})$. If no such j exists, outputs $\perp$.
  4. Pads the string $\widetilde{U} = U \circ \mathbf{0}^{n-t}$, where $\mathbf{0} = 0^{3\lambda}$.
  5. Hashes $(\mathsf{dig}, \pi_1, \ldots, \pi_n) = \mathsf{FEH.Hash}(\mathsf{hk}, \widetilde{U})$.
  6. Outputs $P(\mathsf{dig}, j, \mathsf{pk}_j, \mathsf{sk}, \pi_j)$.

Perfect correctness. Observe that for any honestly generated key pair pk, sk that is output by Publish, the check $G(\mathsf{sk}) = \mathsf{pk}$ follows. In addition, the program $\mathsf{crsProg}[K, \mathsf{hk}]$ from Figure 3 outputs the same string sh on any dig, as long as it is given a valid opening π_j to some key pk_j in position j, as well as the secret key sk_j corresponding to pk_j. Therefore for any set U and any two parties $\mathsf{pk}_j, \mathsf{pk}_{j'} \in U$ with the corresponding secret keys $\mathsf{sk}_j, \mathsf{sk}_{j'}$, by the perfect correctness of Π_{FEH} and the functionality preserving of $i\mathcal{O}$, we have $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{sk}_j, U) = \mathsf{KeyGen}(\mathsf{crs}, \mathsf{sk}_{j'}, U)$ and perfect completeness of Π_{NIKE} follows.

Succinctness. Key pair succinctness follows trivially since Publish is an efficient algorithm that just samples $\mathsf{sk} \xleftarrow{R} \{0,1\}^\lambda$ and outputs $\mathsf{pk} = G(\mathsf{sk}), \mathsf{sk}$. Shared key succinctness also follows immediately by the functionality preserving of $i\mathcal{O}$. Namely, the shared key is computed by an obfuscation of the program $\mathsf{crsProg}[K, \mathsf{hk}]$ from Figure 3, which just outputs the evaluation of the CPRF, and the output length of the CPRF is set to be λ. Finally, the CRS consists of the hash key hk and the obfuscated program P. The hash key hk has length $\mathsf{poly}(\lambda, \log n)$ by the succinctness of Π_{FEH}. The program crsProg has the following complexity:

1. The first step of verifying the opening of FEH can be implemented by in time $\mathsf{poly}(\lambda, \log n)$ by the succinctness of Π_{FEH}.
2. The second step consists of evaluating a length tripling PRG on a seed of length λ, therefore can be implemented in time $\mathsf{poly}(\lambda)$.
3. The final step can be computed in time $\mathsf{poly}(\lambda, \ell_{\mathsf{FEH}})$ which is $\mathsf{poly}(\lambda, \log n)$ by the succinctness of Π_{FEH}. Note that this step might use the constrained key of Π_{CPRF}, but this key has size $\mathsf{poly}(\lambda, \ell_{\mathsf{FEH}})$ which is $\mathsf{poly}(\lambda, \log n)$.

In total, crsProg can be implemented by a circuit of size $\mathsf{poly}(\lambda, \log n)$. Since this is the class of circuits that $i\mathcal{O}$ needs to support, then the obfuscated program P has size $\mathsf{poly}(\lambda, \log n)$. In total, the CRS is also succinct.

5.1 Security Analysis

In this section, we prove that Construction 5.1 is adaptively secure against static corruptions. We prove the following theorem:

Theorem 5.2. *Assume that $i\mathcal{O}$ is an ε-secure indistinguishability obfuscator, G is an ε-secure PRG, Π_{CPRF} satisfies ε-constrained pseudorandomness, Π_{PDE} satisfies ε-ciphertext pseudorandomness, and Π_{FEH} satisfies ε-collision-resistance, ε-mode indistinguishability, and resilient function extraction. Then Construction 5.1 satisfies ε-adaptive security against static corruptions.*

Let $\mathcal{A}$ be an adversary for the adaptive security with static corruptions game. Throughout this section, we assume that $\mathcal{A}$ is admissible. We define the following sequence of hybrids starting with the real security game Real^b.

- Real^b: This is the real security game with parameter b.

  1. **Setup Phase:**
     (a) On input 1^λ, the adversary $\mathcal{A}$ sends n to the challenger.
     (b) The challenger initializes two empty tables $T_{\mathsf{users}} \leftarrow \emptyset$ and $T_{\mathsf{evals}} \leftarrow \emptyset$, and a counter $i \leftarrow 0$.
     (c) Samples a PRF key $K \leftarrow \mathsf{CPRF.Setup}(1^\lambda, 1^{\ell_{\mathsf{FEH}}}, 1^\lambda)$.
     (d) Samples a FEH key $\mathsf{hk} \leftarrow \mathsf{FEH.Setup}(1^\lambda, n)$.
     (e) Samples $P \leftarrow i\mathcal{O}(\mathsf{crsProg}[K, \mathsf{hk}])$ where crsProg is described in Figure 3.
     (f) The challenger sends $\mathsf{crs} = (\mathsf{hk}, P)$ to $\mathcal{A}$.
  2. **Query Phase:** $\mathcal{A}$ can adaptively make the following queries.
     – **Honest user registration:**
       (a) The challenger increments $i \leftarrow i + 1$.
       (b) The challenger samples a uniform $\mathsf{sk} \xleftarrow{R} \{0,1\}^\lambda$ and sets $\mathsf{pk} = G(\mathsf{sk})$.
       (c) The challenger adds the entry $(i, \mathsf{pk}, \mathsf{sk}, \mathsf{uncorrupted})$ to the table T_{users}.
       (d) The challenger sends pk to $\mathcal{A}$.
     – **Malicious user registration:**
       (a) The adversary sends a public key pk.
       (b) The challenger increments $i \leftarrow i + 1$.
       (c) The challenger adds the entry $(i, \mathsf{pk}, \perp, \mathsf{corrupted})$ to the table T_{users}.
     – **Evaluation:**
       (a) The adversary sends a set of indices $\mathcal{I}$ and a specific index $j^* \in \mathcal{I}$.
       (b) The challenger retrieves $(j, \mathsf{pk}_j, \cdot, \cdot)$ for all $j \in \mathcal{I}$. If $\mathsf{pk}_j = 0^{3\lambda}$ for some $j \in \mathcal{I}$, the challenger responds with $\perp$ (consistently with KeyGen, which rejects any set containing the padding block).
       (c) The challenger forms the ordered set $U = (\mathsf{pk}_j)_{j \in \mathcal{I}}$, extends it to $\widetilde{U}$ by padding with zeros, hashes $(\mathsf{dig}, \cdot) = \mathsf{FEH.Hash}(\mathsf{hk}, \widetilde{U})$ and sends $\mathsf{CPRF.Eval}(K, \mathsf{dig})$ to $\mathcal{A}$.[10]
       (d) Additionally, the challenger adds U to the table T_{evals}.
  3. **Challenge Phase:**
     (a) At any point, $\mathcal{A}$ sends the challenge set of indices $\mathcal{I}^*$.
     (b) The challenger computes the ordered set $U^* = (\mathsf{pk}_j)_{j \in \mathcal{I}^*}$.
     (c) The challenger computes the challenge value y_b.
       – If $b = 0$, the challenger extends U^* to $\widetilde{U}$ by padding with zeros, and then hashes $(\mathsf{dig}^*, \cdot) = \mathsf{FEH.Hash}(\mathsf{hk}, \widetilde{U})$ and sets $y_0 = \mathsf{CPRF.Eval}(K, \mathsf{dig}^*)$.

[10] We justify applying the PRF directly on dig instead of feeding everything to crsProg by the functionality-preserving of $i\mathcal{O}$ and the fact that the key pair $(\mathsf{pk}_{j^*}, \mathsf{sk}_{j^*})$ and the digest were computed honestly. As we will see later on, this is a crucial modification since it allows the challenger to answer queries even when the challenger-generated keys are not computed according to the specification of $\mathsf{Publish}(1^\lambda)$.

- If $b = 1$, the challenger samples a uniformly random key $y_1 \xleftarrow{R} \{0,1\}^\lambda$.

(d) The challenger sends y_b to $\mathcal{A}$.

4. **Output Phase:**
   (a) $\mathcal{A}$ is allowed to continue making queries as in the **Query Phase**.
   (b) $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

- Hyb_1^b: This is the same as Real^b, except that the experiment aborts and outputs a random bit if the hash of the challenge set is equal to the hash of any set in T_{evals}. Specifically:
  - **Query Phase:** for each **Evaluation** query $\mathcal{I}$, the challenger adds to the table T_{evals} the entry $(\mathcal{I}, \mathsf{dig})$.
  - **Challenge Phase:** The challenger always computes dig^*. If there exists an entry $(\cdot, \mathsf{dig}^*)$ in T_{evals}, the experiment aborts and outputs a random bit $b' \xleftarrow{R} \{0,1\}$.
- Hyb_2^b: This is the same as Hyb_1^b, except that the challenger responds to **honest user registration** with random public keys instead of the output of G. Specifically, the challenger makes the following changes:
  - **Query Phase:** The challenger modifies how it responds to **honest user registration** queries as follows:
    1. The challenger increments $i \leftarrow i + 1$.
    2. The challenger samples $\mathsf{pk}_i \xleftarrow{R} \{0,1\}^{3\lambda}$.
    3. The challenger adds the entry $(i, \mathsf{pk}, \perp, \mathsf{uncorrupted})$ to the table T.
    4. The challenger sends pk to $\mathcal{A}$.
- Hyb_3^b: This is the same as Hyb_2^b, except that the challenger responds to **honest user registration** with what we call *labeled public keys*. Specifically, the challenger makes the following changes:
  - **Setup Phase:** The challenger additionally samples a PDE key $\mathsf{ek} \leftarrow \mathsf{PDE.Gen}(1^\lambda)$ and a uniform string $\mathsf{pad} \xleftarrow{R} \{0,1\}^{3\lambda}$.
  - **Query Phase:** The challenger modifies how it responds to **honest user registration** queries as follows:
    1. The challenger sets $\mathsf{pk}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i) \oplus \mathsf{pad}$.
- Hyb_4^b: This is the same as Hyb_3^b, except that the FEH hash key is binding on the function that checks whether all the hashed keys are labeled. Specifically, the challenger makes the following changes:
  - **Setup Phase:** The challenger samples the FEH hash key as $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{FEH.SetupBinding}(1^\lambda, n, f_g)$, where $f_g(x_1, \ldots, x_n) = \bigwedge_{i \in [n]} g(x_i)$ and

$$g(x) = \begin{cases} 1 & x = \mathbf{0} \text{ or } \mathsf{PDE.Dec}(\mathsf{ek}, x \oplus \mathsf{pad}) \neq \perp \\ 0 & \text{otherwise} \end{cases}$$

    Here $\mathbf{0} = 0^{3\lambda}$ is the padding block used by KeyGen. In other words, g accepts a block if it is either a labeled key or the zero padding, so that f_g extracts to 1 exactly when every *non-padding* block is a labeled key.
- Hyb_5^b: This is the same as Hyb_4^b, except that the PRF key is constrained so that it does *not* evaluate FEH digests that extract to 1, i.e., when all the hashed keys are labeled. Specifically, the challenger makes the following changes:
  - **Setup Phase:** The challenger constrains the PRF key $K' = \mathsf{CPRF.Constrain}(K, C)$ where

$$C(\mathsf{dig}) = \neg \mathsf{FEH.Extract}(\mathsf{td}, \mathsf{dig})$$

Lemma 5.3. *Assume that Π_{FEH} is ε-collision-resistant as per Definition 3.2. Then for any efficient $\mathcal{A}$ and any $b \in \{0,1\}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that*

$$\left| \Pr[\mathsf{Real}^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^b(\mathcal{A}) = 1] \right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Proof. Let $\mathcal{A}$ be an adversary for the adaptive security with static corruptions against Construction 5.1. The difference between Real^b and Hyb_1^b is the added abort condition. The idea is that the abort only happens if the adversary manages to find a collision in Π_{FEH}. Formally, we define the following event

$$E \equiv \left(\exists(\widetilde{U}_i, \mathsf{dig}_i) \in T_{\mathsf{evals}} \text{ s.t. } \mathsf{dig}^* = \mathsf{dig}_i\right).$$

By the specification of the hybrids,

$$\Pr[\mathsf{Hyb}_1^b(\mathcal{A}) = 1] = \Pr[\mathsf{Real}^b(\mathcal{A}) = 1 \ \wedge \ \neg E] + \frac{1}{2}\Pr[E]$$

By basic probability, we have

$$\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] - \Pr[E] \leq \Pr[\mathsf{Real}^b(\mathcal{A}) = 1 \ \wedge \ \neg E] \leq \Pr[\mathsf{Real}^b(\mathcal{A}) = 1].$$

So in total, we have that $\left|\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^b(\mathcal{A}) = 1]\right|$ is bounded by

$$\left|\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] - \Pr[\mathsf{Real}^b(\mathcal{A}) = 1 \ \wedge \ \neg E]\right| + \left|\Pr[\mathsf{Real}^b(\mathcal{A}) = 1 \ \wedge \ \neg E] - \Pr[\mathsf{Hyb}_1^b(\mathcal{A}) = 1]\right| \leq \frac{3}{2}\Pr[E].$$

Therefore if we prove that there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that $\Pr[E] \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$, then the claim follows. Let Λ and $\mathsf{negl}(\cdot)$ be the parameter guaranteed by the ε-collision resistance of Π_{FEH}. We construct an adversary for collision-resistance game against Π_{FEH} as follows:

1. Algorithm $\mathcal{A}$ sends n to $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ gets the hash key hk from the Π_{FEH} challenger.
3. Algorithm $\mathcal{B}$ initializes two empty tables $T_{\mathsf{users}} \leftarrow \emptyset$ and $T_{\mathsf{evals}} \leftarrow \emptyset$, and a counter $i \leftarrow 0$.
4. Algorithm $\mathcal{B}$ samples the following:
   (a) Samples a PRF key $K \leftarrow \mathsf{CPRF.Setup}(1^\lambda, 1^{\ell_{\mathsf{FEH}}}, 1^\lambda)$.
   (b) Samples $P \leftarrow i\mathcal{O}(\mathsf{crsProg}[K, \mathsf{hk}])$ where crsProg is described in Figure 3.
5. Algorithm sends $\mathsf{crs} = (\mathsf{hk}, P)$ to $\mathcal{A}$.
6. For any **honest user registration** and **corrupted users registration** query that $\mathcal{A}$ makes, algorithm $\mathcal{B}$ has all the information to simulate the answer exactly as in the adaptive security game.
7. Whenever $\mathcal{A}$ makes an **evaluation** query on a valid index set $\mathcal{I}_i$ of size $t \leq n$, algorithm $\mathcal{B}$ computes $U_i = (\mathsf{pk}_j)_{j \in \mathcal{I}_i}$. If some key in U_i equals $\mathbf{0}$, then $\mathcal{B}$ responds with $\perp$ and does not record the query (just as the challenger does). Otherwise, $\mathcal{B}$ extends the string to $\widetilde{U}_i = U_i \circ \mathbf{0}^{n-t}$, hashes $(\mathsf{dig}_i, \cdot) = \mathsf{FEH.Hash}(\mathsf{hk}, \widetilde{U}_i)$ and sends $\mathsf{sh} = \mathsf{CPRF.Eval}(K, \mathsf{dig}_i)$ to $\mathcal{A}$. Algorithm $\mathcal{B}$ then adds $(\widetilde{U}_i, \mathsf{dig}_i)$ to T_{evals}.
8. When $\mathcal{A}$ submits the challenge $\mathcal{I}^*$ of size t, algorithm $\mathcal{B}$ does the following:
   - Computes $U^* = (\mathsf{pk}_j)_{j \in \mathcal{I}^*}$, and $\widetilde{U}^* = U^* \circ \mathbf{0}^{n-t}$, then hashes $(\mathsf{dig}^*, \cdot) = \mathsf{FEH.Hash}(\mathsf{hk}, \widetilde{U}^*)$.
   - If there exists an entry $(\widetilde{U}_i, \mathsf{dig}_i)$ in T_{evals} such that $\mathsf{dig}^* = \mathsf{dig}_i$, then $\mathcal{B}$ sends $\widetilde{U}^*, \widetilde{U}_i$ to the Π_{FEH} challenger. Otherwise, $\mathcal{B}$ aborts.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. Clearly $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in the adaptive security game of Construction 5.1. If the event E occurs in the simulation, then by definition, $\mathcal{B}$ outputs $\widetilde{U}_i$ and $\widetilde{U}^*$ such that $\mathsf{FEH.Hash}(\mathsf{hk}, \widetilde{U}^*) = \mathsf{FEH.Hash}(\mathsf{hk}, \widetilde{U}_i)$. We claim that moreover $\widetilde{U}^* \neq \widetilde{U}_i$, so that this pair constitutes a genuine collision. Here we crucially rely on the fact that KeyGen (and accordingly the challenger when answering **evaluation** and **challenge** queries) rejects any set that contains the padding block $\mathbf{0} = 0^{3\lambda}$. As a result, every block

of an admissible set $\widetilde{U}$ is non-zero exactly on the positions corresponding to the keys of the underlying set and is $\mathbf{0}$ exactly on the padding positions; hence the padded representation $U \mapsto \widetilde{U}$ is *injective*, and distinct index sets yield distinct padded strings. Since $\mathcal{A}$ is admissible, the challenge index set $\mathcal{I}^*$ differs from the index set $\mathcal{I}_i$ of every prior evaluation query (otherwise $\mathcal{I}^* \in T_{\mathsf{evals}}$), and therefore $\widetilde{U}^* \neq \widetilde{U}_i$. Without this rejection rule the implication would fail: for example, the distinct index sets $\mathcal{I}^*$ and $\mathcal{I}^* \cup \{i_0\}$, where i_0 is the index of a (malicious) key equal to $\mathbf{0}$, would pad to the *same* string $\widetilde{U}$, allowing the adversary to recover the challenge key through an admissible evaluation query. By the ε-collision-resistance of Π_{FEH}, the event E thus happens with probability at most $\varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$ for all $\lambda > \Lambda$, and so the claim follows. □

Lemma 5.4. *Assume that G is an ε-secure PRG. Then for any efficient $\mathcal{A}$ that makes $q(\lambda)$ **honest user registration** queries and any $b \in \{0,1\}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that*

$$\left|\Pr[\mathsf{Hyb}_1^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^b(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Proof. Let $\mathcal{A}$ be an adversary for the adaptive security with static corruptions against Construction 5.1, that makes at most $q := q(\lambda)$ **honest user registration** queries. We construct an adversary $\mathcal{B}$ for the PRG security game with q samples as follows:

1. Algorithm $\mathcal{A}$ sends n to $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ gets as input λ and samples $x_1, \ldots, x_q$.
3. Algorithm $\mathcal{B}$ initializes a counter $i \leftarrow 0$, simulates the setup of Construction 5.1, and sends the crs to $\mathcal{A}$.
4. Whenever $\mathcal{A}$ makes an **honest user registration**, algorithm $\mathcal{B}$ increments $i \leftarrow i + 1$, and sends x_i to $\mathcal{A}$.
5. Algorithm $\mathcal{B}$ can respond to all other queries, as well as the challenge, exactly as in the experiments Hyb_1 and Hyb_2. Observe that here, we crucially rely on the fact that **honest user corruption** queries are not allowed, since otherwise algorithm $\mathcal{B}$ would have to find a preimage $s_i \in \{0,1\}^\lambda$ with respect to the PRG G for an arbitrary string $x_i \in \{0,1\}^{3\lambda}$.
6. When $\mathcal{A}$ outputs a bit b', algorithm $\mathcal{B}$ forwards it as its own output.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. If $x_1, \ldots, x_q$ are PRG samples, then for each $i \in [q]$ we have $x_i = G(s_i)$ where $s_i \leftarrow \{0,1\}^\lambda$. In this case, the **honest user registration** queries are answered exactly as in Hyb_1. If $x_1, \ldots, x_q$ are uniform samples from $\{0,1\}^{3\lambda}$, then the **honest user registration** queries are answered exactly as in Hyb_2. Therefore the advantage of $\mathcal{B}$ in the PRG security game is exactly

$$\left|\Pr[\mathsf{Hyb}_1^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^b(\mathcal{A}) = 1]\right|$$

By ε-security of G, there exists Λ such that for all $\lambda \geq \Lambda$ and a negligible function $\mathsf{negl}(\lambda)$, this probability is bounded by $q(\lambda) \cdot \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$. Since $\mathcal{A}$ is efficient, then $q(\lambda)$ is polynomial and therefore can be absorbed into the negligible function, and the claim follows. □

Lemma 5.5. *Assume that Π_{PDE} satisfies ε-ciphertext pseudorandomness. Then for any efficient $\mathcal{A}$ and any $b \in \{0,1\}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that*

$$\left|\Pr[\mathsf{Hyb}_2^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^b(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Proof. Let $\mathcal{A}$ be an adversary for the adaptive security with static corruptions. We construct an adversary $\mathcal{B}$ for the ciphertext pseudorandomness game of Π_{PDE} as follows:

1. Algorithm $\mathcal{A}$ sends n to $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes a counter $i \leftarrow 0$, simulates the setup of Construction 5.1, and sends the crs to $\mathcal{A}$.

