Adaptive Multiparty NIKE

Venkata Koppula[1], Brent Waters[2,3], and Mark Zhandry[2,4]

[1] IIT Delhi, Delhi, India, kvenkata@cse.iitd.ac.in
[2] NTT Research, Sunnyvale, USA
[3] UT Austin, Austin, USA, bwaters@cs.utexas.edu
[4] Princeton University, Princeton, USA, mzhandry@gmail.com

Abstract. We construct adaptively secure multiparty non-interactive key exchange (NIKE) from polynomially-hard indistinguishability obfuscation and other standard assumptions. This improves on all prior such protocols, which required sub-exponential hardness. Along the way, we establish several compilers which simplify the task of constructing new multiparty NIKE protocols, and also establish a close connection with a particular type of constrained PRF.

1 Introduction

Non-interactive key exchange (NIKE) is a fundamental application in public key cryptography. In a G-party NIKE protocol, a group of G users simultaneously publish individual public keys to a bulletin board, keeping individual secret keys to themselves. Then just by reading the bulletin board and using their individual private keys but no further interaction, the G users can arrive at a common key hidden to anyone outside the group.

In this work, we build multiparty NIKE attaining *adaptive* security under polynomially-hard non-interactive assumptions. Our assumptions are indistinguishability obfuscation (iO) and standard assumptions on cryptographic groups[1]. The main restriction is that we must bound the number of users that can be adaptively corrupted. The number of honest users, and even the number of adversarially generated users, can be unbounded; only the number of users that were initially honest and later corrupted must be bounded. This improves on prior standard-model adaptively secure schemes [BZ14, Rao14], which all bound the *total* number of users, and also required either *interactive* or *sub-exponential* assumptions. Along the way, we several compilers to simplify the design process of iO-based multiparty NIKE. We also explore adaptive security for constrained PRFs, giving a new construction for "one symbol fixing" constraints, and show a close connection to multiparty NIKE.

1.1 Prior Work and Motivation

NIKE has a long history, with the 2-party case dating back to the foundational work of Diffie and Hellman [DH76], and the multiparty case already re-

[1] We note two uses of the term "group": the group of users establishing a shared key, and the cryptographic group used as a tool. Which use should be clear from context.

ferred to as “a long-standing open problem” in 2002 [BS02]. Joux gave a 3-party protocol from pairings [Jou00]. The first protocol for $G \geq 4$ used multilinear maps [GGH13], though the only protocols directly based on multilinear maps that have not been attacked are limited to a *constant* number of users [MZ18]. Currently, the only known solutions for a super-constant number of users are built from indistinguishability obfuscation (iO). The first such construction for polynomially-many users was due to Boneh and Zhandry [BZ14] (using punctured programming techniques [SW14]), with a number of follow-up works [Rao14, KRS15, HJK+16, MZ17, GPSZ17, BGK+18].

Multiparty NIKE remains a fascinating object: the central feature of *non-interactive* key exchange (as opposed to protocols requiring multiple interaction rounds) is that public keys can be re-used across many groups, simplifying key management and significantly reducing communication. This feature makes NIKE an important tool with many applications. Multiparty NIKE in particular is a useful tool for group key management [STW96] and broadcast encryption with small parameters [BZ14]. Multiparty NIKE is also interesting from a foundational perspective, being perhaps the simplest cryptographic object which currently is *only* known via obfuscation[2].

Adaptive Security. The re-use of public keys in a NIKE protocol, on the other hand, opens the door to various *active* attacks. For example, if a shared key for one group is accidentally leaked, it should not compromise the shared key of other groups, including those that may intersect. Worse, an adversary may participate in certain groups using maliciously generated public keys, or may be able to corrupt certain users. Finally, decisions about which groups' shared keys to compromise, how the adversary devises its own malicious public keys, which users to corrupt, and even which set of users to ultimately attack, can all potentially be made *adaptively*.

Adaptive security is an important goal in cryptography generally, being the focus of hundreds if not thousands of papers. Numerous works have considered adaptive NIKE. In the 2-party case, adaptive security can often be obtained *generically* by guessing the group that the adversary will attack. If there are a total of N users in the system, the reduction loss is N^2, a polynomial. The focus of works in the 2-party case (e.g. [CKS08, FHKP13, BJLS16, HHK18, HHKL21]) has therefore been *tight* reductions, which still remains unresolved.

The situation becomes more critical in the multiparty case, where the generic guessing reduction looses a factor of $\binom{N}{G} \approx N^G$, which is exponential for polynomial group size G. In order to make this generic reduction work, one must assume the (sub)exponential hardness of the underlying building blocks and scale up the security parameter appropriately. This results in qualitatively stronger underlying computational assumptions. A couple works have attempted to improve on this reduction, achieving security in the random oracle model [HJK+16], or

[2] Multiparty NIKE can also be built via functional encryption [GPSZ17], which is equivalent to iO [BV15a, AJ15] under sub-exponential reductions.

under *interactive* assumptions [BZ14, Rao14] [3]. In fact, Rao [Rao14] argues that exponential loss or interactive assumption is likely necessary, giving a black box impossibility of a polynomial reduction to non-interactive assumptions. This impossibility will be discussed in more depth momentarily. We also note existing standard-model adaptively secure schemes all limit the *total* number of users, including both honest *and* dishonest users, to an a priori polynomial bound.

Constrained PRFs. A constrained PRF is a pseudorandom function which allows the key holder to produce *constrained* keys k_C corresponding to functions C. The key k_C should allow for evaluating the PRF on any input x where $C(x) = 1$, but the output should remain pseudorandom if $C(x) = 0$. First proposed in three concurrent works [BW13, KPTZ13, BGI14], constrained PRFs have become a fundamental concept in cryptography, with many follow-up works (e.g. [BV15b, BFP$^+$15, CRV16, DKW16, CC17, BTVW17, AMN$^+$18]). A particularly interesting class of constrained PRFs are those for *bit-fixing* constraints, which give secret key broadcast encryption [BW13], for example.