3. For any **malicious user registration**, **evaluation**, or **challenge** query, algorithm $\mathcal{B}$ has all the information to simulate the answer exactly as in the experiments Hyb_2^b and Hyb_3^b, as these steps are identical in both hybrids. Similar to before, we crucially rely on the fact that **honest user corruption** queries are not allowed.

4. Whenever $\mathcal{A}$ makes an **honest user registration** query:

   (a) Algorithm $\mathcal{B}$ increments $i \leftarrow i + 1$, and sends i as a query to the Π_{PDE} challenger. If $i \geq 2^\lambda$, algorithm $\mathcal{B}$ aborts the experiment and outputs a random $b' \xleftarrow{\text{R}} \{0,1\}$. Note that this ensures that the encryption queries that $\mathcal{B}$ makes are unique, which is necessary for the Π_{PDE} ciphertext pseudorandomness security game.

   (b) The Π_{PDE} challenger responds with a ciphertext $\mathsf{ct}_i \in \{0,1\}^{3\lambda}$.

   (c) Algorithm $\mathcal{B}$ computes $\mathsf{pk}_i = \mathsf{ct}_i \oplus \mathsf{pad}$.

   (d) $\mathcal{B}$ adds $(i, \mathsf{pk}_i, \bot, \text{uncorrupted})$ to T_{users} and sends pk_i to $\mathcal{A}$.

5. When $\mathcal{A}$ outputs a bit b', algorithm $\mathcal{B}$ forwards b' as its own output.

If $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. Since $\mathcal{A}$ is efficient, then it makes a polynomial number of **honest user registration** queries, therefore there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$, algorithm $\mathcal{B}$ never aborts. We analyze the simulation of $\mathcal{B}$:

- If the ciphertexts are random, i.e., $\mathsf{ct}_i \xleftarrow{\text{R}} \{0,1\}^{3\lambda}$, then the public keys $\mathsf{pk}_i = \mathsf{ct}_i \oplus \mathsf{pad}$ are uniformly random strings, therefore $\mathcal{B}$ perfectly simulates of the experiment $\mathsf{Hyb}_2^b(\mathcal{A})$.
- If the ciphertexts are encryptions of the corresponding indices, i.e., $\mathsf{ct}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i)$, then the public keys $\mathcal{B}$ computes are $\mathsf{pk}_i = \mathsf{PDE.Enc}(\mathsf{ek}, i) \oplus \mathsf{pad}$. This is a perfect simulation of the experiment Hyb_3^b.

Therefore, the advantage of $\mathcal{B}$ in the Π_{PDE} security game is exactly

$$\left|\Pr[\mathsf{Hyb}_2^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^b(\mathcal{A}) = 1]\right|$$

By the ε-ciphertext pseudorandomness of Π_{PDE}, there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ this probability is bounded by $\varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$, and the claim follows. □

Lemma 5.6. *Assume that Π_{FEH} satisfies ε-mode indistinguishability. Then for any efficient $\mathcal{A}$ and any $b \in \{0,1\}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that*

$$\left|\Pr[\mathsf{Hyb}_3^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4^b(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Proof. Let $\mathcal{A}$ be an adversary for the adaptive security with static corruptions. We construct an adversary $\mathcal{B}$ for the mode indistinguishability game of Π_{FEH} as follows:

1. Algorithm $\mathcal{A}$ sends n to $\mathcal{B}$.

2. Algorithm $\mathcal{B}$ initializes tables $T_{\mathsf{users}}, T_{\mathsf{evals}}$ and counter $i \leftarrow 0$, and samples $K, \mathsf{ek}, \mathsf{pad}$ as in the setup of Hyb_3^b and Hyb_4^b.

3. Algorithm $\mathcal{B}$ defines the function g and f_g based on ek and pad:

$$g(x) = \begin{cases} 1 & x = \mathbf{0} \text{ or } \mathsf{PDE.Dec}(\mathsf{ek}, x \oplus \mathsf{pad}) \neq \bot \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad f_g(x_1, \ldots, x_n) = \bigwedge_{i \in [n]} g(x_i)$$

   and sends n and the function f_g to its Π_{FEH} challenger.

4. The Π_{FEH} challenger responds with a hash key hk.

5. Algorithm $\mathcal{B}$ samples $P \leftarrow i\mathcal{O}(\mathsf{crsProg}[K, \mathsf{hk}])$ where crsProg is described in Figure 3 and sends $\mathsf{crs} = (\mathsf{hk}, P)$ to $\mathcal{A}$.

6. Algorithm $\mathcal{B}$ answers all queries as well as the challenge exactly as specified in the descriptions of Hyb_3^b and Hyb_4^b. Note that the query-answering logic is identical in both hybrids. Similar to before, we crucially rely on the fact that **honest user corruption** queries are not allowed.

7. When $\mathcal{A}$ outputs its guess b', algorithm $\mathcal{B}$ forwards b' as its own output to the Π_{FEH} challenger.

If $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. We analyze the simulation:

- If $\mathsf{hk} \leftarrow \mathsf{FEH.Setup}(1^\lambda, n)$ then $\mathcal{B}$ perfectly simulates the experiment $\mathsf{Hyb}_3^b(\mathcal{A})$.
- If $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{FEH.SetupBinding}(1^\lambda, n, f_g)$ then $\mathcal{B}$ perfectly simulates the experiment Hyb_4^b for $\mathcal{A}$.

Therefore, the advantage of $\mathcal{B}$ in the Π_{FEH} mode indistinguishability game is exactly

$$\left|\Pr[\mathsf{Hyb}_3^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4^b(\mathcal{A}) = 1]\right|$$

By the ε-mode indistinguishability of Π_{FEH}, there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ this probability is bounded by $\varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$, and the claim follows. □

Lemma 5.7. *Assume that $i\mathcal{O}$ is an ε-secure indistinguishability obfuscator, Π_{FEH} satisfies resilient function extraction, Π_{CPRF} satisfies constrained correctness, and Π_{PDE} satisfies strong correctness. Then for any efficient $\mathcal{A}$ and any $b \in \{0, 1\}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that*

$$\left|\Pr[\mathsf{Hyb}_4^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5^b(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Proof. Let $\mathcal{A}$ be an adversary for the adaptive security with static corruptions. We construct an adversary $\mathcal{B}$ for the security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends n to $\mathcal{B}$.

2. Algorithm $\mathcal{B}$ initializes tables $T_{\mathsf{users}}, T_{\mathsf{evals}}$ and counter $i \leftarrow 0$, and samples $K, \mathsf{hk}, \mathsf{td}, \mathsf{ek}, \mathsf{pad}$ as in the setup of Hyb_4^b and Hyb_5^b.

3. Algorithm $\mathcal{B}$ additionally computes the constrained key $K' = \mathsf{CPRF.Constrain}(K, C)$, where the circuit is define as $C(\mathsf{dig}) = \neg\mathsf{FEH.Extract}(\mathsf{td}, \mathsf{dig})$.

4. Algorithm $\mathcal{B}$ sends a challenge to the $i\mathcal{O}$ challenger consisting of two circuits C_0, C_1 which compute the programs $\mathsf{crsProg}[K, \mathsf{hk}]$ and $\mathsf{crsProg}[K', \mathsf{hk}]$, respectively. Note that both circuits are in the class that supported by $i\mathcal{O}$ since they are instantiations of the algorithms from Figure 3.

5. The challenger responds with an obfuscated program P.

6. Algorithm $\mathcal{B}$ sends $\mathsf{crs} = (\mathsf{hk}, P)$ to $\mathcal{A}$.

7. Algorithm $\mathcal{B}$ answers all queries as well as the challenge exactly as specified in the descriptions of Hyb_4^b and Hyb_5^b. Note that the query-answering logic is identical in both hybrids. Similar to before, we crucially rely on the fact that **honest user corruption** queries are not allowed.

8. When $\mathcal{A}$ outputs its guess b', algorithm $\mathcal{B}$ forwards b' as its own output to the $i\mathcal{O}$ challenger.

If $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. Clearly, if P is an obfuscation of C_0 then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_4^b(\mathcal{A})$, and if P is an obfuscation of C_1 then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_5^b(\mathcal{A})$. For $\mathcal{B}$ to be a valid adversary for the indistinguishability obfuscation game, C_0 and C_1 have to be equivalent. If C_0 and C_1 are *not* functionally equivalent with probability at most $\delta(\lambda)$, then the advantage of $\mathcal{B}$ against $i\mathcal{O}$ is:

$$\left|\Pr[\mathsf{Hyb}_4^b(\mathcal{A})=1]-\Pr[\mathsf{Hyb}_5^b(\mathcal{A})=1]\right|-\delta(\lambda)$$

All that remains is to prove that C_0, C_1 are functionally equivalent with probability $\varepsilon(\lambda)\cdot\mathsf{negl}(\lambda)$ and the claim would follow by the ε-security of $i\mathcal{O}$. We start by proving the following statistical claim:

Claim 5.8. *Let* A *be the following event over the reduction:*

$$\exists\ \mathsf{dig}, j, \pi, s \text{ such that } \mathsf{FEH.Extract}(\mathsf{td},\mathsf{dig})=1\ \wedge\ \mathsf{FEH.Verify}(\mathsf{hk},\mathsf{dig},j,G(s),\pi)=1.$$

If Π_{FEH} *satisfies resilient function extraction and* Π_{PDE} *satisfies strong correctness then:*

$$\Pr\left[\mathsf{A}:\mathsf{pad}\xleftarrow{R}\{0,1\}^{3\lambda}\right]=2^{-\lambda}.$$

Proof. Fix $\lambda, n\in\mathbb{N}$. Fix ek in the support of $\mathsf{PDE.Gen}(1^\lambda)$ and $(\mathsf{hk},\mathsf{td})$ in the support of $\mathsf{FEH.SetupBinding}(1^\lambda,n,f_g)$. We start by observing that for each $x\in\{0,1\}^{3\lambda}$, we have

$$\Pr\left[\exists s\in\{0,1\}^\lambda \text{ such that } G(s)\oplus\mathsf{pad}=x:\mathsf{pad}\xleftarrow{R}\{0,1\}^{3\lambda}\right]\le 2^{-2\lambda}$$

just by a union bound over all possible values of s. By taking a union bound over all $i\in\{0,1\}^\lambda$, we get

$$\Pr\left[\exists s,i\in\{0,1\}^\lambda \text{ such that } G(s)\oplus\mathsf{pad}=\mathsf{PDE.Enc}(\mathsf{ek},i):\mathsf{pad}\xleftarrow{R}\{0,1\}^{3\lambda}\right]\le 2^{-\lambda}$$

By the strong correctness of Π_{PDE} and the fact that the message space of Π_{PDE} is $\{0,1\}^\lambda$, the event we bounded is equivalent to the event

$$\mathsf{B}\equiv(\exists s\in\{0,1\}^\lambda \text{ such that } \mathsf{PDE.Dec}(\mathsf{ek},G(s)\oplus\mathsf{pad})\neq\bot).$$

Therefore, we can write $\Pr\left[\mathsf{B}:\mathsf{pad}\xleftarrow{R}\{0,1\}^{3\lambda}\right]\le 2^{-\lambda}$. Recall the definition of g used for the function f_g in SetupBinding: $g(x)=1$ if and only if $x=\mathbf{0}$ or $\mathsf{PDE.Dec}(\mathsf{ek},x\oplus\mathsf{pad})\neq\bot$. Since we assume that $\mathbf{0}$ is not in the image of G, we have $G(s)\neq\mathbf{0}$ for all s, so the event $\neg\mathsf{B}$ implies that for all $s\in\{0,1\}^\lambda$, $g(G(s))=0$. We can bound the probability of A by conditioning on B:

$$\Pr[\mathsf{A}]=\Pr[\mathsf{A}\wedge\mathsf{B}]+\Pr[\mathsf{A}\wedge\neg\mathsf{B}]\le\Pr[\mathsf{B}]+\Pr[\mathsf{A}\mid\neg\mathsf{B}]$$

Now we bound $\Pr[\mathsf{A}\mid\neg\mathsf{B}]$. Assume $\neg\mathsf{B}$ holds. By the resilient function extraction of Π_{FEH}, there do not exist dig, j, π, s such that $\mathsf{FEH.Extract}(\mathsf{td},\mathsf{dig})=1$ and $\mathsf{FEH.Verify}(\mathsf{hk},\mathsf{dig},j,G(s),\pi)=1$ since $\neg\mathsf{B}$ implies $g(G(s))=0$ for all s. Therefore $\Pr[\mathsf{A}\mid\neg\mathsf{B}]=0$, and in total: $\Pr[\mathsf{A}]\le 2^{-\lambda}$ and the claim follows. □

By the constrained correctness of Π_{CPRF} and the choice of the constrain C, the key K' can be evaluated on all dig such that $\mathsf{FEH.Extract}(\mathsf{td},\mathsf{dig})=0$. Claim 5.8 implies that, except with probability $2^{-\lambda}$, for every $\mathsf{dig}, j, \pi, \mathsf{sk}, \mathsf{pk}$ such that $\mathsf{Verify}(\mathsf{hk},\mathsf{dig},j,\mathsf{pk},\pi)=1$ and $G(\mathsf{sk})=\mathsf{pk}$ we have that $\mathsf{FEH.Extract}(\mathsf{td},\mathsf{dig})=0$. Thus, the circuits C_0 and C_1 are functionally equivalent except with probability $\varepsilon(\lambda)\cdot\mathsf{negl}(\lambda)$ and the claim follows. □

Lemma 5.9. *Assume that* Π_{CPRF} *satisfies* ε*-constrained pseudorandomness. Then for any efficient* $\mathcal{A}$ *there exists* $\Lambda\in\mathbb{N}$ *such that for all* $\lambda\ge\Lambda$ *and a negligible function* $\mathsf{negl}(\lambda)$ *it holds that*

$$\left|\Pr[\mathsf{Hyb}_5^0(\mathcal{A})=1]-\Pr[\mathsf{Hyb}_5^1(\mathcal{A})=1]\right|\le\varepsilon(\lambda)\cdot\mathsf{negl}(\lambda)$$

Proof. Let $\mathcal{A}$ be an adversary distinguishing between Hyb_5^0 and Hyb_5^1. The idea is that if the experiment does not abort, then $\mathcal{A}$ can distinguish between a truly random string and the constrained PRF on a fresh value dig^* (which it has not queried before). We construct an adversary $\mathcal{B}$ for the constrained pseudorandomness game of Π_{CPRF} as follows:

1. Algorithm $\mathcal{A}$ sends n to $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ samples ek, pad, and $(\mathsf{hk}, \mathsf{td})$. Next, algorithm $\mathcal{B}$ defines the constraint circuit $C(\mathsf{dig}) = \neg\mathsf{FEH.Extract}(\mathsf{td}, \mathsf{dig})$, and sends the circuit C as well as $1^{\ell_{\mathsf{in}}} = 1^{\ell_{\mathsf{FEH}}}$ and $1^{\ell_{\mathsf{out}}} = 1^{\lambda}$ to the Π_{CPRF} challenger.
3. The Π_{CPRF} challenger responds with the constrained key $K' = \mathsf{CPRF.Constrain}(K, C)$. Algorithm $\mathcal{B}$ samples $P \leftarrow i\mathcal{O}(\mathsf{crsProg}[K', \mathsf{hk}])$ where crsProg is described in Figure 3 and sends $\mathsf{crs} = (\mathsf{hk}, P)$ to $\mathcal{A}$.
4. Algorithm $\mathcal{B}$ answers **honest user registration** and **malicious user registration** exactly as specified in the descriptions of Hyb_5^b. Note that the query-answering logic is identical in both hybrids.
5. Whenever $\mathcal{A}$ makes an **evaluation** query $\mathcal{I}$, algorithm $\mathcal{B}$ computes dig exactly like the challenger does in Hyb_5^b, and sends dig as a query to the Π_{CPRF} challenger, who in turn responds with $\mathsf{sh} = \mathsf{CPRF.Eval}(K, \mathsf{dig})$. Algorithm $\mathcal{B}$ forwards sh to $\mathcal{A}$.
6. When $\mathcal{A}$ makes its challenge query $\mathcal{I}^*$, algorithm $\mathcal{B}$ computes dig^* exactly like the challenger does in Hyb_5^b. If dig^* is equal to some dig_i that was produced by a prior **evaluation** query, then $\mathcal{B}$ aborts the experiment and outputs a random $b' \xleftarrow{R} \{0,1\}$. Otherwise, $\mathcal{B}$ sends dig^* as a challenge. The Π_{CPRF} challenger responds with y_b which $\mathcal{B}$ forwards to $\mathcal{A}$.
7. When $\mathcal{A}$ outputs its guess b', algorithm $\mathcal{B}$ forwards b' as its own output to the Π_{CPRF} challenger.

If $\mathcal{A}$ is efficient then so is $\mathcal{B}$. If $b = 0$, then $y_0 = \mathsf{CPRF.Eval}(K, \mathsf{dig}^*)$ and therefore $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_5^0(\mathcal{A})$. If $b = 1$, then $y_1 \xleftarrow{R} \{0,1\}^{\lambda}$ and therefore $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_5^1(\mathcal{A})$. Define the following event

$$E \equiv \left(\exists(\widetilde{U}_i, \mathsf{dig}_i) \in T_{\mathsf{evals}} \text{ s.t. } \mathsf{dig}^* = \mathsf{dig}_i\right).$$

Note that until $\mathcal{A}$ submits the challenge query, its view in the two experiments is identical, therefore $\Pr[E]$ is a well-define probability that is identical in both experiments. Moreover, given the event E, we know that both experiments output 1 with probability exactly $1/2$, therefore:

$$\Pr[E] \cdot \left|\Pr[\mathsf{Hyb}_5^0(\mathcal{A}) = 1|E] - \Pr[\mathsf{Hyb}_5^1(\mathcal{A}) = 1|E]\right| = \Pr[E] \cdot \left|\frac{1}{2} - \frac{1}{2}\right| = 0.$$

Therefore the advantage of $\mathcal{A}$ in distinguishing the two experiments is at most:

$$\left|\Pr[\mathsf{Hyb}_5^0(\mathcal{A}) = 1|\neg E] - \Pr[\mathsf{Hyb}_5^1(\mathcal{A}) = 1|\neg E]\right|.$$

Similarly, this is exactly the advantage of $\mathcal{B}$ in breaking the constrained pseudorandomness of Π_{CPRF}.

Assuming $\neg E$, the algorithm $\mathcal{B}$ submits a legal challenge query to the Π_{CPRF} challenger. Indeed, a legal challenge requires both that dig^* was not previously queried for evaluation, which is guaranteed by $\neg E$, and that $C(\mathsf{dig}^*) = 0$, i.e., that the constrained key cannot evaluate dig^*. The latter holds because $\mathcal{A}$ is admissible: the challenge set consists exclusively of challenger-generated users, which in Hyb_5^b are all labeled keys, so the padded tuple $\widetilde{U}^*$ satisfies g on every block (labeled keys and the zero padding), giving $\mathsf{FEH.Extract}(\mathsf{td}, \mathsf{dig}^*) = f_g(\widetilde{U}^*) = 1$ and hence $C(\mathsf{dig}^*) = \neg 1 = 0$. Therefore by the ε-constrained pseudorandomness of Π_{CPRF} we conclude that there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that

$$\left|\Pr[\mathsf{Hyb}_5^0(\mathcal{A}) = 1|\neg E] - \Pr[\mathsf{Hyb}_5^1(\mathcal{A}) = 1|\neg E]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

and the claim follows. □

Proof of Theorem 5.2. Let $\mathcal{A}$ be an adversary for the adaptive security with static corruptions game against Construction 5.1. By Lemmas 5.3 to 5.7, there exists a $\Lambda_1 \in \mathbb{N}$ and a negligible function $\mathsf{negl}_1(\lambda)$ such that for all $\lambda \geq \Lambda_1$ and all $b \in \{0,1\}$ it holds that

$$\left|\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5^b(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}_1(\lambda)$$

By Lemma 5.9, there exists a $\Lambda_2 \in \mathbb{N}$ and a negligible function $\mathsf{negl}_2(\lambda)$ such that for all $\lambda \geq \Lambda_2$ it hods that

$$\left|\Pr[\mathsf{Hyb}_5^0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5^1(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}_2(\lambda)$$

In total, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for any $\lambda \geq \Lambda = \max\{\Lambda_1, \Lambda_2\}$, we have

$$\left|\Pr[\mathsf{Real}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}^1(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Therefore Construction 5.1 satisfies ε-adaptive security against static corruptions. □

As a corollary, we get that if all of the underlying ingredients are sub-exponentially secure, then Construction 5.1 is also sub-exponentially secure. Formally:

Corollary 5.10. *Let $\delta \in (0,1)$. Assume that $i\mathcal{O}$ is a $2^{-\lambda^\delta}$-secure indistinguishability obfuscator, G is an $2^{-\lambda^\delta}$-secure PRG, Π_{CPRF} satisfies $2^{-\lambda^\delta}$-constrained pseudorandomness, Π_{PDE} satisfies $2^{-\lambda^\delta}$-ciphertext pseudorandomness, and Π_{FEH} satisfies $2^{-\lambda^\delta}$-collision-resistance, $2^{-\lambda^\delta}$-mode indistinguishability, and resilient function extraction. Then Construction 5.1 satisfies $2^{-\lambda^\delta}$-adaptive security against static corruptions.*

Finally, we can get our main result by combining Corollary 5.10 with Theorem 4.5, as well as the facts that constrained pseudorandom functions can be constructed from one-way functions and indistinguishability obfuscation, function-extractable hash with collision-resistance can be constructed from leveled homomorphic encryption, and pseudorandom deterministic encryption and pseudorandom generators can be constructed from one-way functions:

Corollary 5.11. *Let $\delta \in (0,1)$. Assume there exists a $2^{-\lambda^\delta}$-secure indistinguishability obfuscator, a $2^{-\lambda^\delta}$-secure one-way function, and leveled homomorphic encryption satisfying $2^{-\lambda^\delta}$-IND-CPA security. Then for any $c \in \mathbb{N}$, there exists a succinct NIKE scheme that is adaptively secure against c dynamic corruptions with polynomial hardness, where the CRS, shared key and key pair all have size polynomial in $\lambda, \log n, c$, where n is a bound on the set size.*

Getting rid of the CRS generically. One downside of Construction 5.1 is the fact that it requires a trusted setup to generate the common reference string. Boneh and Zhandry [BZ14] give a generic approach for removing the CRS, but this approach fails in the presence of **malicious user registration** queries. Koppula et al. [KWZ22] give a generic compiler (assuming iO) for lifting adaptive NIKE to the trustless setting. Their compiler proceeds in three steps:

1. The first step uses a non-interactive zero-knowledge proof to achieve what they call "adversarial correctness".
2. The second step applies a Boneh-Zhandry-style approach to remove the CRS at the expense of no longer supporting **malicious user registration** and **evaluation** queries. This step uses iO and preserves adversarial correctness.
3. The final step reintroduces those queries using a clever application of error-correcting codes. This step makes no additional assumptions, but relies on adversarial correctness.

Importantly, and as noted in [KWZ22], this compiler preserves the support for unbounded parties, therefore we can apply it to Construction 5.1 to get rid of the CRS and get a trustless adaptive NIKE for unbounded parties.

Direct construction without a CRS. In addition, our ideas can be used to get a direct construction in the trustless setting. This construction replaces the PRG in Construction 5.1 with *puncturable signatures* [GVW19], and does not need to go through the compiler of [KWZ22]. We give this construction and prove its security in Appendix B.

6 Unbounded Corruptions in the Random Oracle Model

In this section, we show how to upgrade any unbounded NIKE in the random oracle model (ROM). We start by formally defining the syntax and security of NIKE in the ROM. We then show that an adaptively secure unbounded NIKE that only allows **Honest user registration** queries (in addition to the challenge) can be turned into an unbounded NIKE in the ROM with full adaptive security, including an unbounded number of dynamic **Honest user corruption**, **Malicious user registration**, and **Shared key evaluation** queries. This allows us to upgrade either of Construction 5.1 or Construction B.5 to support unbounded dynamic corruptions, with no additional assumptions or sub-exponential security, at the price of working in the ROM.

6.1 Multiparty Non-interactive Key Exchange in the Random Oracle Model

We adapt the NIKE definition from Section 4 to the random oracle model. In this model, all algorithms have access to an oracle $\mathcal{H} : \{0,1\}^{\ell_{\text{in}}(\lambda)} \to \{0,1\}^{\ell_{\text{out}}(\lambda)}$. We will consider NIKE without a trusted setup, and additionally, we will not bound the number of dynamic corruptions.

Definition 6.1 (Unbounded Non-interactive Key Exchange in the ROM)**.** An unbounded non-interactive key exchange scheme (NIKE) in the random oracle model is a tuple of polynomial time oracle algorithms $\Pi_{\mathsf{NIKE}} = (\mathsf{Publish}^{\mathcal{H}}, \mathsf{KeyGen}^{\mathcal{H}})$ with the following syntax:

- $\mathsf{Publish}^{\mathcal{H}}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$: On input a security parameter, the key publishing algorithm outputs a key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{KeyGen}^{\mathcal{H}}(\mathsf{sk}, U) \to \mathsf{sh}$: On input a secret key sk and a set (without repetitions) of public keys U of size n, the key generation algorithm outputs a shared key sh. This algorithm is deterministic relative to $\mathcal{H}$ and can run in time $\mathsf{poly}(\lambda, n)$.

We now define two notions of correctness: the first, which we call adversarial correctness, states that for two honest users, the probability that an adversary manages to maliciously build a set which contains both honest users and makes the honest users derive different keys, is negligible. Intuitively, for any two honest users that participate in the same set, they will derive the same shared key with overwhelming probability, even if the other participants in the set are malicious. This is stronger than traditional (imperfect) correctness which just guarantees functionality if all users are honest. Koppula et al. [KWZ22] initially defined and used this property purely for their NIKE transformations, but we view it as a natural security property which guarantees a functional system even if an adversary plants a corrupted user in a critical set. We point out that we require adversarial correctness to hold even against *unbounded* adversaries. The second notion is perfect correctness, which is essentially adversarial correctness where the adversary never succeeds i.e., the honest players always agree.

Definition 6.2 (Adversarial Correctness)**.** A multiparty NIKE scheme $\Pi_{\mathsf{NIKE}} = (\mathsf{Publish}^{\mathcal{H}}, \mathsf{KeyGen}^{\mathcal{H}})$ satisfies *adversarial correctness* if for all $\lambda \in \mathbb{N}$ and all (computationally unbounded) adversaries $\mathcal{A}$, there exists a negligible function negl such that:

$$\Pr\left[\begin{array}{c|l} \mathsf{KeyGen}^{\mathcal{H}}(\mathsf{sk}_0, U) \neq \mathsf{KeyGen}^{\mathcal{H}}(\mathsf{sk}_1, U) & (\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Publish}^{\mathcal{H}}(1^\lambda) \\ \wedge & (\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{Publish}^{\mathcal{H}}(1^\lambda) \\ \{\mathsf{pk}_0, \mathsf{pk}_1\} \subseteq U & U \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}_0, \mathsf{pk}_1) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of the algorithms and the choice of the random oracle $\mathcal{H}$. We say that Π_{NIKE} satisfies *perfect correctness* if $\mathsf{negl}(\lambda) = 0$.

Next, we define two notions of security: full adaptive security, and basic adaptive security. In both notions, the adversary is allowed to adaptively pick its challenge; the difference is that in the first notion, we allow the adversary to make an unbounded number of other queries such as dynamic corruptions, malicious user registration and shared key evaluation. Whereas in the basic notion, we only allow the adversary to register honest users.

Definition 6.3 (Full Adaptive Security in the ROM)**.** Let $\Pi_{\mathsf{NIKE}} = (\mathsf{Publish}^{\mathcal{H}}, \mathsf{KeyGen}^{\mathcal{H}})$ be a succinct NIKE scheme in the random oracle model, where the oracle has inputs of length $\ell_{\mathsf{in}}(\lambda)$ and outputs of length $\ell_{\mathsf{out}}(\lambda)$. For any adversary $\mathcal{A}$, we define the following security experiment parameterized by a bit $b \in \{0,1\}$:

1. On input 1^λ, the challenger initializes two empty tables T_{users} and T_{evals}, and a counter $i \leftarrow 0$. Additionally, a uniformly random function $\mathcal{H} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$ is sampled.

2. $\mathcal{A}$ can make the following queries:

   - **Random oracle evaluation:** The adversary sends a string $x \in \{0,1\}^{\ell_{\mathsf{in}}}$. The challenger sends $\mathcal{H}(x)$ to $\mathcal{A}$.
   - **Honest user registration:** The challenger increments $i \leftarrow i+1$, runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Publish}^{\mathcal{H}}(1^\lambda)$, adds the entry $(i, \mathsf{pk}, \mathsf{sk}, \mathsf{uncorrupted})$ to the table T_{users}, and sends pk to $\mathcal{A}$.
   - **Malicious user registration:** The adversary sends a public key pk. If there is *no* entry in T_{users} with the same public key, the challenger increments $i \leftarrow i+1$, and adds the entry $(i, \mathsf{pk}, \perp, \mathsf{corrupted})$ to the table T_{users}. Otherwise, we say that the adversary is not admissible, and the experiment outputs a random bit $b' \xleftarrow{R} \{0,1\}$.
   - **Shared key evaluation:** The adversary sends a set of indices $\mathcal{I} \subseteq [2^\lambda]$ without repetitions and an index $j^* \in \mathcal{I}$. For each $j \in \mathcal{I}$, the adversary looks for an entry $(j, \mathsf{pk}_j, \perp, \cdot)$ in T_{users}. For j^*, the challenger looks for an entry with a secret key $(j^*, \mathsf{pk}_{j^*}, \mathsf{sk}_{j^*}, \cdot)$. If any entry is not found or $\mathsf{sk}_{j^*} = \perp$, then we say that the adversary is not admissible, and the experiment outputs a random bit $b' \xleftarrow{R} \{0,1\}$. The challenger then computes the set $U = \{\mathsf{pk}_j\}_{j \in \mathcal{I}}$ and responds with $\mathsf{KeyGen}^{\mathcal{H}}(\mathsf{sk}_{j^*}, U)$. Additionally, the challenger adds $\mathcal{I}$ to the table T_{evals}.
   - **Honest user corruption:** The adversary sends an index i'. The challenger looks in T_{users} for an entry $(i', \mathsf{pk}, \mathsf{sk}, \mathsf{uncorrupted})$, updates the entry to $(i', \mathsf{pk}, \mathsf{sk}, \mathsf{corrupted})$ and sends sk to $\mathcal{A}$. If the entry is not found, then we say that the adversary is not admissible, and the experiment outputs a random bit $b' \xleftarrow{R} \{0,1\}$.

3. At any point, the adversary sends the challenge set (without repetitions) $\mathcal{I}^*$, such that for each $j \in \mathcal{I}^*$ there exists an entry $(j, \mathsf{pk}_j, \mathsf{sk}_j, \mathsf{uncorrupted})$ in T_{users}. The challenger then computes the set $U^* = \{\mathsf{pk}_j\}_{j \in \mathcal{I}^*}$ and sends y_b, where $y_0 = \mathsf{KeyGen}^{\mathcal{H}}(\mathsf{sk}_1, U^*)$ (using the secret key sk_1 of the first user in $\mathcal{I}^*$) and y_1 is uniform over the range of $\mathsf{KeyGen}^{\mathcal{H}}$. If $\mathcal{I}^*$ is in the table T_{evals}, or there exists $j \in \mathcal{I}^*$ such that $(j, \mathsf{pk}_j, \mathsf{sk}_j, \mathsf{uncorrupted})$ is not in T_{users}, then we say that the adversary is not admissible, and the experiment outputs a random bit $b' \xleftarrow{R} \{0,1\}$.

4. The adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

We say that Π_{NIKE} satisfies full adaptive security if for every efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

Definition 6.4 (Basic Adaptive Security in the ROM)**.** Let $\Pi_{\mathsf{NIKE}} = (\mathsf{Publish}^{\mathcal{H}}, \mathsf{KeyGen}^{\mathcal{H}})$ be an unbounded NIKE scheme. The basic adaptive security game is identical to the full adaptive security game, except that the only **Honest user registration** queries are allowed. The setup and challenge phases remain the same. We say that Π_{NIKE} satisfies adaptive security with static corruptions if for every efficient adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

6.2 Achieving Full Adaptive Security

In this section, we show how to upgrade a NIKE in the plain model with basic adaptive security, into a fully adaptive NIKE in the random oracle model. Specifically, the upgraded NIKE supports unbounded dynamic corruptions. We start by presenting our construction.

Construction 6.5. Let $\Pi_{\mathsf{NIKE}} = (\mathsf{Publish}, \mathsf{KeyGen})$ be a NIKE scheme in the plain model. Assume that public keys for Π_{NIKE} have length ℓ. The random oracle shrinks its input by a factor of ℓ. Specifically, for any security parameter λ, the random oracle consists of a sequence of functions $\mathcal{H}_n : \{0,1\}^{n\cdot\lambda\cdot\ell} \to \{0,1\}^{n\cdot\lambda}$. When the length of the input x is known, we simply write $\mathcal{H}(x)$. We construct a NIKE scheme $\Pi'_{\mathsf{NIKE}} = (\mathsf{Publish}^{\mathcal{H}}, \mathsf{KeyGen}^{\mathcal{H}})$ in the random oracle model as follows.

- $\mathsf{Publish}^{\mathcal{H}}(1^\lambda)$:
  1. Sample a random selection string $s \xleftarrow{R} \{0,1\}^\lambda$.
  2. For each $i \in [\lambda]$ and $b \in \{0,1\}$, generate a key pair:
  $$(\mathsf{pk}_{i,b}, \mathsf{sk}_{i,b}) \leftarrow \mathsf{Publish}(1^\lambda).$$
  3. Construct the public key pk and secret key sk as:
  $$\mathsf{pk} = \left(\mathsf{pk}_{i,b}\right)_{i\in[\lambda],b\in\{0,1\}}, \quad \mathsf{sk} = \left(s, \left(\mathsf{sk}_{i,s_i}\right)_{i\in[\lambda]}\right).$$
- $\mathsf{KeyGen}^{\mathcal{H}}(\mathsf{sk}, U)$:
  1. Parse the set of public keys $U = \{\mathsf{pk}^{(1)}, \ldots, \mathsf{pk}^{(n)}\}$. Let $\mathsf{pk}^{(j)}$ be the public key in U corresponding to the secret key sk.
  2. Query the random oracle to obtain the selector string $R = \mathcal{H}(U)$, where $R \in \{0,1\}^{n\cdot\lambda}$. Parse R as $R^{(1)} \| \ldots \| R^{(n)}$, where each $R^{(k)} = R_1^{(k)} \cdots R_\lambda^{(k)} \in \{0,1\}^\lambda$.
  3. Parse $\mathsf{sk} = \left(s, (\mathsf{sk}_{i,s_i})_{i\in[\lambda]}\right)$, and find the minimum $i' \in [\lambda]$ such that $s_{i'} = 1$ and $R_{i'}^{(j)} = 1$. If no such index i' exists, output $\perp$.
  4. For each $k \in [n]$, parse $\mathsf{pk}^{(k)} = \left(\mathsf{pk}_{i,b}^{(k)}\right)_{i\in[\lambda],b\in\{0,1\}}$ and prepare the set $V^{(k)} = \left\{\mathsf{pk}_{i,R_i^{(k)}}^{(k)}\right\}_{i\in[\lambda]}$. Let $V = \bigcup_{k=1}^n V^{(k)}$.
  5. Output $\mathsf{KeyGen}(\mathsf{sk}_{i',s_{i'}}, V)$.

Theorem 6.6 (Adversarial Correctness)**.** *Assume Π_{NIKE} is an unbounded NIKE scheme that satisfies perfect correctness. Then Construction 6.5 is an unbounded NIKE scheme Π'_{NIKE} in the ROM that satisfies adversarial correctness.*

Proof. By the perfect correctness of the underlying NIKE, the only way to get a correctness error in Construction 6.5 is if the random oracles outputs a selection string that prevents one of the secret keys from evaluating the shared key, which happens if the random oracle's selection string completely misses the secret key's selection string. In the adversarial correctness experiment, the adversary does not get to learn anything about the secret keys, and specifically, the selection strings. Therefore even with unbounded computational power and unbounded queries to the random oracle, only has a negligible probability of finding a set of public keys U such that $\mathcal{H}(U)$ causes any of the secret keys to fail, because the secret key's selection string are uniformly random and hidden from the adversary. □

Theorem 6.7 (Full Adaptive Security)**.** *Assume Π_{NIKE} is an unbounded NIKE scheme in the plain model that satisfies basic adaptive security and perfect correctness. Then Construction 6.5 is an unbounded NIKE scheme Π'_{NIKE} in the ROM that satisfies full adaptive security.*

We prove the full adaptive security of Π'_{NIKE} via a hybrid argument. Eventually, we reduce to the basic adaptive security of Π_{NIKE}, but first, we have to modify the experiment to something that the reduction can simulate. Let $\mathcal{A}$ be an adversary against the full adaptive security of Π'_{NIKE}, and let Q be (an upper bound on) the number of random oracle queries that $\mathcal{A}$ makes. We assume without loss of generality that for any set U that $\mathcal{A}$ queries to the random oracle, all users in U have been registered. We also assume that $\mathcal{A}$ queries the random oracle on the challenge set U^* before it makes the challenge query.

We define a sequence of hybrids.

- Hyb_0^b: This is the real adaptive security experiment with parameter b.
- Hyb_1^b: This is the same as Hyb_0, except that the experiment aborts if the random oracle outputs a string that does not match the selection strings of the corresponding secret keys. Namely:
  - Whenever the adversary makes a **Random oracle query** on a set $U = (\mathsf{pk}^{(1)}, \ldots, \mathsf{pk}^{(n)})$, let $R = R^{(1)} \| \cdots \| R^{(n)}$ be the answer of the random oracle. The challenger checks that for every challenger-generated public key $\mathsf{pk}^{(i)} \in U$, the corresponding secret key can evaluate the shared key for U. That is, the challenger checks that $s^{(i)} \oplus R^{(i)} \neq \mathbf{1}$, where $\mathsf{sk}^{(i)} = \left(s^{(i)}, (\mathsf{sk}_{j,s_j})_{j\in[\lambda]}\right)$ is the secret key corresponding to $\mathsf{pk}^{(i)}$. If the check fails, the experiment aborts.
- Hyb_2^b: This is the same as Hyb_1, except that the challenger always responds to **Shared key evaluation** queries using the lexicographically first inner secret key $\mathsf{sk}_{1,b}$ indicated by the random oracle answer, regardless of the selection string of the secret key. Namely:
  - Whenever the adversary makes a **Register honest user** query, the challenger keeps all secret keys $\mathsf{sk}_{j,b}^{(i)}$ for all $(j, b) \in [\lambda] \times \{0, 1\}$ (instead of discarding the ones that do not correspond to the selection string).
  - Whenever the adversary makes a **Shared key evaluation** on a set $\mathcal{I}$ and an index $j^* \in \mathcal{I}$, the challenger answer as follows:
    1. Builds the set $U = \left\{\mathsf{pk}^{(i)}\right\}_{i\in\mathcal{I}}$ and evaluates $R = \mathcal{H}(U)$. Let t be the index of $\mathsf{pk}^{(j^*)}$ in the ordered set U.
    2. For each $k \in \mathcal{I}$, parses $\mathsf{pk}^{(k)} = \left(\mathsf{pk}_{i,b}^{(k)}\right)_{i\in[\lambda],b\in\{0,1\}}$ and prepares the set $V^{(k)} = \left\{\mathsf{pk}_{i,R_i^{(k)}}^{(k)}\right\}_{i\in[\lambda]}$. Let $V = \bigcup_{k=1}^{n} V^{(k)}$.
    3. Finds the secret key $\mathsf{sk}_{1,R_1^{(t)}}^{(j^*)}$ and responds with $\mathsf{KeyGen}\left(\mathsf{sk}_{1,R_1^{(t)}}^{(j^*)}, V\right)$.
- Hyb_3^b: This is the same as Hyb_2, except that the challenger guesses the index $q \in [Q_{\mathsf{ro}}]$ of the random oracle query that corresponds to the challenge set U^*. Specifically, the challenger picks $q \xleftarrow{R} [Q_{\mathsf{ro}}]$ at the beginning. If the adversary submits the challenge set U^* and it was not the qth unique query made to the random oracle, the experiment aborts and outputs a random bit.
- Hyb_4^b: This is the same as Hyb_3, except that the challenger responds to the qth **Random oracle evaluation** query using the selection strings of the users in the set. That is, let $U^* = \{\mathsf{pk}^{(k_j)}\}_{j\in[n]}$ be the qth **Random oracle evaluation** query. Then the challenger finds the corresponding secret keys $\mathsf{sk}^{(k_j)} = \left(s^{(k_j)}, \left(\mathsf{sk}_{i,s_i^{(k_j)}}\right)_{i\in[\lambda]}\right)$ and responds with $R^* = s^{(k_1)} \| \cdots \| s^{(k_n)}$.
- Hyb_5^b: This is the same as Hyb_4^b, except that the challenger responds to **Shared key evaluation** queries as in the real game.

Proposition 6.8. *For any $b \in \{0, 1\}$ and any adversary $\mathcal{A}$ that makes a polynomial a number of queries to the random oracle, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\Pr[\mathsf{Hyb}_0^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^b(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. For each secret key $\mathsf{sk} = \left(s, \left(\mathsf{sk}_{i,s_i}\right)_{i\in[\lambda]}\right)$, there exists a single selection string which does not allow it to evaluate a shared key, and that string is the complement of s. The probability that a random string hits the bad selection string is $2^{-\lambda}$. Since the adversary makes a polynomial number of **Random oracle queries**, and each one gives it a chance of hitting a polynomial number of bad strings, then the total probability that the adversary hits a bad string is negligible by a union bound. The two experiments are identical otherwise, therefore the claim follows. □

Proposition 6.9. *If Π_{NIKE} satisfies perfect correctness, then for any $b \in \{0,1\}$ and any $\mathcal{A}$ we have*

$$\Pr[\mathsf{Hyb}_1^b(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_2^b(\mathcal{A}) = 1].$$

Proof. Conditioned on the experiment not aborting, the **Shared key evaluation** queries in both experiments always succeed. By the perfect correctness of the underlying NIKE, whenever two honest secret keys evaluate the same set which contains both their corresponding public keys, the KeyGen algorithm returns the same key 0 regardless of how the other keys in the set were computed. Therefore the view of the adversary in both experiments is identical and the claim follows. □

Proposition 6.10. *For any adversary $\mathcal{A}$ that makes a Q_{ro} queries to the random oracle, we have*

$$\left|\Pr[\mathsf{Hyb}_3^0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^1(\mathcal{A}) = 1]\right| = \frac{1}{Q_{\mathrm{ro}}}\left|\Pr[\mathsf{Hyb}_2^0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^1(\mathcal{A}) = 1]\right|$$

Proof. For any $b \in \{0,1\}$, the experiment Hyb_3^b conditioned on the challenger guessing the index of the **Random oracle evaluation** query that corresponds to the challenge is identical to Hyb_2^b, and the guess is right with probability $1/Q_{\mathrm{ro}}$. Therefore if we denote the challenger's guess by q, then we have:

$$\Pr[\mathsf{Hyb}_3^b(\mathcal{A}) = 1] = \frac{1}{Q_{\mathrm{ro}}}\Pr[\mathsf{Hyb}_2^b(\mathcal{A}) = 1] + \frac{Q_{\mathrm{ro}} - 1}{Q_{\mathrm{ro}}}\Pr[\mathsf{Hyb}_3^b(\mathcal{A}) = 1 | q \neq Q_{\mathrm{ro}}].$$

On the other hand, the experiments Hyb_3^0 and Hyb_3^1 just output 1 with probability $1/2$ if the guess is wrong, which means that

$$\Pr[\mathsf{Hyb}_3^0(\mathcal{A}) = 1 | q \neq Q_{\mathrm{ro}}] = \Pr[\mathsf{Hyb}_3^1(\mathcal{A}) = 1 | q \neq Q_{\mathrm{ro}}] = \frac{1}{2}.$$

Therefore in total:

$$\left|\Pr[\mathsf{Hyb}_3^0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^1(\mathcal{A}) = 1]\right| = \frac{1}{Q_{\mathrm{ro}}}\left|\Pr[\mathsf{Hyb}_2^0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^1(\mathcal{A}) = 1]\right|$$

and the claim follows. □

Proposition 6.11. *For any $b \in \{0,1\}$ and any adversary $\mathcal{A}$, we have*

$$\Pr[\mathsf{Hyb}_3^b(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_4^b(\mathcal{A}) = 1].$$

Proof. Observe that the abort conditions for both experiments are the same, so in those cases they behave identically. Conditioned on the experiments not aborting, we know that the qth **Random oracle evaluation** query corresponds to the challenge set U^*. For an admissible adversary, the users in U^* are never corrupted, therefore the adversary does not directly see their secret keys and specifically does not learn the selection strings for each user in that set. Moreover, the challenger answers **Shared key evaluation** in a way that does not depend on the selection string of any secret key, since it always answers with $\mathsf{sk}_{1,0}^{(j^*)}$ or $\mathsf{sk}_{1,1}^{(j^*)}$ depending on the random oracle answer (this is the change introduced in Hyb_2^b), Therefore, the view of the adversary does not depend on the selection strings of the users in U^*. Since those strings are uniformly sampled, then replacing the random oracle value on U^* with the selection strings preserves the same distribution, and the claim follows. □

Proposition 6.12. *If Π_{NIKE} satisfies perfect correctness, then for any $b \in \{0,1\}$ and any $\mathcal{A}$ we have*

$$\Pr[\mathsf{Hyb}_4^b(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_5^b(\mathcal{A}) = 1].$$

The proof of Proposition 6.12 is completely analogous to that of Proposition 6.9.

Lemma 6.13. *If Π_{NIKE} satisfies basic adaptive security, then for any $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\left|\Pr[\mathsf{Hyb}_5^0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5^1(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. We construct a reduction $\mathcal{B}$ that breaks the basic adaptive security of Π_{NIKE} by simulating Hyb_5^b for $\mathcal{A}$. The underlying NIKE experiment will only support **Honest user registration** and **Challenge** queries. **Malicious user registration**, **Shared key evaluation**, as well as **Honest user corruption** and **Random oracle evaluation** queries will be handled by the reduction. Algorithm $\mathcal{B}$ interacts with a basic adaptive security challenger for Π_{NIKE} and operates as follows:

1. **Setup:** $\mathcal{B}$ initializes the experiment for $\mathcal{A}$. It chooses a random index $q \xleftarrow{R} [Q_{\mathsf{ro}}]$ as the guess for the challenge RO query. It also initializes an empty table T_{ro} for the random oracle and a counter $c_{\mathsf{ro}} \leftarrow 0$.

2. **Honest user registration:** When $\mathcal{A}$ requests to register a new honest user j, $\mathcal{B}$ performs the following:

   - Sample a random selector string $R^{(j)} \xleftarrow{R} \{0,1\}^\lambda$.
   - For each $i \in [\lambda]$:
     - For the bit $b = R_i^{(j)}$, $\mathcal{B}$ requests a new **honest public key** from the underlying static challenger. Let this key be $\mathsf{pk}_{i,R_i^{(j)}}$.
     - For the bit $b = 1 - R_i^{(j)}$, $\mathcal{B}$ runs the underlying Publish algorithm to generate $(\mathsf{pk}_{i,1-R_i^{(j)}}, \mathsf{sk}_{i,1-R_i^{(j)}})$.
   - $\mathcal{B}$ constructs the public key $\mathsf{pk}^{(j)} = \left(\mathsf{pk}_{i,b}\right)_{i\in[\lambda],b\in\{0,1\}}$ and sends it to $\mathcal{A}$.
   - $\mathcal{B}$ stores $(j, R^{(j)}, \{\mathsf{sk}_{i,1-R_i^{(j)}}\}_i)$.

3. **Malicious user registration:** Whenever $\mathcal{A}$ sends a public key pk for registration, algorithm $\mathcal{B}$ parses $\mathsf{pk} = \left(\mathsf{pk}_{i,b}\right)_{i\in[\lambda],b\in\{0,1\}}$ and registers every $\mathsf{pk}_{i,b}$ locally, *without* sending them to the underlying NIKE challenger.

4. **Random oracle queries:** When $\mathcal{A}$ makes a query U to the random oracle, algorithm $\mathcal{B}$ first checks whether there is an entry (U, R) in the table T_{ro} and uses it as the answer. If not, then:

   - $\mathcal{B}$ increments $c_{\mathsf{ro}} \leftarrow c_{\mathsf{ro}} + 1$.
   - If $c_{\mathsf{ro}} = q$ (the guessed challenge query): $\mathcal{B}$ retrieves the stored strings $R^{(j)}$ for each user $j \in U$. It constructs the response $R = R^{(1)} \| \cdots \| R^{(\lambda)}$ and returns R to $\mathcal{A}$.
   - If $c_{\mathsf{ro}} \neq q$: $\mathcal{B}$ samples a random string $R \xleftarrow{R} \{0,1\}^{|U|\cdot\lambda}$ and returns it.
   - In any case, algorithm $\mathcal{B}$ adds the entry (U, R) to the table T_{ro}.

5. **Honest user corruption:** When $\mathcal{A}$ requests to corrupt user j:

   - $\mathcal{B}$ defines the secret selection string $s^{(j)}$ to be the bitwise negation of the selector string: $s^{(j)} = \mathbf{1} \oplus R^{(j)}$.
   - $\mathcal{B}$ constructs the secret key $\mathsf{sk}^{(j)} = (s^{(j)}, \{\mathsf{sk}_{i,s_i^{(j)}}\}_i)$. Note that for every i, the key $\mathsf{sk}_{i,s_i^{(j)}}$ corresponds to the bit $1 - R_i^{(j)}$, which is the key generated by $\mathcal{B}$ (registered as malicious). Thus, $\mathcal{B}$ knows all the necessary secret keys to answer this query.
   - $\mathcal{B}$ sends $\mathsf{sk}^{(j)}$ to $\mathcal{A}$.

6. **Shared key evaluation:** When $\mathcal{A}$ makes a shared key evaluation query for a user j^* and a set U, we assume that U has been queried on the random oracle already.

   - Algorithm $\mathcal{B}$ looks up the entry (U, R) in the random oracle table T_{ro}, and parses R to find the segment $R_{\mathsf{ro}}^{(j^*)} \in \{0,1\}^\lambda$ corresponding to user j^*.
   - Algorithm $\mathcal{B}$ checks if there exists an index i' such that $R_{\mathsf{ro},i'}^{(j^*)} = 1$ and $\mathcal{B}$ holds the secret key for this index (i.e., $1 - R_{i'}^{(j^*)} = 1$). If no such index exists, algorithm $\mathcal{B}$ returns $\perp$.
   - Otherwise, algorithm $\mathcal{B}$ uses the known secret key $\mathsf{sk}_{j^*,1}$ at index i' to run the underlying KeyGen and returns the result to $\mathcal{A}$.

7. **Challenge:** When $\mathcal{A}$ sends the challenge set $\mathcal{I}$, algorithm $\mathcal{B}$ builds the corresponding set of public keys U^*, and checks whether U^* was the qth **Random oracle query**. If not, algorithm $\mathcal{B}$ aborts and outputs a random guess. Otherwise:
   - Algorithm $\mathcal{B}$ retrieves the entry (U^*, R) from T_{ro}.
   - By construction, for every user $j \in U^*$, the relevant public keys in the set V (as defined in $\mathsf{KeyGen}^{\mathcal{H}}$) are exactly $\left\{\mathsf{pk}_{i,R_i^{(j)}}\right\}$, which are the honest keys from the underlying challenger.
   - $\mathcal{B}$ collects all such keys and submits them as the challenge set to the underlying static challenger.
   - The underlying challenger returns a shared key K_b. $\mathcal{B}$ forwards K_b to $\mathcal{A}$.
8. **Output:** When $\mathcal{A}$ outputs a guess b', algorithm $\mathcal{B}$ forwards b' as its own output.

Analysis: When $\mathcal{B}$ guesses q wrong, then it outputs a random bit just like the experiment does in Hyb_5^b. We want to prove that $\mathcal{B}$ perfectly simulates Hyb_5^b when the guess q is correct. Algorithm $\mathcal{B}$ simulates the responses to all queries exactly like the challenger in Hyb_5:

- **Random oracle evaluation:** By construction, $\mathcal{B}$ answers all queries with uniform strings, except for the qth query which it answers using the selection string of the users in the set. This is identical to how the challenger answers **Random oracle evaluation** queries in Hyb_5^b.
- **Honest user registration:** Algorithm $\mathcal{B}$ samples half of the public keys honestly itself, and the other half comes from the underlying NIKE challenger, therefore all public keys are sampled honestly.
- **Honest user corruption:** Whenever an honest user j was registered, it is assigned a uniform string $R^{(j)}$, and the secret keys consists of $\mathbf{1} \oplus R^{(j)}$ and the corresponding secrets, which is exactly how a secret key of an honest user is distributed.
- **Shared key evaluation:** In Hyb_5 (and in fact, in the real security game), a **Shared key evaluation** query could fail if the random oracle returns a selection string for which the evaluating user has no secret key that allows it to evaluate the underlying shared key. Algorithm $\mathcal{B}$ simulates that by selecting the public keys for which each user has the corresponding secret keys uniformly, and then refusing to answer any query if the user (i.e., the reduction itself) does not have any of the sufficient secret keys. When algorithm $\mathcal{B}$ actually has the key, it answers the query as expected.
- **Challenge:** By construction, the set V that algorithm $\mathcal{B}$ builds from the challenge set U^* consists of honest keys for the underlying NIKE scheme. Algorithm $\mathcal{B}$ then sends those users to its challenger and gets a string y_b which is then forwarded to $\mathcal{A}$. If y_b is the shared key of the set V, then $\mathcal{B}$ has simulated Hyb_5^0. If y_b is a truly random string, then $\mathcal{B}$ has simulated Hyb_5^1.

In total, we proved that $\mathcal{B}$ perfectly simulates Hyb_5^b (depending on how its challenger responds to the challenge set). In conclusion,

$$\left|\Pr[\mathsf{Hyb}_5^0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5^1(\mathcal{A}) = 1]\right|$$

is exactly the advantage that $\mathcal{B}$ has in breaking the underlying NIKE scheme, therefore by the basic adaptive security of the Π_{NIKE}, this quantity is negligible and the claim follows. □

Acknowledgments

Shafik Nassar would like to thank George Lu for useful discussions. Brent Waters is supported by NSF CNS-2318701 and a Simons Investigator award.

References

[ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. *Cryptology ePrint Archive*, 2013.

[ABG+25] Joël Alwen, Chris Brzuska, Jérôme Govinden, Patrick Harasser, and Stefano Tessaro. Succinct pprfs via memory-tight reductions. In *Annual International Cryptology Conference*, pages 628–662. Springer, 2025.

[BBK+23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In *CRYPTO*, pages 252–283, 2023.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *Theory of cryptography conference*, pages 52–73. Springer, 2014.

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

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2), 2012.

[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.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual cryptology conference*, pages 868–886. Springer, 2012.

[BS02] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Cryptology ePrint Archive*, 2002.

[BSW16] Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 792–821. Springer, 2016.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on computing*, 43(2):831–871, 2014.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology-ASIACRYPT 2013: 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II 19*, pages 280–300. Springer, 2013.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.

[CHL+15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–12. Springer, 2015.

[CLLT16] Jean-Sébastien Coron, Moon Sung Lee, Tancrede Lepoint, and Mehdi Tibouchi. Cryptanalysis of ggh15 multilinear maps. In *Annual international cryptology conference*, pages 607–628. Springer, 2016.

[CLT13] Jean-Sébastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *Annual Cryptology Conference*, pages 476–493. Springer, 2013.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In *Theory of cryptography conference*, pages 468–497. Springer, 2015.

[DH76] WHITFIELD DIFFIE and MARTIN E HELLMAN. New directions in cryptography. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 22(6), 1976.

[DKN+20] Alex Davidson, Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Adaptively secure constrained pseudorandom functions in the standard model. In *Annual International Cryptology Conference*, pages 559–589. Springer, 2020.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered abe, flexible broadcast, and more. In *CRYPTO*, 2023.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, USA, 2009.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 1–17. Springer, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *Theory of Cryptography Conference*, pages 498–527. Springer, 2015.

[GGHW17] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. *Algorithmica*, 79(4):1353–1373, 2017.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 365–377. ACM, 1982.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Annual cryptology conference*, pages 75–92. Springer, 2013.

[GVW19] Rishab Goyal, Satyanarayana Vusirikala, and Brent Waters. Collusion resistant broadcast and trace from positional witness encryption. In *IACR international workshop on public key cryptography*, pages 3–33. Springer, 2019.

[GW09] Craig Gentry and Brent Waters. Adaptive security in broadcast encryption systems (with short ciphertexts). In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 171–188. Springer, 2009.

[HJ16] Yupu Hu and Huiwen Jia. Cryptanalysis of ggh map. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 537–565. Springer, 2016.

[HJK+16] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal samplers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 715–744. Springer, 2016.

[HW15] Pavel Hubácek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *ITCS*, 2015.

[JJ22] Abhishek Jain and Zhengzhong Jin. Indistinguishability obfuscation via mathematical proofs of equivalence. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1023–1034. IEEE, 2022.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *STOC*, 2021.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb {F}_p$, dlin, and prgs in nc^0. In *EUROCRYPT*, 2022.

[Jou00] Antoine Joux. A one round protocol for tripartite diffie–hellman. In *International algorithmic number theory symposium*, pages 385–393. Springer, 2000.

[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.

[KPW17] Venkata Koppula, Andrew Poelstra, and Brent Waters. Universal samplers with fast verification. In *IACR International Workshop on Public Key Cryptography*, pages 525–554. Springer, 2017.

[KRS15] Dakshita Khurana, Vanishree Rao, and Amit Sahai. Multi-party key exchange for unbounded parties from indistinguishability obfuscation. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 52–75. Springer, 2015.

[KWZ22] Venkata Koppula, Brent Waters, and Mark Zhandry. Adaptive multiparty nike. In *Theory of Cryptography Conference*, pages 244–273. Springer, 2022.

[Mer89] Ralph C. Merkle. A certified digital signature. In *CRYPTO*, 1989.

[NWW24] Shafik Nassar, Brent Waters, and David J Wu. Monotone policy bargs from bargs and additively homomorphic encryption. In *Theory of Cryptography Conference*, pages 399–430. Springer, 2024.

[NWW25] Shafik Nassar, Brent Waters, and David J Wu. Monotone-policy bargs and more from bargs and quadratic residuosity. In *IACR International Conference on Public-Key Cryptography*, pages 283–313. Springer, 2025.

[Rao14] Vanishree Rao. Adaptive multiparty non-interactive key exchange without setup in the standard model. *Cryptology ePrint Archive*, 2014.

[RVV25] Seyoon Ragavan, Neekon Vafa, and Vinod Vaikuntanathan. Indistinguishability obfuscation from bilinear maps and lpn variants. In *Theory of Cryptography Conference*, pages 3–36. Springer, 2025.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, 2014.

A Adaptive Corruptions from Static Corruptions

The goal of this section is to prove Theorem 4.5 from Section 4. Namely, we prove that if a NIKE scheme Π_{NIKE} is sub-exponentially adaptively secure against static corruptions, then it is also adaptively secure against adaptive corruptions.

We assume the adversaries are non-uniform algorithms. Specifically in the NIKE security game, the non-uniform advice contains the number of corruptions c.

Proof of Theorem 4.5. Fix a NIKE scheme $\Pi_{\mathsf{NIKE}} = (\mathsf{Setup}, \mathsf{Publish}, \mathsf{KeyGen})$ with security only against static corruptions. Throughout this section, we will denote the security parameter used for Π_{NIKE} by $\lambda_{\mathsf{static}}$, and assume that Π_{NIKE} is $2^{-\lambda_{\mathsf{static}}^{\delta}}$-adaptively secure against static corruptions. Define the following NIKE scheme $\Pi'_{\mathsf{NIKE}} = (\mathsf{Setup}', \mathsf{Publish}', \mathsf{KeyGen}')$ as follows:

- The setup algorithm $\mathsf{Setup}'(1^\lambda, 1^c, n)$ runs the underlying NIKE with security parameter $\lambda_{\mathsf{static}} := \lambda_{\mathsf{static}}(\lambda, c) = (c\lambda)^{1/\delta}$ and outputs the CRS. That is, outputs $\mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda_{\mathsf{static}}}, n)$.
- The publishing and key generation algorithm remain unchanged. That is: $\mathsf{Publish}' \equiv \mathsf{Publish}$ and $\mathsf{KeyGen}' \equiv \mathsf{KeyGen}$.

Let $\mathcal{A}$ be an adversary for the adaptive security game with adaptive corruptions against Π'_{NIKE}. Assume that $\mathcal{A}$ makes exactly q **honest user registration** queries. In addition, assume that $\mathcal{A}$ makes **honest user corruption** queries only to users that were generated using **honest user registration** queries. We describe the following sequence of hybrids. For simplicity, since we know q, we assign two different counters in this experiment: one for **honest user registration** queries, and another for **malicious user registration** queries.

- Real^b: this is the real adaptive security game with adaptive corruptions.
  1. On input 1^λ, adversary $\mathcal{A}$ sends $1^c, n$ to the challenger. The challenger initializes two empty tables T_{users} and T_{evals} and two counters $i^{(h)} \leftarrow 0$ and $i^{(c)} \leftarrow q$, and sends $\mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda_{\mathsf{static}}}, 1^c, n)$ to $\mathcal{A}$.
  2. $\mathcal{A}$ can make the following queries:
     - **Honest user registration:** The challenger increments $i^{(h)} \leftarrow i^{(h)}+1$, runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Publish}(1^{\lambda_{\mathsf{static}}})$, adds the entry $(i^{(h)}, \mathsf{pk}, \mathsf{sk}, \mathsf{uncorrupted})$ to the table T_{users}, and sends pk to $\mathcal{A}$.
     - **Malicious user registration:** The adversary sends a public key pk. The challenger increments $i^{(c)} \leftarrow i^{(c)} + 1$, and adds the entry $(i^{(c)}, \mathsf{pk}, \perp, \mathsf{corrupted})$ to the table T_{users}.
     - **Evaluation:** The adversary sends a set of indices $\mathcal{I} \subseteq [2^\lambda]$ and an index $j^* \in \mathcal{I}$. For each $j \in \mathcal{I}$, the adversary looks for an entry $(j, \mathsf{pk}_j, \perp, \cdot)$ in T_{users}. For j^*, the challenger looks for an entry with a secret key $(j^*, \mathsf{pk}_{j^*}, \mathsf{sk}_{j^*}, \cdot)$ If any entry is not found, then the experiment aborts. The challenger then computes the set $U = \{\mathsf{pk}_j\}_{j \in \mathcal{I}}$ and responds with $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{sk}_{j^*}, U)$. Additionally, the challenger adds $\mathcal{I}$ to the table T_{evals}.
     - **Honest user corruption:** The adversary sends an index i'. The challenger looks in T for an entry $(i', \mathsf{pk}, \mathsf{sk}, \cdot)$. If it is not found, the experiment aborts (with output 0). Else, the challenger updates the entry to $(i', \mathsf{pk}, \mathsf{sk}, \mathsf{corrupted})$ and sends sk to $\mathcal{A}$.

     Assume that $\mathcal{A}$ makes exactly c **honest user corruption** queries (we can always have it makes the same query multiple times to get exactly c).
  3. At any point, the adversary sends the challenge set $\mathcal{I}^*$. If $\mathcal{I}^*$ is in the table T_{evals}, then the experiment aborts. The challenger checks that for each $j \in \mathcal{I}$ there exists an entry $(j, \mathsf{pk}_j, \mathsf{sk}_j, \mathsf{uncorrupted})$ in T, otherwise the experiment aborts. The challenger then computes the set $U^* = \{\mathsf{pk}_j\}_{j \in \mathcal{I}^*}$ and sends y_b, where $y_0 = \mathsf{KeyGen}(\mathsf{crs}, \mathsf{sk}_1, U^*)$ and y_1 is uniform over the range of KeyGen.
  4. The adversary outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.
- Hyb^b: same as Real^b, except that the challenger guesses the corruption queries in advance, and aborts if the guess is wrong. That is, the challenger changes its behavior as follows:
  1. During the setup phase, after the adversary sends 1^c, the challenger samples $i_j \xleftarrow{R} [q]$ for each $j \in [c]$.
  2. During the query phase, let i'_j be the jth **honest user corruption** that $\mathcal{A}$ makes. If $i'_j \neq i_j$, then the experiment aborts.
  3. The experiment is otherwise not changed.

Claim A.1. *For any $\mathcal{A}$ that makes at most q **honest user registration** queries, we have*

$$\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] = q^c \cdot \Pr[\mathsf{Hyb}^b(\mathcal{A}) = 1]$$

Proof. For any $b \in \{0,1\}$, the difference between the two experiments is that the challenger samples the indices $i_1, \ldots, i_c$ uniformly at random and aborts if there exists $j \in [c]$ such that $i'_j \neq i_j$, where $i'_1, \ldots, i'_c$ are the corruption queries made by $\mathcal{A}$. Importantly, the view of $\mathcal{A}$ does not depend on the indices sampled by the challenger, therefore the probability that the guesses are all correct is exactly q^c, and is completely independent of the bit b' output by $\mathcal{A}$, therefore we get $\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] = q^c \cdot \Pr[\mathsf{Hyb}^b(\mathcal{A}) = 1]$ and the claim follows. □

Lemma A.2. *Assume that Π_{NIKE} is $2^{-\lambda^{\delta}_{\mathsf{static}}}$-adaptively secure against static corruptions, and suppose that $\lambda_{\mathsf{static}} = (c\lambda)^{1/\delta}$. Then for any efficient adversary $\mathcal{A}$ there exists Λ such that for all $\lambda > \Lambda$, the following holds*

$$\left|\Pr[\mathsf{Hyb}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^1(\mathcal{A}) = 1]\right| \leq 2^{-c\lambda} \cdot \mathsf{negl}(c\lambda).$$

Proof. Let $\mathcal{A}$ be a non-uniform attacker for security game with adaptive corruptions, which makes q **honest registration queries**. We construct a non-uniform attacker $\mathcal{B}$ for the game with static corruptions. Specifically, for each security parameter $\lambda_{\mathsf{static}}$, the algorithm $\mathcal{B}$ will get as non-uniform advice the security parameter $\lambda = \frac{1}{c} \cdot \lambda^{\delta}_{\mathsf{static}}$, on which it should run the algorithm $\mathcal{A}$.

1. On input a security parameter $1^{\lambda_{\mathsf{static}}}$, algorithm $\mathcal{B}$ gets λ and runs $\mathcal{A}$ on that security parameter, and gets the parameters $1^c, n$.
2. Algorithm $\mathcal{B}$ forwards n to its challenger to get crs, which $\mathcal{B}$ then forwards back to $\mathcal{A}$.
3. Algorithm $\mathcal{B}$ sample the indices $i_1, \ldots, i_c \xleftarrow{R} [q]$ and initializes $i^{(h)} \leftarrow 0$ and $k \leftarrow 0$.
4. In the query phase, algorithm $\mathcal{B}$ answers the queried made by $\mathcal{A}$ as follows:
   - **Honest user registration:** Algorithm $\mathcal{B}$ increments $i^{(h)} \leftarrow i^{(h)} + 1$. If $i^{(h)} \neq i_j$ for all $j \in [c]$, then $\mathcal{B}$ makes an **honest user registration** with its own challenger in order to get a public key $\mathsf{pk}_{i^{(h)}}$. Otherwise, $\mathcal{B}$ samples a key pair $(\mathsf{pk}_{i^{(h)}}, \mathsf{sk}_{i^{(h)}}) \leftarrow \mathsf{Publish}(1^{\lambda_{\mathsf{static}}})$ and saves $(i^{(h)}, \mathsf{pk}_{i^{(h)}}, \mathsf{sk}_{i^{(h)}})$. In any case, algorithm $\mathcal{B}$ sends pk_i to $\mathcal{A}$.
   - **Malicious user registration** and **evaluation** queries are forwarded by $\mathcal{B}$ as is to its challenger. Algorithm $\mathcal{B}$ then forwards the response back to $\mathcal{A}$ as is.
   - **Honest user corruption:** Algorithm $\mathcal{A}$ sends an index i'. Algorithm $\mathcal{B}$ increments $k \leftarrow k + 1$. If $i' \neq i_k$, algorithm $\mathcal{B}$ aborts the experiment. Otherwise, algorithm $\mathcal{B}$ looks for the entry $(i', \mathsf{pk}_{i'}, \mathsf{sk}_{i'})$ and responds with $\mathsf{sk}_{i'}$
5. For the challenger phase, algorithm $\mathcal{B}$ simply forwards the challenge query made by $\mathcal{A}$ to its own challenger, and then forwards the response y_b of the challenger back to $\mathcal{A}$.
6. Algorithm $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Suppose there exists an infinite sequence of security parameters λ and a polynomial p for which

$$\left|\Pr[\mathsf{Hyb}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^1(\mathcal{A}) = 1]\right| > 2^{-c\lambda} \cdot p(c\lambda)$$

We show that $\lambda_{\mathsf{static}} = (c\lambda)^{1/\delta}$ for each such λ satisfies is a security parameter on which $\mathcal{B}$ breaks the adaptive security game with static corruptions against Π_{NIKE}. Namely, that there exists a polynomial p' such that for each such $\lambda_{\mathsf{static}}$, we have $\mathsf{Adv}_{\mathcal{B}}(\lambda_{\mathsf{static}}) > 2^{-\lambda^{\delta}_{\mathsf{static}}} \cdot p'(\lambda_{\mathsf{static}})$.

Define the polynomial $p'(\lambda_{\mathsf{static}}) = p\left(\lambda^{\delta}_{\mathsf{static}}\right)$. Fix any $\lambda_{\mathsf{static}} = (c\lambda)^{1/\delta}$. Note that for such $\lambda_{\mathsf{static}}$, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_b(\mathcal{A})$, since $\mathcal{B}$ only aborts if it cannot answer an **honest use corruption** query, but the specification of Hyb_b and the fact that $\mathcal{B}$ samples the indices $i_1, \ldots, i_c$ randomly ensures that the experiment Hyb_b would have aborted in that case anyway. Moreover, if $\mathcal{A}$ submits a valid challenge then specifically it only contain

uncorrupted honest users from $\mathcal{A}$'s view, which map to honest users in the static corruptions game that $\mathcal{B}$ is playing, therefore the challenge that $\mathcal{B}$ forwards is also legal, therefore the advantage of $\mathcal{B}$ in that case is exactly

$$\mathsf{Adv}_{\mathcal{B}}(\lambda_{\mathsf{static}}) = \left|\Pr[\mathsf{Hyb}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^1(\mathcal{A}) = 1]\right| > 2^{-c\lambda} \cdot p(c\lambda) = 2^{-\lambda_{\mathsf{static}}^{\delta}} \cdot p'(\lambda_{\mathsf{static}})$$

thus breaking the $2^{-\lambda_{\mathsf{static}}^{\delta}}$-adaptively secure against static corruptions of Π_{NIKE}. □

We can now complete the proof of Theorem 4.5, which boils down to proving

$$|\Pr[\mathsf{Real}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}_1(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

By Lemma A.2, we have $\left|\Pr[\mathsf{Hyb}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^1(\mathcal{A}) = 1]\right| \leq 2^{-c\lambda} \cdot \mathsf{negl}(c\lambda)$. Combining that with Claim A.1, we get

$$\left|\Pr[\mathsf{Real}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}^1(\mathcal{A}) = 1]\right| \leq q^c \cdot 2^{-c\lambda} \cdot \mathsf{negl}(c\lambda).$$

Since $\mathcal{A}$ is efficient, then q and c are polynomials in λ. Specifically, we have $q^c = O(2^{-c\lambda})$ and therefore

$$\left|\Pr[\mathsf{Real}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}^1(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$$

and the theorem follows.

B Constructing Adaptive NIKE Without a Trusted Setup

In this section, we give a construction of adaptively secure NIKE against static corruptions, without a common reference string. Looking ahead, the plan is to adapt our main construction to the trustless setting while solving the problems that were pointed out by Boneh and Zhandry [BZ14].

The starting point is Construction 5.1. Imagine now that each user publishes their own evaluation program, similar to the CRS program from Figure 3, and a hash key as part of their public key. When a set of users wants to evaluate a shared key, they deterministically pick a *canonical user* from the set and use that user's evaluation program to generate the shared key.

As noted by Boneh and Zhandry [BZ14], this approach fails in the presence of corrupted users because a corrupted program can behave arbitrarily. Specifically, it can just output the secret key that was fed into it. In general, it seems like we cannot afford to feed an entire honest secret key into an adversarial program.

Signatures to the rescue. Boneh and Zhandry solve this problem using signatures. The public key now includes a verification key vk of a signature scheme, and the secret is the corresponding signing key sk. The evaluation program now expects to get a signature on some specific string, instead of the entire secret key. The point being that this signature would be useless for the adversary in other contexts.

More specifically, imagine we want to generate a shared key for a set of users U. Let $\widetilde{\mathsf{pk}} = (\widetilde{\mathsf{hk}}, \widetilde{P}, \widetilde{\mathsf{vk}})$ be the public key of the canonical user and sk and vk be the signing key and verification key of the evaluating user. The evaluating user hashes the set U using the canonical hash key $\widetilde{\mathsf{hk}}$ and signs the *canonical public key* using its own sk. The evaluating user then feeds its public key, the digest and the opening (to its key), and the signature to the canonical program $\widetilde{P}$. The canonical program verifies the opening using its own hash key $\widetilde{\mathsf{hk}}$, verifies the signature using the evaluating user's verification key vk, and if the checks pass, computes the shared key as the evaluation of a constrained PRF on the digest.

Intuitively, the signatures that are fed into any canonical program $\widetilde{P}$ can only be used to generate shared keys for sets that include that same canonical user. Therefore if the canonical program was generated by the adversary, even if we give those signatures to the adversary in the clear, it still should not be able to use them in other contexts (unless it breaks the signature scheme).

To formally prove security, we need to use puncturable signatures. First, we guess the canonical user in the **challenge query**, incurring only a polynomial loss. Second, we puncture the verification keys of all challenger-generated users on the public key of the guessed canonical user. Next, we make the function-extractable hash key of

the canonical user extractable on whether the hashed set exclusively consists of challenger-generated keys. Finally, we constrain the PRF of the canonical user on digests that only include challenger-generated keys. As we can see, the strategy follows the one that we used in Section 5. However, we still face two problems:

- In Section 5, the challenger-generated public keys were labeled by replacing the PRG output with a pseudo-random deterministic encryption. This mechanism allowed the hash function to decide whether the hashed string included a challenger-generated key or not using the PDE decryption key. Since the public key in this construction does not involve a PRG or a uniform string component, we solve this problem by artificially adding a random string r to the public key.
- The public key includes the evaluation program, therefore we cannot simply pass an entire public key into another evaluation program, since the size of the program does not allow it. On one hand, this means that we cannot hash the obfuscated programs when computing the digests. On the other hand, if the digest does not depend on the entirety of each public key, then we leave the construction vulnerable to mix-and-match attacks. Namely, the adversary may register a corrupted user whose public key matches an honest user only on the part that is used in the hash, and use it to evaluate the challenge shared key. Moreover, the evaluation program cannot verify the signature that the evaluating user produces, because the message includes the evaluation program itself! We solve this problem by signing the entire public key using a one-time signature[11], and outputting the verification key and the signature as part of the public key. The shared key generation algorithm is modified to abort if any public key has an invalid one-time signature. When computing the digest, it suffices to hash the puncturable verification key and the random string components. Moreover, it suffices to only sign the one-time verification key of the canonical user.

This concludes our overview of the trustless NIKE construction. The rest of this section is devoted to formally presenting the construction and proving its security.

B.1 Additional Preliminaries: Signature Schemes

We now recall the signature scheme definitions that we need for our construction.

Definition B.1 (One-Time Digital Signatures)**.** A one-time digital signature scheme for arbitrary-length messages is a tuple of efficient algorithms $\Pi_{\mathsf{OTS}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$: On input the security parameter λ, the generation algorithm outputs a verification key vk and a signing key sk.
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$: On input a signing key sk and a message $m \in \{0,1\}^*$, the signing algorithm outputs a signature σ.
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to b$: On input a verification key vk, a message $m \in \{0,1\}^*$, and a signature σ, the verification algorithm outputs a bit $b \in \{0,1\}$.

The signature scheme should satisfy the following properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$ and all $m \in \{0,1\}^*$, it holds that

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1 \;:\; \begin{matrix} (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m) \end{matrix} \right] = 1.$$

- **Selective strong one-time unforgeability:** For any adversary $\mathcal{A}$, the selective one-time unforgeability game $\mathsf{ExptSOTU}_{\mathcal{A}}(\lambda)$ is defined as follows:
  1. On input a security parameter λ, the adversary $\mathcal{A}$ sends a message $m \in \{0,1\}^*$ to the challenger.

[11] In fact, we only need to sign the components of the public key that are not fed into the evaluation program. As we will see later, the random component r needs to be passed into the evaluation program for our proof strategy to work, so we do not need to sign it using the one-time signature.

2. The challenger samples a key pair $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and a signature $\mathsf{sk} \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$, and sends vk, σ to $\mathcal{A}$.
3. The adversary outputs m^*, σ^*.
4. The output of the experiment is 1 if $(m^*, \sigma^*) \neq (m, \sigma)$ and $\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1$.

We say that Π_{OTS} satisfies ε-selective one-time unforgeability if for any efficient $\mathcal{A}$, there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$, the following holds $\Pr[\mathsf{ExptSOTU}_{\mathcal{A}}(\lambda) = 1] \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$.

Fact B.2. A (one-time) signature scheme can be constructed from one-way functions.

Next, we recall the definition of puncturable signatures, first introduced by [GVW19] (under the name all-but-one signatures), which are a weaker form of constrained signatures that were introduced by [BZ14]. The following definition is copied from [NWW25].

Definition B.3 (Puncturable Signature [GVW19])**.** A puncturable (or all-but-one) signature scheme with messages of length $\ell := \ell(\lambda)$ is a tuple of efficient algorithms $\Pi_{\mathsf{PS}} = (\mathsf{Gen}, \mathsf{GenPunc}, \mathsf{Sign}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$: On input the security parameter λ, the key-generation algorithm outputs a key pair $(\mathsf{vk}, \mathsf{sk})$.
- $\mathsf{GenPunc}(1^\lambda, m^*) \to (\mathsf{vk}, \mathsf{sk})$: On input a security parameter λ and a message $m^* \in \{0,1\}^\ell$, the punctured key generation algorithm outputs a key pair $(\mathsf{vk}, \mathsf{sk})$.
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$: On input a signing key sk and a message $m \in \{0,1\}^\ell$, the signing algorithm outputs a signature σ.
- $\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to b$: On input a verification key vk, a message $m \in \{0,1\}^\ell$, and a signature σ, the verification algorithm outputs a bit $b \in \{0,1\}$.

The puncturable signature scheme should satisfy the following properties:

- **Correctness:** For all $\lambda \in \mathbb{N}$ and all $m \in \{0,1\}^\ell$, it holds that
$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1 \;:\; \begin{matrix}(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)\end{matrix} \right] = 1.$$
- **Perfect unforgeability in punctured mode:** For all $\lambda \in \mathbb{N}$, all $m^* \in \{0,1\}^\ell$, and all $\sigma^* \in \{0,1\}^*$, it holds that
$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1 \;:\; (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{GenPunc}(1^\lambda, m^*) \right] = 0.$$
- **Verification key indistinguishability:** For any adversary $\mathcal{A}$ and any $b \in \{0,1\}$, we define the verification key indistinguishability experiment $\mathsf{ExptVKI}_{\mathcal{A}}(\lambda, b)$ as follows:
  1. On input a security parameter λ, the adversary $\mathcal{A}$ sends a message $m^* \in \{0,1\}^\ell$ to the challenger.
  2. The challenger samples key pairs $(\mathsf{vk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Gen}(1^\lambda)$ and $(\mathsf{vk}_1, \mathsf{sk}_1) \leftarrow \mathsf{GenPunc}(1^\lambda, m^*)$ and gives vk_b to the adversary.
  3. Next, the adversary can make signing queries on messages $m \in \{0,1\}^\ell \setminus \{m^*\}$. On each signing query, the challenger replies with $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_b, m)$.
  4. The adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{PS} satisfies ε-verification key indistinguishability if for any efficient adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that
$$\left|\Pr[\mathsf{ExptVKI}_{\mathcal{A}}(\lambda, 0) = 1] - \Pr[\mathsf{ExptVKI}_{\mathcal{A}}(\lambda, 1) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda).$$

Boneh and Zhandry [BZ14] construct constrained signatures from sub-exponentially secure iO and OWFs. However, for the specific case of puncturing constraints, the Sahai-Waters signature scheme [SW14] implicitly gives puncturable signatures from polynomially secure iO and OWFs.

Fact B.4 ([SW14, BZ14])**.** Puncturable signatures can be constructed from indistinguishability obfuscation and one-way functions.

B.2 Trustless NIKE Construction

Construction B.5 (NIKE construction)**.** Our construction uses the following ingredients:

- An indistinguishability obfuscator iO for the class of circuits that implement the algorithm in Figure 4 as per Definition 2.1.
- A one-time signature scheme $\Pi_{\mathsf{OTS}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ as per Definition B.1. Let $\ell_{\mathsf{vk}} := \ell_{\mathsf{vk}}(\lambda)$ be the length of the verification key on security parameter λ.
- A puncturable signature scheme $\Pi_{\mathsf{PS}} = (\mathsf{Gen}, \mathsf{GenPunc}, \mathsf{Sign}, \mathsf{Verify})$ for messages of length $\ell_{\mathsf{vk}} := \ell_{\mathsf{vk}}(\lambda)$ as per Definition B.3. Let $\ell'_{\mathsf{vk}} := \ell'_{\mathsf{vk}}(\lambda)$ be the length of the verification key and $\ell_{\mathsf{GP}} := \ell_{\mathsf{GP}}(\lambda)$ be the amount of randomness that PS.GenPunc uses on security parameter λ and message length $\ell_{\mathsf{vk}}(\lambda)$.
- A pseudorandom deterministic encryption scheme $\Pi_{\mathsf{PDE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as per Definition 2.6 with ciphertexts of length ℓ_{ct}. Assume that the decryption algorithm PDE.Dec can be implemented by a circuit of size $s(\lambda)$ and depth $d(\lambda)$. This ingredient only appears in the security analysis, not in the construction itself.
- A function-extractable hash $\Pi_{\mathsf{FEH}} = (\mathsf{Setup}, \mathsf{SetupBinding}, \mathsf{Hash}, \mathsf{Verify}, \mathsf{Extract})$ as per Definition 3.1 over the domain $\mathcal{X}_\lambda = \{0,1\}^{\ell'_{\mathsf{vk}}+\ell_{\mathsf{ct}}}$. We assume that Π_{FEH} supports conjunction of block functions of size $s(\lambda)$ and depth $d(\lambda)$. Denote the digest length of Π_{FEH} by $\ell_{\mathsf{FEH}} := \ell_{\mathsf{FEH}}(\lambda, n)$. And let $C_\lambda = \left\{\mathsf{FEH.Extract}(\mathsf{td}, \cdot) : \{0,1\}^{\ell_{\mathsf{FEH}}} \to \{0,1\}\right\}_{\mathsf{td}}$ be the class of circuits that implements the extraction algorithm.
- A constrained PRF scheme $\Pi_{\mathsf{CPRF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Constrain})$ as per Definition 2.3 for a class of circuits C_λ that implements the extraction algorithm of the function-extractable hash.

Throughout this construction, we assume that any set U is represented by the *lexicographically ordered* tuple $(\mathsf{pk}_1, \ldots, \mathsf{pk}_t)$, where $U = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_t\}$. The algorithms are defined as follows:

- $\mathsf{Publish}(1^\lambda, n)$:
  1. Samples a random string $r \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{ct}}}$.
  2. Samples a PRF key $K \leftarrow \mathsf{CPRF.Setup}(1^\lambda, 1^{\ell_{\mathsf{FEH}}}, 1^\lambda)$.
  3. Samples a FEH key $\mathsf{hk} \leftarrow \mathsf{FEH.Setup}(1^\lambda, n)$.
  4. Samples a one-time signature key pair $(\mathsf{vk}^{\mathsf{OTS}}, \mathsf{sk}^{\mathsf{OTS}}) \leftarrow \mathsf{OTS.Gen}(1^\lambda)$.
  5. Samples a puncturable signature key pair $(\mathsf{vk}^{\mathsf{PS}}, \mathsf{sk}^{\mathsf{PS}}) \leftarrow \mathsf{PS.Gen}(1^\lambda)$.
  6. Samples $P \leftarrow iO(\mathsf{pkProg}[K, \mathsf{hk}, \mathsf{vk}^{\mathsf{OTS}}])$ where pkProg is described in Figure 4.
  7. Sets $m = (\mathsf{hk}, \mathsf{vk}^{\mathsf{PS}}, P)$ and samples a one-time signature $\sigma^{\mathsf{OTS}} \leftarrow \mathsf{OTS.Sign}(\mathsf{sk}^{\mathsf{OTS}}, m)$.
  8. Outputs $\mathsf{pk} = (r, m, \mathsf{vk}^{\mathsf{OTS}}, \sigma^{\mathsf{OTS}})$ and $\mathsf{sk} = \mathsf{sk}^{\mathsf{PS}}$.

Constants: A constrained PRF key K (which could be a master key or a constrained key), a FEH hash key hk, a verification key $\mathsf{vk}^{\mathsf{OTS}}$.
Inputs: A digest dig, an index $j^* \in [n]$, a block v_{j^*}, an opening π_{j^*}, and a signature σ_{j^*}.

1. If $\mathsf{FEH.Verify}(\mathsf{hk}, \mathsf{dig}, j^*, v_{j^*}, \pi_{j^*}) = 0$, aborts.
2. Parses $v_{j^*} = \left(r_{j^*}, \mathsf{vk}^{\mathsf{PS}}_{j^*}\right)$
3. If $\mathsf{PS.Verify}\left(\mathsf{vk}^{\mathsf{PS}}_{j^*}, \mathsf{vk}^{\mathsf{OTS}}, \sigma_{j^*}\right) = 0$, aborts.
4. Outputs $\mathsf{CPRF.Eval}(K, \mathsf{dig})$.

Figure 4: Program $\mathsf{pkProg}[K, \mathsf{hk}, \mathsf{vk}^{\mathsf{OTS}}]$.

- $\mathsf{KeyGen}(\mathsf{sk}^{\mathsf{PS}}, U)$:
  1. Parses $U = (\mathsf{pk}_1, \ldots, \mathsf{pk}_t)$ and $\mathsf{pk}_j = \left(r_j, m_j, \mathsf{vk}_j^{\mathsf{OTS}}, \sigma_j^{\mathsf{OTS}}\right)$ for all $j \in [t]$, where $t = |U|$.
  2. If there exists $j \in [t]$ such that $\mathsf{OTS.Verify}\left(\mathsf{vk}_j^{\mathsf{OTS}}, m_j, \sigma_j^{\mathsf{OTS}}\right) = 0$, aborts with $\perp$.
  3. Parses the evaluation parameters of the canonical user $m_1 = (\mathsf{hk}, \mathsf{vk}^{\mathsf{PS}}, P)$.
  4. Let $\mathsf{pk}_{j^*} = \left(r_{j^*}, m_{j^*}, \mathsf{vk}_{j^*}^{\mathsf{OTS}}, \sigma_{j^*}^{\mathsf{OTS}}\right)$ be the public key that matches $\mathsf{sk}^{\mathsf{PS}}$, and abort if no such j^* exists.[12]
  5. Let $v_j = (r_j, \mathsf{vk}_j^{\mathsf{PS}})$ for each $j \in [t]$, and denote $V = (v_1, \ldots, v_t)$.
  6. Pads the string $\widetilde{V} = V \circ \mathbf{0}^{n-t}$, where $\mathbf{0} = 0^{\ell_{\mathsf{ct}} + \ell'_{\mathsf{vk}}}$.
  7. Hashes $(\mathsf{dig}, \pi_1, \ldots, \pi_n) = \mathsf{FEH.Hash}(\mathsf{hk}, \widetilde{V})$.
  8. Signs the one-time verification key of the canonical user $\sigma_{j^*} = \mathsf{PS.Sign}(\mathsf{sk}^{\mathsf{PS}}, \mathsf{vk}_1^{\mathsf{OTS}})$.
  9. Outputs $P(\mathsf{dig}, j^*, v_{j^*}, \pi_{j^*}, \sigma_{j^*})$.

Perfect correctness. Observe that for any honestly generated key pair $(\mathsf{pk}, \mathsf{sk})$ that is output by Publish, the check $\mathsf{OTS.Verify}(\mathsf{vk}^{\mathsf{OTS}}, m, \sigma^{\mathsf{OTS}}) = 1$ (performed in Step 2) follows by the correctness of Π_{OTS}. In addition, the program $\mathsf{pkProg}[K, \mathsf{hk}, \mathsf{vk}^{\mathsf{OTS}}]$ from Figure 4 outputs the same string sh on any dig, as long as it is given a valid FEH opening π_j to some key v_j in position j, as well as a valid signature σ_j on $\mathsf{vk}^{\mathsf{OTS}}$ under the key $\mathsf{vk}_j^{\mathsf{PS}}$. Therefore for any set U (of honestly generated keys) and any two parties $j^*, i^* \in U$ with the corresponding secret keys $\mathsf{sk}_{j^*}^{\mathsf{PS}}, \mathsf{sk}_{i^*}^{\mathsf{PS}}$, by the perfect correctness of Π_{FEH} and Π_{PS}, and the functionality preserving of $i\mathcal{O}$, we have $\mathsf{KeyGen}(\mathsf{sk}_{j^*}^{\mathsf{PS}}, U) = \mathsf{KeyGen}(\mathsf{sk}_{i^*}^{\mathsf{PS}}, U)$ and perfect completeness of Π_{NIKE} follows.

Succinctness. Key pair succinctness follows from the efficiency of the underlying primitives. The secret key $\mathsf{sk} = \mathsf{sk}^{\mathsf{PS}}$ is succinct, with size $\mathsf{poly}(\lambda, \ell_{\mathsf{vk}}(\lambda)) = \mathsf{poly}(\lambda)$ by the properties of Π_{PS} (as its message length ℓ_{vk} is $\mathsf{poly}(\lambda)$). The public key $\mathsf{pk} = (r, m, \mathsf{vk}^{\mathsf{OTS}}, \sigma^{\mathsf{OTS}})$ where $m = (\mathsf{hk}, \mathsf{vk}^{\mathsf{PS}}, P)$ consists of:

- The random string r has size ℓ_{ct}, which is polynomial in λ.
- The one-time verification key and signature $\mathsf{vk}^{\mathsf{OTS}}$ and σ^{OTS} are all $\mathsf{poly}(\lambda)$ by the efficiency of Π_{OTS} (which supports arbitrary-length messages).
- The Π_{PS} verification key $\mathsf{vk}^{\mathsf{PS}}$, which is $\mathsf{poly}(\lambda, \ell_{\mathsf{vk}}(\lambda)) = \mathsf{poly}(\lambda)$ by the properties of Π_{PS}.
- The FEH key hk, which has length $\mathsf{poly}(\lambda, \log n)$ by the succinctness of Π_{FEH}.
- The program P, which is an obfuscation of pkProg from Figure 4.

The program pkProg has the following complexity:

1. The first step of verifying the opening of FEH can be implemented in time $\mathsf{poly}(\lambda, \log n)$ by the succinctness of Π_{FEH}.
2. The second step is parsing.
3. The third step of verifying the Π_{PS} signature can be implemented in time $\mathsf{poly}(\lambda, \ell_{\mathsf{vk}}(\lambda)) = \mathsf{poly}(\lambda)$.
4. The final step can be computed in time $\mathsf{poly}(\lambda, \ell_{\mathsf{FEH}})$ which is $\mathsf{poly}(\lambda, \log n)$ by the succinctness of Π_{FEH}. Note that this step might use the constrained key of Π_{CPRF}, but this key has size $\mathsf{poly}(\lambda, \ell_{\mathsf{FEH}})$ which is $\mathsf{poly}(\lambda, \log n)$.

In total, pkProg can be implemented by a circuit of size $\mathsf{poly}(\lambda, \log n)$. Since this is the class of circuits that $i\mathcal{O}$ needs to support, the obfuscated program P has size $\mathsf{poly}(\lambda, \log n)$. In total, the public key pk is also succinct. Shared key succinctness also follows immediately by the functionality preserving of $i\mathcal{O}$. Namely, the shared key is computed by an obfuscation of the program $\mathsf{pkProg}[K, \mathsf{hk}, \mathsf{vk}^{\mathsf{OTS}}]$ from Figure 4, which just outputs the evaluation of the CPRF, and the output length of the CPRF is set to be λ.

[12] For example, we can sign some message using $\mathsf{sk}^{\mathsf{PS}}$ and find $\mathsf{vk}_{j^*}^{\mathsf{PS}}$ that verifies the signature, or we can assume that $\mathsf{sk}^{\mathsf{PS}}$ includes the randomness that was used to generate the key pair or even that the index/public key is passed as an additional input to KeyGen.

B.3 Security Analysis

In this section, we prove that Construction B.5 is adaptively secure against static corruptions. Recall that by Theorem 4.5, this suffices to get an adaptive NIKE with bounded dynamic corruptions. We prove the following theorem:

Theorem B.6. *Assume that $i\mathcal{O}$ is an ε-secure indistinguishability obfuscator, Π_{OTS} is an ε-secure one-time signature scheme, Π_{PS} satisfies ε-verification key indistinguishability, Π_{CPRF} satisfies ε-constrained pseudorandomness, Π_{PDE} satisfies ε-ciphertext pseudorandomness, and Π_{FEH} satisfies ε-collision-resistance, ε-mode indistinguishability, and resilient function extraction. Then Construction B.5 satisfies ε-adaptive security against static corruptions.*

Let $\mathcal{A}$ be an adversary for the adaptive security with static corruptions game. Throughout this section, we assume that $\mathcal{A}$ is admissible (as per Definition 4.2) and makes at most $q = q(\lambda)$ **honest user registration** queries. We define the following sequence of hybrids starting with the real security game Real^b.

- Real^b: This is the real security game with parameter b.

  1. **Setup Phase:**
     (a) On input 1^λ, the adversary $\mathcal{A}$ sends n to the challenger.
     (b) The challenger initializes two empty tables $T_{\mathsf{users}} \leftarrow \emptyset$ and $T_{\mathsf{evals}} \leftarrow \emptyset$, and a counter $i \leftarrow 0$.
  2. **Query Phase:** $\mathcal{A}$ can adaptively make the following queries.
     - **Honest user registration:** The challenger does the following
       (a) Increments the user index $i \leftarrow i + 1$.
       (b) Samples $r_i \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{ct}}}$.
       (c) Samples $K_i \leftarrow \mathsf{CPRF.Setup}(1^\lambda, 1^{\ell_{\mathsf{FEH}}}, 1^\lambda)$.
       (d) Samples $\mathsf{hk}_i \leftarrow \mathsf{FEH.Setup}(1^\lambda, n)$.
       (e) Samples $(\mathsf{vk}_i^{\mathsf{OTS}}, \mathsf{sk}_i^{\mathsf{OTS}}) \leftarrow \mathsf{OTS.Gen}(1^\lambda)$.
       (f) Samples $(\mathsf{vk}_i^{\mathsf{PS}}, \mathsf{sk}_i^{\mathsf{PS}}) \leftarrow \mathsf{PS.Gen}(1^\lambda)$.
       (g) Samples $P_i \leftarrow i\mathcal{O}(\mathsf{pkProg}[K_i, \mathsf{hk}_i, \mathsf{vk}_i^{\mathsf{OTS}}])$ where pkProg is described in Figure 4.
       (h) Sets $m_i = (\mathsf{hk}_i, \mathsf{vk}_i^{\mathsf{PS}}, P_i)$ and samples $\sigma_i^{\mathsf{OTS}} \leftarrow \mathsf{OTS.Sign}(\mathsf{sk}_i^{\mathsf{OTS}}, m_i)$.
       (i) Sets $\mathsf{pk}_i = (r_i, m_i, \mathsf{vk}_i^{\mathsf{OTS}}, \sigma_i^{\mathsf{OTS}})$ and $\mathsf{sk}_i = \mathsf{sk}_i^{\mathsf{PS}}$.
       (j) Adds the entry $(i, \mathsf{pk}_i, \mathsf{sk}_i, \mathsf{uncorrupted}, K_i, \mathsf{hk}_i, \mathsf{vk}_i^{\mathsf{OTS}}, \mathsf{sk}_i^{\mathsf{OTS}})$ to T_{users}.
       (k) Sends pk_i to $\mathcal{A}$.
     - **Malicious user registration:**
       (a) The adversary sends a public key pk.
       (b) The challenger increments $i \leftarrow i + 1$.
       (c) The challenger adds the entry $(i, \mathsf{pk}, \perp, \mathsf{corrupted})$ to T_{users}.
     - **Evaluation:**
       (a) The adversary sends a set of indices $\mathcal{I}$ and a specific index $j^* \in \mathcal{I}$.
       (b) The challenger retrieves entries for all $j \in \mathcal{I}$ from T_{users} and sets $U = (\mathsf{pk}_j)_{j \in \mathcal{I}}$.
       (c) The challenger parses $\mathsf{pk}_j = \left(r_j, m_j, \mathsf{vk}_j^{\mathsf{OTS}}, \sigma_j^{\mathsf{OTS}}\right)$ for all $j \in \mathcal{I}$.
       (d) By the admissibility of $\mathcal{A}$, we have $\mathsf{OTS.Verify}\left(\mathsf{vk}_j^{\mathsf{OTS}}, m_j, \sigma_j^{\mathsf{OTS}}\right) = 1$ for all $j \in \mathcal{I}$.
       (e) Let idx be the index of the canonical user in the set U. The challenger retrieves K_{idx} and $\mathsf{hk}_{\mathsf{idx}}$ from T_{users}.
       (f) The challenger forms the tuple $V = (v_j)_{j \in \mathcal{I}}$ where $v_j = (r_j, \mathsf{vk}_j^{\mathsf{PS}})$, extends it to $\widetilde{V}$, and hashes $(\mathsf{dig}, \cdot) = \mathsf{FEH.Hash}(\mathsf{hk}_{\mathsf{idx}}, \widetilde{V})$.
       (g) The challenger sends $\mathsf{CPRF.Eval}(K_{\mathsf{idx}}, \mathsf{dig})$ to $\mathcal{A}$.
       (h) Additionally, the challenger adds $\mathcal{I}$ to the table T_{evals}.

3. **Challenge Phase:**
   (a) At any point, $\mathcal{A}$ sends the challenge set of indices $\mathcal{I}^*$.
   (b) The challenger retrieves entries for all $j \in \mathcal{I}^*$ from T_{users} and sets $U^* = (\mathsf{pk}_j)_{j \in \mathcal{I}^*}$.
   (c) Let idx^* be the index of the canonical user in the set U^*. The challenger retrieves K_{idx^*} and $\mathsf{hk}_{\mathsf{idx}^*}$ from T_{users}.
   (d) The challenger computes the challenge value y_b.
      – If $b = 0$, the challenger forms $\widetilde{V}^*$ similar to before, hashes $(\mathsf{dig}^*, \cdot) = \mathsf{FEH.Hash}(\mathsf{hk}_{\mathsf{idx}^*}, \widetilde{V}^*)$ and sets $y_0 = \mathsf{CPRF.Eval}(K_{\mathsf{idx}^*}, \mathsf{dig}^*)$.
      – If $b = 1$, the challenger samples a uniformly random key $y_1 \xleftarrow{R} \{0,1\}^\lambda$.
   (e) The challenger sends y_b to $\mathcal{A}$.
4. **Output Phase:**
   (a) $\mathcal{A}$ is allowed to continue making queries as in the **Query Phase**.
   (b) $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

- Hyb_1^b: same as Real^b, except that the challenger guesses the canonical user in the **challenge** query ahead of time. If the guess is wrong, the experiment aborts and outputs a random bit.

  1. **Setup Phase:** The challenger samples a uniform $\mathsf{idx}^* \xleftarrow{R} [q]$.
  2. **Challenge Phase:** If the canonical user in the challenge query is not the user with index idx^*, the experiment aborts and outputs a uniform $b' \xleftarrow{R} \{0,1\}$.

- Hyb_2^b: same as Hyb_1^b, except that the experiment aborts and outputs a random bit if the hash of the challenge set is equal to the hash of any set that was queried using an **evaluation** query and had the same canonical user as the **challenge** query.

  1. **Setup Phase:** The challenger initializes an empty table T_{dig}.
  2. **Query Phase:** Whenever the adversary makes an **evaluation** query with idx^* as the canonical user, the challenger adds the digest dig to T_{dig}.
  3. **Challenge Phase:** If dig^* is already in T_{dig}, the experiment aborts and outputs a random $b' \xleftarrow{R} \{0,1\}$.

- Hyb_3^b: same as Hyb_2^b, except that the experiment aborts and outputs a random bit if the adversary registers any corrupted user with a one-time verification key that is identical to any honest user, with a *valid* signature such that either the message m is different or the signature is different.

  2. **Query Phase:** Whenever the adversary makes a **malicious user registration** query with a public $\mathsf{pk} = (r, m, \mathsf{vk}_u^{\mathsf{OTS}}, \sigma^{\mathsf{OTS}})$, the challenger checks if there already exists an *honest* user entry u with the public key $\mathsf{pk}_u = (r_u, m_u, \mathsf{vk}_u^{\mathsf{OTS}}, \sigma_u^{\mathsf{OTS}})$ where $(m, \sigma^{\mathsf{OTS}}) \neq (m_u, \sigma_u^{\mathsf{OTS}})$. If such an entry exists and $\mathsf{OTS.Verify}(\mathsf{vk}_u^{\mathsf{OTS}}, m, \sigma^{\mathsf{OTS}}) = 1$, then the experiment aborts and outputs a random $b' \xleftarrow{R} \{0,1\}$.

- Hyb_4^b: same as Hyb_3^b, except that the challenger-generated puncturable verification keys are punctured on the one-time verification key of the canonical user.

  1. **Setup Phase:** The challenger pre-samples the one-time signature key pair $(\mathsf{vk}_{\mathsf{idx}^*}^{\mathsf{OTS}}, \mathsf{sk}_{\mathsf{idx}^*}^{\mathsf{OTS}}) \leftarrow \mathsf{OTS.Gen}(1^\lambda)$ of the canonical user idx^*.
  2. **Query Phase:** Whenever the adversary makes an **honest user registration** query, the challenger samples the verification key $\mathsf{vk}_i^{\mathsf{PS}}$ as follows: $(\mathsf{vk}_i^{\mathsf{PS}}, \mathsf{sk}_i^{\mathsf{PS}}) \leftarrow \mathsf{GenPunc}(1^\lambda, \mathsf{vk}_{\mathsf{idx}^*}^{\mathsf{OTS}})$. This also includes the puncturable signature key pair of the canonical user idx^*.

- Hyb_5^b: same as Hyb_4^b, except that r_i for each honest user i is replaced with an encryption of i using a pseudorandom deterministic encryption.

  1. **Setup Phase:** The challenger samples a PDE key $\mathsf{ek} \leftarrow \mathsf{PDE.Gen}(1^\lambda)$.

2. **Query Phase:** Whenever the adversary makes an **honest user registration** query, the challenger computes r_i as follows $r_i = \mathsf{PDE.Enc}(\mathsf{ek}, i)$. This also includes the canonical user idx^*.

- Hyb_6^b: same as Hyb_5^b, except that the challenger samples a PRF key and derives the randomness used to sample the puncturable verification key $\mathsf{vk}_i^{\mathsf{PS}}$ of each honest user i using the PRF on the value i. We refer to the blocks $v_i = (r_i, \mathsf{vk}^{\mathsf{PS}})$ that are sampled this way as *labeled blocks*.

  1. **Setup Phase:** The challenger samples a PRF key $K_\rho \leftarrow \mathsf{PRF.Setup}(1^\lambda, 1^\lambda, 1^{\ell_{\mathsf{GP}}})$.
  2. **Query Phase:** Whenever the adversary makes an **honest user registration** query, the challenger computes the verification key $\mathsf{vk}_i^{\mathsf{PS}}$ as follows: $(\mathsf{vk}_i^{\mathsf{PS}}, \mathsf{sk}_i^{\mathsf{PS}}) = \mathsf{GenPunc}(1^\lambda, \mathsf{vk}_{\mathsf{idx}^*}^{\mathsf{OTS}}; \mathsf{PRF.Eval}(K_\rho, i))$. This also includes the canonical user idx^*.

- Hyb_7^b: same as Hyb_6^b, except that the hash key of the canonical user $\widetilde{\mathsf{hk}}_i$ is now extractable on the function that checks if all blocks are labeled. This can be checked using the PRF key that we introduced in the last hybrid, as well as the PDE decryption key.

  1. **Setup Phase:** Define the function $g : \{0,1\}^{\ell_{\mathsf{ct}}+\ell'_{\mathsf{vk}}} \to \{0,1\}$ as follows:

$$g(r, \mathsf{vk}^{\mathsf{PS}}) = \begin{cases} 1 & (\mathsf{vk}^{\mathsf{PS}}, \cdot) = \mathsf{GenPunc}\Big(1^\lambda, \mathsf{vk}_{\mathsf{idx}^*}^{\mathsf{OTS}}; \mathsf{PRF.Eval}\big(K_\rho, \mathsf{PDE.Dec}(\mathsf{ek}, r)\big)\Big) \\ 0 & \text{otherwise} \end{cases}$$

  and let $f_g(v_1, \ldots, v_n) = \bigwedge_{i\in[n]} g(v_i)$. The hash key of the canonical user is sampled as follows:

$$(\mathsf{hk}_{\mathsf{idx}^*}, \mathsf{td}_{\mathsf{idx}^*}) \leftarrow \mathsf{FEH.SetupBinding}(1^\lambda, n, f_g)$$

- Hyb_8^b: same as Hyb_7^b, except that the PRF key of the canonical user is constrained so that it does *not* evaluate FEH digests that extract to 1, i.e., digest that hash exclusively labeled blocks.

  1. **Query Phase:** The challenger constrains the PRF key of the canonical user idx^* as follows $K'_{\mathsf{idx}^*} = \mathsf{CPRF.Constrain}(K_{\mathsf{idx}^*}, C)$ where

$$C(\mathsf{dig}) = \neg\mathsf{FEH.Extract}(\mathsf{td}^*_{\mathsf{idx}}, \mathsf{dig})$$

  The program P_{idx^*} is sampled with the constrained key, i.e., $P \leftarrow i\mathcal{O}(\mathsf{pkProg}[K'_{\mathsf{idx}^*}, \mathsf{hk}_{\mathsf{idx}^*}, \mathsf{vk}_{\mathsf{idx}^*}^{\mathsf{OTS}}])$. However, the master key K_{idx^*} is stored in T_{users} to answer **evaluation** queries.

Claim B.7. *For any adversary $\mathcal{A}$ that makes at most $q := q(\lambda)$ **honest user registration** queries, it holds that*

$$\Pr[\mathsf{Real}^b(\mathcal{A}) = 1] = q \cdot \Pr[\mathsf{Hyb}_1^b(\mathcal{A}) = 1]$$

Claim B.7 follows by a standard guessing argument.

Lemma B.8. *Assume that Π_{FEH} is ε-collision-resistant. Then for any efficient $\mathcal{A}$ and any $b \in \{0,1\}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that*

$$\Big|\Pr[\mathsf{Hyb}_1^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^b(\mathcal{A}) = 1]\Big| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Lemma B.8 follows by an analogous argument to Lemma 5.3.

Lemma B.9. *Assume that Π_{OTS} satisfies ε-selective string one-time unforgeability. Then for any efficient $\mathcal{A}$ and any $b \in \{0,1\}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that*

$$\Big|\Pr[\mathsf{Hyb}_2^b(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^b(\mathcal{A}) = 1]\Big| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Proof. Fix an adversary $\mathcal{A}$ and let $q := q(\lambda)$ be an upper bound on the number of honest user registration that $\mathcal{A}$ makes. We define a sequence of $q+1$ hybrids. For each $k \in [0, q]$, we define $\mathsf{Hyb}^b_{2,k}$ as follows: The same as Hyb^b_2, except that the experiment aborts and outputs a random bit if the adversary registers any corrupted user with a one-time verification key that is identical to any of the first k honest users that were registered, with a valid signature such that either the message m is different or the signature is different.

2. **Query Phase:** Whenever the adversary makes a **malicious user registration** query with a public $\mathsf{pk} = (r, m, \mathsf{vk}^{\mathsf{OTS}}_u, \sigma^{\mathsf{OTS}})$, the challenger checks if there already exists an *honest* user entry u such that $u \leq k$ with the public key $\mathsf{pk}_u = (r_u, m_u, \mathsf{vk}^{\mathsf{OTS}}_u, \sigma^{\mathsf{OTS}}_u)$ where $(m, \sigma^{\mathsf{OTS}}) \neq (m_u, \sigma^{\mathsf{OTS}}_u)$. If such an entry exists and $\mathsf{OTS.Verify}(\mathsf{vk}^{\mathsf{OTS}}_u, m, \sigma^{\mathsf{OTS}}) = 1$, then the experiment aborts and outputs a random $b' \xleftarrow{R} \{0,1\}$.

Clearly, we have that $\mathsf{Hyb}^b_{2,0}$ is identical to Hyb^b_2 and $\mathsf{Hyb}^b_{2,q}$ is identical to Hyb^b_3. Therefore it suffices to prove that there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $k \in [q]$,

$$\left|\Pr[\mathsf{Hyb}^b_{2,k-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b_{2,k}(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda).$$

By definition of the hybrids, the quantity $\left|\Pr[\mathsf{Hyb}^b_{2,k-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b_{2,k}(\mathcal{A}) = 1]\right|$ is upper bounded by the probability that $\mathcal{A}$ manages to register a corrupted user with a public $\mathsf{pk} = (r, m, \mathsf{vk}^{\mathsf{OTS}}_k, \sigma^{\mathsf{OTS}})$, where $(m, \sigma^{\mathsf{OTS}}) \neq (m_k, \sigma^{\mathsf{OTS}}_k)$ and $\mathsf{OTS.Verify}(\mathsf{vk}^{\mathsf{OTS}}_k, m, \sigma^{\mathsf{OTS}}) = 1$. Denote this probability by $\mu(\lambda)$.

We construct an adversary $\mathcal{B}$ for the selective strong one-time unforgeability game of Π_{OTS}: the adversary $\mathcal{B}$ simulates for adversary $\mathcal{A}$ everything just like the challenger does in Hyb^b_2, except that when registering the honest user k, it sends the message m_k to the Π_{OTS} challenger in order to get a verification key $\mathsf{vk}^{\mathsf{OTS}}_k$ and a signature σ^{OTS}_k, and uses those values for the honest user k. If the adversary $\mathcal{A}$ manages to register a corrupted user with the same verification key $\mathsf{vk}^{\mathsf{OTS}}_k$ and a different message m or signature σ^{OTS}, then $\mathcal{B}$ forwards the message and signature to the Π_{OTS} challenger. Clearly, algorithm $\mathcal{B}$ is efficient if $\mathcal{A}$ is efficient and moreover its advantage in the selective strong one-time unforgeability game is exactly $\mu(\lambda)$. The claim now follows by the ε-selective string one-time unforgeability of Π_{OTS}. □

Lemma B.10. *Assume that Π_{PS} satisfies ε-verification key indistinguishability. Then for any efficient $\mathcal{A}$ and any $b \in \{0,1\}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that*

$$\left|\Pr[\mathsf{Hyb}^b_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b_4(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Proof. Fix an adversary $\mathcal{A}$ and let $q := q(\lambda)$ be an upper bound on the number of honest user registration that $\mathcal{A}$ makes. We define a sequence of $q+1$ hybrids. For each $k \in [0, q]$, we define $\mathsf{Hyb}^b_{3,k}$ as follows: The same as Hyb^b_3, except that the first k challenger-generated puncturable verification keys are punctured on the one-time verification key of the canonical user.

1. **Setup Phase:** The challenger pre-samples the one-time signature key pair $(\mathsf{vk}^{\mathsf{OTS}}_{\mathsf{idx}^*}, \mathsf{sk}^{\mathsf{OTS}}_{\mathsf{idx}^*}) \leftarrow \mathsf{OTS.Gen}(1^\lambda)$ of the canonical user idx^*.

2. **Query Phase:** Whenever the adversary makes an **honest user registration** query, the challenger samples the verification key $\mathsf{vk}^{\mathsf{PS}}_i$ as follows:

$$(\mathsf{vk}^{\mathsf{PS}}_i, \mathsf{sk}^{\mathsf{PS}}_i) \leftarrow \begin{cases} \mathsf{Gen}(1^\lambda) & i > k \\ \mathsf{GenPunc}(1^\lambda, \mathsf{vk}^{\mathsf{OTS}}_{\mathsf{idx}^*}) & i \leq k \end{cases}$$

   This also includes the canonical user idx^*.

Clearly, we have that $\mathsf{Hyb}^b_{3,0}$ is identical to Hyb^b_3 and $\mathsf{Hyb}^b_{3,q}$ is identical to Hyb^b_4. Therefore it suffices to prove that there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $k \in [q]$,

$$\left|\Pr[\mathsf{Hyb}^b_{3,k-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b_{3,k}(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda).$$

We construct an adversary $\mathcal{B}$ for the verification key indistinguishability game of Π_{PS}: the adversary $\mathcal{B}$ simulates for adversary $\mathcal{A}$ everything just like the challenger does in $\mathsf{Hyb}^b_{3,k-1}$ and $\mathsf{Hyb}^b_{3,k}$, except that when registering the honest user k, it sends the message $\mathsf{vk}^{\mathsf{OTS}}_{\mathsf{idx}^*}$ to the Π_{PS} challenger in order to get a verification key $\mathsf{vk}^{\mathsf{PS}}_k$, and uses this verification key for the honest user k. Eventually, algorithm $\mathcal{B}$ outputs whatever bit b' that $\mathcal{A}$ outputs. Note that even though some of the key are punctured, the algorithm $\mathcal{B}$ can still answer all **evaluation** queries, as well as the **challenge** query because when the canonical user is idx^* then it can just directly use its constrained PRF key K_{idx^*} to compute the shared key.

Clearly, algorithm $\mathcal{B}$ is efficient if $\mathcal{A}$ is efficient and moreover its advantage in the verification key indistinguishability game is exactly

$$\left|\Pr[\mathsf{Hyb}^b_{3,k-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b_{3,k}(\mathcal{A}) = 1]\right|.$$

The claim now follows by the ε-verification key indistinguishability of Π_{PS}. □

Lemma B.11. *Assume that Π_{PDE} satisfies ε-ciphertext pseudorandomness. Then for any efficient $\mathcal{A}$ and any $b \in \{0,1\}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that*

$$\left|\Pr[\mathsf{Hyb}^b_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b_5(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Lemma B.11 follows by an analogous argument to Lemma 5.5.

Lemma B.12. *Assume that Π_{PRF} satisfies ε-pseudorandomness. Then for any efficient $\mathcal{A}$ and any $b \in \{0,1\}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that*

$$\left|\Pr[\mathsf{Hyb}^b_5(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b_6(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Lemma B.12 follows by standard reduction to PRF security.

Lemma B.13. *Assume that Π_{FEH} satisfies ε-mode indistinguishability. Then for any efficient $\mathcal{A}$ and any $b \in \{0,1\}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that*

$$\left|\Pr[\mathsf{Hyb}^b_6(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b_7(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Lemma B.13 follows by an analogous argument to Lemma 5.6.

Lemma B.14. *Assume that $i\mathcal{O}$ is an ε-secure indistinguishability obfuscator, Π_{FEH} satisfies resilient function extraction, Π_{CPRF} satisfies constrained correctness, and Π_{PS} satisfies perfect unforgeability. Then for any efficient $\mathcal{A}$ and any $b \in \{0,1\}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \geq \Lambda$ it holds that*

$$\left|\Pr[\mathsf{Hyb}^b_7(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b_8(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Proof. Let $\mathcal{A}$ be an adversary for the adaptive security with static corruptions. We construct an adversary $\mathcal{B}$ for the security game of $i\mathcal{O}$ as follows:

1. Algorithm $\mathcal{A}$ sends n to $\mathcal{B}$.
2. Algorithm $\mathcal{B}$ initializes tables $T_{\mathsf{users}}, T_{\mathsf{evals}}$ and counter $i \leftarrow 0$. It samples the guess $\mathsf{idx}^* \xleftarrow{\mathrm{R}} [q]$. It samples all the necessary keys for Hyb^b_7: ek, pad, K_ρ, and $(\mathsf{vk}^{\mathsf{OTS}}_{\mathsf{idx}^*}, \mathsf{sk}^{\mathsf{OTS}}_{\mathsf{idx}^*})$ (setting $m^* = \mathsf{vk}^{\mathsf{OTS}}_{\mathsf{idx}^*}$).
3. For the canonical user idx^*, algorithm $\mathcal{B}$ samples the PRF key $K_{\mathsf{idx}^*} \leftarrow \mathsf{CPRF.Setup}(1^\lambda, 1^{\ell_{\mathsf{FEH}}}, 1^\lambda)$ and the FEH key $(\mathsf{hk}_{\mathsf{idx}^*}, \mathsf{td}_{\mathsf{idx}^*}) \leftarrow \mathsf{FEH.SetupBinding}(1^\lambda, n, f_g)$ (where f_g is as defined in Hyb^b_7).
4. Algorithm $\mathcal{B}$ additionally computes the constrained key $K'_{\mathsf{idx}^*} = \mathsf{CPRF.Constrain}(K_{\mathsf{idx}^*}, C)$, where the circuit is defined as $C(\mathsf{dig}) = \neg\mathsf{FEH.Extract}(\mathsf{td}_{\mathsf{idx}^*}, \mathsf{dig})$.

5. Algorithm $\mathcal{B}$ sends a challenge to the $i\mathcal{O}$ challenger consisting of two circuits C_0, C_1 which compute the programs $\mathsf{pkProg}[K_{\mathsf{idx}^*}, \mathsf{hk}_{\mathsf{idx}^*}, \mathsf{vk}^{\mathsf{OTS}}_{\mathsf{idx}^*}]$ and $\mathsf{pkProg}[K'_{\mathsf{idx}^*}, \mathsf{hk}_{\mathsf{idx}^*}, \mathsf{vk}^{\mathsf{OTS}}_{\mathsf{idx}^*}]$ from Figure 4, respectively.
6. The $i\mathcal{O}$ challenger responds with an obfuscated program P_{idx^*}.
7. Algorithm $\mathcal{B}$ answers all queries from $\mathcal{A}$.
   - For **honest user registration** $i = \mathsf{idx}^*$, algorithm $\mathcal{B}$ uses the program P_{idx^*} to build $\mathsf{pk}_{\mathsf{idx}^*}$. It stores the original key K_{idx^*} in T_{users} (as specified in Hyb^b_8).
   - For all other queries (**honest user registration** $i \neq \mathsf{idx}^*$, **malicious user registration**, **evaluation**, **challenge**), algorithm $\mathcal{B}$ simulates the challenger exactly as specified in the descriptions of Hyb^b_7 and Hyb^b_8. Note that the logic for all these queries is identical in both hybrids.
8. When $\mathcal{A}$ outputs its guess b', algorithm $\mathcal{B}$ forwards b' as its own output to the $i\mathcal{O}$ challenger.

If $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. Clearly, if P_{idx^*} is an obfuscation of C_0 then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}^b_7(\mathcal{A})$, and if P_{idx^*} is an obfuscation of C_1 then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}^b_8(\mathcal{A})$. For $\mathcal{B}$ to be a valid adversary for the indistinguishability obfuscation game, $C_0 = \mathsf{pkProg}[K_{\mathsf{idx}^*}, \mathsf{hk}_{\mathsf{idx}^*}, \mathsf{vk}^{\mathsf{OTS}}_{\mathsf{idx}^*}]$ and $C_1 = \mathsf{pkProg}[K'_{\mathsf{idx}^*}, \mathsf{hk}_{\mathsf{idx}^*}, \mathsf{vk}^{\mathsf{OTS}}_{\mathsf{idx}^*}]$ have to be equivalent. We prove next that this is indeed the case.

By the constrained correctness of Π_{CPRF}, the two programs can only differ on an input $(\mathsf{dig}, j, v_j, \pi_j, \sigma_j)$ that passes the initial checks (Step 1. and Step 3.) and triggers the constraint $\mathsf{FEH.Extract}(\mathsf{td}_{\mathsf{idx}^*}, \mathsf{dig}) = 1$. Fix an input $\mathsf{dig}, j, v_j, \pi_j, \sigma_j$ to the program and parse $v_j = (r, \mathsf{vk}^{\mathsf{PS}})$. If $\mathsf{PS.Verify}(\mathsf{vk}^{\mathsf{PS}}, \mathsf{vk}^{\mathsf{OTS}}_{\mathsf{idx}^*}, \sigma_j) = 1$ then by the perfect unforgeability of Π_{PS}, the key $\mathsf{vk}^{\mathsf{PS}}$ is not punctured on $\mathsf{vk}^{\mathsf{OTS}}_{\mathsf{idx}^*}$. Namely, there does not exist any string ρ such that

$$(\mathsf{vk}^{\mathsf{PS}}, \cdot) = \mathsf{GenPunc}\big(1^\lambda, \mathsf{vk}^{\mathsf{OTS}}_{\mathsf{idx}^*}; \rho\big).$$

Therefore regardless of the value of r, we get that $g(v_j) = g\big(r, \mathsf{vk}^{\mathsf{PS}}\big) = 0$. By the resilient function extraction, since $\mathsf{hk}_{\mathsf{idx}^*}$ binds on the conjunction of g and $g(v_j) = 0$, we conclude that if $\mathsf{FEH.Verify}(\mathsf{hk}_{\mathsf{idx}^*}, \mathsf{dig}, j, v_j, \pi_j) = 1$ then $\mathsf{FEH.Extract}(\mathsf{td}_{\mathsf{idx}^*}, \mathsf{dig}) = 0$. Thus, the circuits C_0 and C_1 are functionally equivalent.

The claim now follows from the ε-security of $i\mathcal{O}$. □

Lemma B.15. *Assume that Π_{CPRF} satisfies ε-constrained pseudorandomness and Π_{PDE} satisfies correctness. Then for any efficient $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ such that for all $\lambda \geq \Lambda$ and a negligible function $\mathsf{negl}(\lambda)$ it holds that*

$$\left|\Pr[\mathsf{Hyb}^0_8(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^1_8(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Lemma B.15 follows by an analogous argument to Lemma 5.9.

Proof of Theorem B.6. Let $\mathcal{A}$ be an adversary for the adaptive security with static corruptions game against Construction B.5. By Lemmas B.8 to B.14, there exists a $\Lambda_1 \in \mathbb{N}$ and a negligible function $\mathsf{negl}_1(\lambda)$ such that for all $\lambda \geq \Lambda_1$ and all $b \in \{0, 1\}$ it holds that

$$\left|\Pr[\mathsf{Hyb}^b_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^b_8(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}_1(\lambda)$$

By Lemma B.15, there exists a $\Lambda_2 \in \mathbb{N}$ and a negligible function $\mathsf{negl}_2(\lambda)$ such that for all $\lambda \geq \Lambda_2$ it holds that

$$\left|\Pr[\mathsf{Hyb}^0_8(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^1_8(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}_2(\lambda)$$

In total, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for any $\lambda \geq \Lambda = \max\{\Lambda_1, \Lambda_2\}$, we have

$$\left|\Pr[\mathsf{Real}^0(\mathcal{A}) = 1] - \Pr[\mathsf{Real}^1(\mathcal{A}) = 1]\right| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda).$$

Therefore Construction B.5 satisfies ε-adaptive security against static corruptions. □

C Collision-Resistant Function-Extractable Hash

In this section, we prove Proposition 3.3 by constructing a collision-resistant function-extractable hash from a collision-resistant hash and a function-extractable hash.

Definition C.1 (Collision-Resistant Hash)**.** Let $\ell_{\mathsf{in}}(\lambda)$ and $\ell_{\mathsf{out}}(\lambda)$ be any polynomials such that $\ell_{\mathsf{in}}(\lambda) > \ell_{\mathsf{out}}(\lambda)$. A collision-resistant hash (CRH) with input length $\ell_{\mathsf{in}}(\lambda)$ and output length $\ell_{\mathsf{out}}(\lambda)$ is a pair of efficient algorithms $\Pi_{\mathsf{CRH}} = (\mathsf{Setup}, \mathsf{Hash})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{hk}$: On input a security parameter λ, the key generation algorithm outputs a hash key hk.
- $\mathsf{Hash}(\mathsf{hk}, x) \to \mathsf{dig}$: On input a hash key hk and a string $x \in \{0,1\}^{\ell_{\mathsf{in}}(\lambda)}$, the hashing algorithm outputs a short digest $\mathsf{dig} \in \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}$. Note that the length of dig is fixed and independent of $|x|$.

We say Π_{CRH} is ε-collision-resistant if for every efficient adversary $\mathcal{A}$ there exists $\Lambda \in \mathbb{N}$ and a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda > \Lambda$ the following holds:

$$\Pr\left[\begin{array}{c} |x| = |x'| = \ell_{\mathsf{in}}(\lambda) \wedge \\ x \neq x' \wedge \\ \mathsf{Hash}(\mathsf{hk}, x) = \mathsf{Hash}(\mathsf{hk}, x') \end{array} : \begin{array}{c} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda) \\ (x, x') \leftarrow \mathcal{A}(1^\lambda, \mathsf{hk}) \end{array}\right] \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda).$$

We say that Π_{CRH} is polynomially collision-resistant if $\varepsilon(\lambda) = 1$.

Proof of Proposition 3.3 Let $\Pi_{\mathsf{FEH}} = (\mathsf{Setup}, \mathsf{SetupBinding}, \mathsf{Hash}, \mathsf{Verify}, \mathsf{Extract})$ and $\Pi_{\mathsf{CRH}} = (\mathsf{Setup}, \mathsf{Hash})$ be an $\varepsilon_{\mathsf{FEH}}$-secure function-extractable hash and an $\varepsilon_{\mathsf{CRH}}$-secure collision-resistant hash, respectively. We construct a new function-extractable hash Π'_{FEH} in the natural way:

- $\mathsf{Setup}'(1^\lambda, n)$: Samples $\mathsf{hk}_{\mathsf{FEH}} \leftarrow \mathsf{FEH.Setup}(1^\lambda, n)$ and $\mathsf{hk}_{\mathsf{CRH}} \leftarrow \mathsf{CRH.Setup}(1^\lambda)$, and outputs $(\mathsf{hk}_{\mathsf{FEH}}, \mathsf{hk}_{\mathsf{CRH}})$.
- $\mathsf{SetupBinding}'(1^\lambda, n, f)$: Samples $(\mathsf{hk}_{\mathsf{FEH}}, \mathsf{td}_{\mathsf{FEH}}) \leftarrow \mathsf{FEH.SetupBinding}(1^\lambda, n, f)$ and $\mathsf{hk}_{\mathsf{CRH}} \leftarrow \mathsf{CRH.Setup}(1^\lambda)$ and outputs $\mathsf{hk} = (\mathsf{hk}_{\mathsf{FEH}}, \mathsf{hk}_{\mathsf{CRH}})$ and $\mathsf{td} = \mathsf{td}_{\mathsf{FEH}}$.
- $\mathsf{Hash}'\big((\mathsf{hk}_{\mathsf{FEH}}, \mathsf{hk}_{\mathsf{CRH}}), x\big)$: Outputs $\big(\mathsf{FEH.Hash}(\mathsf{hk}_{\mathsf{FEH}}, x), \mathsf{CRH.Hash}(\mathsf{hk}_{\mathsf{CRH}}, x)\big)$.
- $\mathsf{Verify}'\big((\mathsf{hk}_{\mathsf{FEH}}, \mathsf{hk}_{\mathsf{CRH}}), (\mathsf{dig}_{\mathsf{FEH}}, \mathsf{dig}_{\mathsf{CRH}}), i, \sigma, \pi\big)$: Outputs $\mathsf{FEH.Verify}(\mathsf{hk}_{\mathsf{FEH}}, \mathsf{dig}_{\mathsf{FEH}}, i, \sigma, \pi)$.
- $\mathsf{Extract}'\big(\mathsf{td}_{\mathsf{FEH}}, (\mathsf{dig}_{\mathsf{FEH}}, \mathsf{dig}_{\mathsf{CRH}})\big)$: Outputs $\mathsf{FEH.Extract}(\mathsf{td}_{\mathsf{FEH}}, \mathsf{dig}_{\mathsf{FEH}})$.

We argue the properties of Π'_{FEH}:

- **Efficiency, Correctness, Extraction Correctness:** These follow directly from the corresponding properties of Π_{FEH} and Π_{CRH}, as the new algorithms simply invoke the underlying ones.
- **Mode Indistinguishability:** Follows directly from the mode indistinguishability of Π_{FEH}. An adversary distinguishing modes for Π'_{FEH} with advantage at least $\varepsilon_{\mathsf{FEH}}(\lambda) \cdot \frac{1}{\mathsf{poly}(\lambda)}$ can be used to distinguish modes for Π_{FEH} with the same advantage, by simulating the Π_{CRH} component locally.
- **Resilient Function Extraction:** Follows directly from the resilient function extraction of Π_{FEH}. An adversary breaking this property for Π'_{FEH} (providing $\mathsf{dig} = (\mathsf{dig}_{\mathsf{FEH}}, \mathsf{dig}_{\mathsf{CRH}}), i, \sigma, \pi$) can be used to break Π_{FEH} by forwarding $(\mathsf{dig}_{\mathsf{FEH}}, i, \sigma, \pi)$, since Verify' and $\mathsf{Extract}'$ only depend on the Π_{FEH} components.
- **Collision Resistance:** Follows directly from the collision-resistance of Π_{CRH}. An adversary that finds a collision in Π'_{FEH} with probability at least $\varepsilon_{\mathsf{CRH}}(\lambda) \cdot \frac{1}{\mathsf{poly}(\lambda)}$ can be used to find a collision in Π_{CRH} with the same probability, by simulating the Π_{FEH} component locally.

Therefore, Π'_{FEH} is an $\varepsilon_{\mathsf{FEH}}$-secure function-extractable hash that is also $\varepsilon_{\mathsf{CRH}}$-collision-resistant.

D Constructing Function-Extractable Hash

In this section, we prove Theorem 3.6 by constructing function-extractable hash for conjunction of block functions [13] from a leveled homomorphic encryption scheme [Gen09].

Definition D.1 (Leveled Homomorphic Encryption)**.** A *leveled homomorphic encryption* over message space $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ is a tuple of polynomial time algorithms $\Pi_{\mathsf{LHE}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, 1^d) \to (\mathsf{sk}, \mathsf{pk})$: On input a security parameter $\lambda \in \mathbb{N}$ and a depth bound $d \in \mathbb{N}$, the key-generation algorithm outputs a secret key sk and a public key pk.
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{msg}) \to \mathsf{ct}$: On input a public key pk and a message $\mathsf{msg} \in \mathcal{X}_\lambda$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{msg} \in \mathcal{X}_\lambda \cup \{\bot\}$: On input a secret key sk and a ciphertext ct, the decryption algorithm either outputs a plaintext $\mathsf{msg} \in \mathcal{X}_\lambda$ or a special symbol $\mathsf{msg} = \bot$. The decryption algorithm is deterministic.
- $\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}) \to \mathsf{ct}'$: On input a public key pk, a Boolean circuit C, and a ciphertext ct, the homomorphic evaluation algorithm outputs a new ciphertext ct′. The evaluation algorithm is *deterministic*. We extend Eval to take multiple ciphertexts in the natural way.

We require the following properties:

- **Correctness:** For all $\lambda, n \in \mathbb{N}$ and all messages $\mathsf{msg} \in \mathcal{X}_\lambda$ and all Boolean circuits of depth at most C, it holds that:
$$\Pr\left[\ \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, \mathsf{msg})) = \mathsf{msg} \ \ :\ \ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^d)\ \right] = 1.$$
- **Evaluation correctness:** For all $\lambda, n \in \mathbb{N}$, all messages $\mathsf{msg} \in \mathcal{X}_\lambda$, and all Boolean circuits of depth at most C, it holds that:
$$\Pr\left[\ \mathsf{Dec}(\mathsf{sk}, \mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct})) = C(\mathsf{msg}) \ \ :\ \ \begin{array}{c}(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^d)\\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg})\end{array}\ \right] = 1.$$
  We extend this property to evaluations on multiple ciphertexts in the natural way.
- **Compactness:** There exists a polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, d \in \mathbb{N}$, all $(\mathsf{sk}, \mathsf{pk})$ in the support of $\mathsf{KeyGen}(1^\lambda, 1^d)$, all messages $\mathsf{msg} \in \mathcal{X}_\lambda$, all ciphertexts ct in the support of $\mathsf{Enc}(\mathsf{pk}, \mathsf{msg})$, and all Boolean circuits C, it holds that
$$|\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct})| \le \mathsf{poly}(\lambda, \log d, |C(\mathsf{msg})|).$$
- **CPA-security:** For an adversary $\mathcal{A}$ and a bit $b \in \{0, 1\}$, define the CPA-security experiment as follows:
  1. On input the security parameter 1^λ, the adversary $\mathcal{A}$ starts by outputting a depth bound 1^d.
  2. The challenger samples a key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^d)$ and sends pk to the adversary.
  3. The adversary can now make (arbitrarily many) queries on pairs of messages $(\mathsf{msg}_0, \mathsf{msg}_1)$. On each query, the challenger replies with $\mathsf{Enc}(\mathsf{pk}, \mathsf{msg}_b)$.
  4. After the adversary $\mathcal{A}$ is done making queries, it outputs a guess $b' \in \{0, 1\}$.

  We say that Π_{LHE} satisfies CPA security if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that
$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

[13] The construction can be easily extended to disjunction of block functions as well.

Binary tree notation. A complete binary T with depth $\alpha+1$ has $n=2^\alpha$ leaves in level 1. Each leaf has a distinct label in $[n]$. Each non-leaf node has a distinct label in $[n+1, 2n-1]$. For each non-root node i, we denote the parent of i by $\mathsf{parent}(i)$ and the sibling of i by $\mathsf{sibling}(i)$. The children of i are the nodes L, R such that $\mathsf{parent}(L)=\mathsf{parent}(R)=i$. We assume the labeling guarantees that $L, R < i$. We denote by $\ell(i) \in [\alpha+1]$ the level of a node i. The root node r is in level $\ell(r)=\alpha+1$ and every other node i has $\ell(i)=\ell(\mathsf{parent}(i))-1$. For any node i, let $\mathsf{ST}(i)$ denote the set of all *leaves* that belong to the sub-tree of i. For any siblings L, R, it must be that either $\max \mathsf{ST}(L) < \min \mathsf{ST}(R)$ or $\max \mathsf{ST}(R) < \min \mathsf{ST}(L)$. If $\max \mathsf{ST}(L) < \min \mathsf{ST}(R)$, then we say that L is the left sibling and R is the right sibling.

Definition D.2 (Path to Root)**.** Let T be a complete binary tree and i be a leaf in T. We denote the set of nodes on the path from i to the root by $\mathsf{PR}(i)$.

Definition D.3 (Merkle Path)**.** Let T be a complete binary tree with n leaves and i be a leaf in T. The Merkle path $\mathsf{MP}(i)$ consists all of the siblings of nodes in $\mathsf{PR}(i)$ (excluding the root).

Construction D.4 (Function-Extractable Hash for Conjunctions)**.** Let $\Pi_{\mathsf{LHE}}=(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a leveled homomorphic encryption scheme. Our construction of function-extractable hash is almost identical to the function-binding hash construction of [FWW23]. For simplicity, we assume that n is always a power of 2. Finally, since we are constructing the hash function for conjunction of block functions, then we assume that SetupBinding receives the function g rather than f_g. We define the two following circuits:

Constants: Ciphertext ct_0 and a block σ.
Inputs: Secret key sk_0.

1. Decrypts $g=\mathsf{LHE.Dec}(\mathsf{sk}_0, \mathsf{ct}_0)$.
2. Outputs $g(\sigma)$.

Figure 5: Circuit $\mathsf{Base}[\mathsf{ct}_0, \sigma]$.

Constants: Ciphertexts $\mathsf{ct}_L, \mathsf{ct}_R$.
Inputs: A secret key sk_{k-1}.

1. Decrypts $v_L=\mathsf{LHE.Dec}(\mathsf{sk}_{k-1}, \mathsf{ct}_L)$.
2. Decrypts $v_R=\mathsf{LHE.Dec}(\mathsf{sk}_{k-1}, \mathsf{ct}_R)$.
3. If $v_L=0$ or $v_R=0$, outputs 0.
4. Otherwise, outputs 1.

Figure 6: Circuit $\mathsf{Merge}[\mathsf{ct}_L, \mathsf{ct}_R]$.

- $\mathsf{Setup}(1^\lambda, n)$: Denote $\alpha=\log n$.
  1. For each $i=1, \ldots, \alpha+1$, samples a key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{LHE.KeyGen}(1^\lambda, 1^{d'})$.
  2. Samples an additional key pair $(\mathsf{pk}_{\mathsf{func}}, \mathsf{sk}_{\mathsf{func}}) \leftarrow \mathsf{LHE.KeyGen}(1^\lambda, 1^{d'})$.
  3. Encrypts $\mathsf{ct}_{\mathsf{func}} \leftarrow \mathsf{LHE.Enc}(\mathsf{pk}_{\mathsf{func}}, \perp)$, where $\perp$ is a dummy function over $\mathcal{X}_\lambda$.
  4. For each $i=1, \ldots, \alpha+1$, encrypts $\mathsf{ct}_i \leftarrow \mathsf{LHE.Enc}(\mathsf{pk}_i, \mathsf{sk}_{i-1})$.
  5. Outputs $\mathsf{hk}=(\mathsf{pk}_1, \ldots, \mathsf{pk}_{\alpha+1}, \mathsf{ct}_{\mathsf{func}}, \mathsf{ct}_1, \ldots, \mathsf{ct}_{\alpha+1})$.
- $\mathsf{SetupBinding}(1^\lambda, n, g)$: Denote $\alpha=\log n$.

1. For each $i = 1, \ldots, \alpha + 1$, samples a key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{LHE.KeyGen}(1^\lambda, 1^{d'})$.
2. Samples an additional key pair $(\mathsf{pk}_{\mathsf{func}}, \mathsf{sk}_{\mathsf{func}}) \leftarrow \mathsf{LHE.KeyGen}(1^\lambda, 1^{d'})$.
3. Encrypts $\mathsf{ct}_{\mathsf{func}} \leftarrow \mathsf{LHE.Enc}(\mathsf{pk}_{\mathsf{func}}, g)$
4. For each $i = 1, \ldots, \alpha + 1$, encrypts $\mathsf{ct}_i \leftarrow \mathsf{LHE.Enc}(\mathsf{pk}_i, \mathsf{sk}_{i-1})$.
5. Outputs $\mathsf{hk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_{\alpha+1}, \mathsf{ct}_{\mathsf{func}}, \mathsf{ct}_1, \ldots, \mathsf{ct}_{\alpha+1})$ and $\mathsf{td} = \mathsf{sk}_{\alpha+1}$.

- $\mathsf{Hash}(\mathsf{hk}, x)$:
  1. Parses $\mathsf{hk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_{\alpha+1}, \mathsf{ct}_{\mathsf{func}}, \mathsf{ct}_1, \ldots, \mathsf{ct}_{\alpha+1})$ and $x = x_1 \cdots x_n$.
  2. Computes a complete binary tree with n leaves. For each node $i \in [2n-1]$, we associate a ciphertext c_i, which we compute in a bottom up fashion starting from the leaves. Namely, for $i = 1, \ldots, 2n-1$:
     (a) If $i \in [n]$, then sets $c_i = \mathsf{LHE.Eval}(\mathsf{pk}_1, \mathsf{Base}[\mathsf{ct}_{\mathsf{func}}, x_i], \mathsf{ct}_1)$, where $\mathsf{Base}[\mathsf{ct}_{\mathsf{func}}, x_i]$ is described in Figure 5.
     (b) If $i \in [n+1, 2n-1]$, then the algorithm sets $c_i = \mathsf{LHE.Eval}(\mathsf{pk}_{\ell(i)}, \mathsf{Merge}[c_L, c_R], \mathsf{ct}_{\ell(i)})$, where $\mathsf{Merge}[c_L, c_R]$ is described in Figure 6, and L, R are the children of i.
  3. For each $i \in [n]$, sets $\pi_i = \{(c_j, j)\}_{j \in \mathsf{MP}(i)}$.
  4. Outputs $\mathsf{dig} = c_{2n-1}$ and $\pi_1, \ldots, \pi_n$.
- $\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i, \sigma, \pi)$:
  1. Parses $\mathsf{hk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_{\alpha+1}, \mathsf{ct}_{\mathsf{func}}, \mathsf{ct}_1, \ldots, \mathsf{ct}_{\alpha+1})$ and $\pi = \{(c_{j_1}, j_1), \ldots, (c_{j_\alpha}, j_\alpha)\}$, where $j_k < j_{k+1}$ for all k.
  2. Let $i_1, \ldots, i_{\alpha+1}$ be the path starting from i and ending in the root node $2n-1$, where $i_k < i_{k+1}$ for all k.
  3. Computes $c_{i_1} = \mathsf{LHE.Eval}(\mathsf{pk}_1, \mathsf{Base}[\mathsf{ct}_{\mathsf{func}}, \sigma], \mathsf{ct}_1)$.
  4. For $k = 2, \ldots, \alpha + 1$: computes $c_{i_k} = \mathsf{LHE.Eval}(\mathsf{pk}_k, \mathsf{Merge}[c_{i_{k-1}}, c_{j_{k-1}}], \mathsf{ct}_k)$.
  5. Checks that $\mathsf{dig} = c_{i_{\alpha+1}}$.
- $\mathsf{Extract}(\mathsf{td}, \mathsf{dig})$: Outputs $\mathsf{LHE.Dec}(\mathsf{td}, \mathsf{dig})$.

Correctness, succinctness and mode indistinguishability have all been proved in [FWW23, Theorems 3.6 and 3.7]. Since our construction proofs would be identical, we refer the reader to [FWW23].

Lemma D.5. *If Π_{LHE} satisfies correctness and evaluation correctness, then Construction D.4 satisfies resilient function extraction.*

Proof. Fix any $\lambda, n \in \mathbb{N}$ and function $f_g \in \mathcal{F}_\lambda$, where $g : \mathcal{X}_\lambda \to \{0,1\}$. Let $(\mathsf{pk}_0, \mathsf{sk}_0), \ldots, (\mathsf{pk}_\alpha, \mathsf{sk}_\alpha)$ be any key pairs, and let $\mathsf{ct}_0, \ldots, \mathsf{ct}_\alpha$ be any ciphertexts sampled while running $\mathsf{SetupBinding}(1^\lambda, n, f)$. Denote $\mathsf{hk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\alpha, \mathsf{ct}_0, \ldots, \mathsf{ct}_\alpha)$. In addition, fix a digest dig, an index $i \in [n]$, a block σ and an opening π.

Assume that $\mathsf{Verify}(\mathsf{hk}, \mathsf{dig}, i, \sigma, \pi) = 1$ and $g(\sigma) = 0$. We need to prove that $\mathsf{Dec}(\mathsf{sk}_{\alpha+1}, \mathsf{dig}) = 0$. First, parse $\pi = \{(c_{j_1}, j_1), \ldots, (c_{j_\alpha}, j_\alpha)\}$, and let $i_1, \ldots, i_{\alpha+1}$ be the path starting from i and ending in the root, where $i_k < i_{k+1}$ for all k. Let $c_{i_1} = \mathsf{LHE.Eval}(\mathsf{pk}_1, \mathsf{Base}[\mathsf{ct}_{\mathsf{func}}, \sigma], \mathsf{ct}_1)$ and for each $k \in [2, \alpha+1]$, let $c_{i_k} = \mathsf{LHE.Eval}(\mathsf{pk}_k, \mathsf{Merge}[c_{i_{k-1}}, c_{j_{k-1}}], \mathsf{ct}_k)$. We prove by induction on $k \in [\alpha + 1]$ that $\mathsf{LHE.Dec}(\mathsf{pk}_k, c_{i_k}) = 0$. The base follows by the correctness of Π_{LHE}. Assume $\mathsf{LHE.Dec}(\mathsf{pk}_{k-1}, c_{i_{k-1}}) = 0$. Then by the specification of $\mathsf{Merge}[c_{i_{k-1}}, c_{j_{k-1}}]$ and the correctness of Π_{LHE}, we have that $\mathsf{Merge}[c_{i_{k-1}}, c_{j_{k-1}}](\mathsf{sk}_{k-1}) = 0$ regardless of $c_{j_{k-1}}$. By the evaluation correctness of Π_{LHE} and the fact that ct_k is an encryption of sk_{k-1} under pk_k, we conclude that $\mathsf{Dec}(\mathsf{sk}_k, c_{i_k}) = 0$. By applying the inductive claim to $c_{i_{\alpha+1}}$ and recalling that Verify checks $\mathsf{dig} = c_{i_{\alpha+1}}$, we conclude $\mathsf{Dec}(\mathsf{sk}_{\alpha+1}, \mathsf{dig}) = 0$ and the claim follows. □

Theorem 3.6 now follows from Lemma D.5 and [FWW23].