Adaptivel secure constrained PRFs of of particular interest [Hof14, FKPR14, HKW15, HKKW14, DKN$^+$20]. Unfortunately, with one exception, all known adaptively secure constrained PRFs require random oracles, super-polynomial hardness, or a constant collusion resistance bound. The one exception is [HKW15] for simple puncturing constraints, where C contains a list of polynomially-many points, and accepts all inputs not in the list. Even with such simple constraints, the construction requires iO, algebraic tools, and a non-trivial proof.

1.2 Technical Challenges

Rao's impossibility. Rao [Rao14] proves that multiparty NIKE protocols with standard model proofs relative to non-interactive assumptions (including iO) must incur an exponential loss. The proof follows a meta-reduction, which runs the reduction until the reduction receives the challenge from the underlying non-interactive assumption. At this point, Rao argues that the adversary need not commit to the group it will attack. Now, we split the reduction into two branches:

- In the first branch, choose and corrupt an arbitrary honest user i, obtaining secret key sk_i. Then abort the branch.
- In the second branch, choose the group S to attack such that (1) S contains only honest users for this branch, and (2) $i \in S$. User i is honest in this branch since it was never corrupted here, despite being corrupted in the other branch. Use sk_i to compute the shared group key.

From the view of the reduction, the second branch appears to be a valid adversary. Hence, by the guarantees of the reduction, it must break the underlying hard problem, a contradiction. Hence, no such reduction could exist.

Rao's proof is quite general, and handles reductions that may rewind the adversary or run it many times concurrently. It also works in the more restricted setting where there is an upper bound on the total number of users in the system.

[3] Note that multiparty NIKE itself is an interactive assumption.

There is *one* way in which Rao's result does not completely rule out a construction: in order to guarantee that the second branch is successful, one needs that the shared key derived from sk_i must match the shared key in the second branch. This would seem to follow from correctness, as i is a member of the group S. However, correctness only holds with respect to honestly generated public and secret keys. The reduction may, however, give out malformed public or secret keys that are indistinguishable from the honest keys. In this case, it may be that sk_i actually computes the wrong shared key, causing the meta-reduction to fail.

Rao therefore considers "admissible reductions" where, roughly, the public keys of users outputted by the reduction, even if not computed honestly, uniquely determine the shared key. Analogous lower bounds have been shown for tight reductions in the 2-party setting [BJLS16, HHK18, HHKL21], making similar restrictions on the reduction referred to as "committing reductions."

All existing reductions for multiparty NIKE from iO are admissible. A closer look reveals that all such schemes derive the shared key from a constrained PRF applied to the public values of the users. While the secret key is used to compute this value, the value itself is not dependent on the secret key, only the public key. Therefore, Rao's impossibility captures all the existing techniques, and new ideas are required to achieve adaptive security from static polynomial assumptions.

Dual system methodology? The situation is reminiscent of HIBE and ABE, where Lewko and Waters [LW14] showed that adaptive security cannot be proved under polynomially hard non-interactive assumptions, using reductions that always output secret keys which decrypt consistently. Solutions overcoming this barrier were already known, say based on dual system encryption [Wat09, LOS$^+$10]. The point of [LW14] was to explain necessary features of those proofs.

The multiparty NIKE setting appears much more challenging. HIBE and ABE benefit from a central authority which issues keys. In the proof, the reduction provides the adversary with all of the keys, which will have a special structure that allows for decrypting some ciphertexts and not others. In the NIKE setting, the adversary is allowed to introduce *his own* users. This presents many challenges as we cannot enforce any dual system structure on such users. It also gives the adversary a lot more power to distinguish the *reduction's* keys from honestly generated keys, as the adversary can request the shared keys of groups containing both honest and malicious users.

Recently, Hesse et al.[HHKL21] circumvent the above barriers in the 2-party setting. However there is no obvious analog to the multiparty setting.

Another barrier: adaptive constrained PRFs. Looking ahead, we will show that adaptive multiparty NIKE implies adaptive constrained PRFs for a "one symbol fixing" functionality (1-SF-PRF). Here, inputs are words over a polynomial-sized alphabet Σ, and constrains have the form $(?, ?, \cdots, ?, s, ?, \dots)$, constraining only a single position to some character. The resulting PRFs are fully collusion resistant. 1-SF-PRFs can be seen as a special case of bit-fixing PRFs, where only a single contiguous block of bits can be fixed. Adaptive constrained PRFs for even very simple functionalities have remained a very challenging open question.

In particular, no prior standard-model construction from polynomial hardness achieves functionalities that have a superpolynomial number of both accepting and rejecting inputs. Any adaptive multiparty NIKE construction would along the way imply such a functionality, representing another barrier.

1.3 Result Summary

- We give several compilers, allowing us to simplify the process of designing multiparty NIKE schemes. One compiler shows how to generically remove a common setup from multiparty NIKE (assuming iO). We note that many iO-based solutions could be tweaked to remove setup, but the solutions were ad hoc and in the adaptive setting often required significant effort; we accomplish this generically.
  Another compiler shows that it suffices to ignore the case where the adversary can compromise the security of shared keys for a different groups of users. That is, we show how to generically compile any scheme that is secure against adversaries that *cannot* compromise shared keys into one that is secure even if the adversary *can*.
- We show a close connection between multiparty NIKE and 1-SF-PRFs:
  - Adaptively secure multiparty NIKE implies adaptively secure one-symbol-fixing PRF.
  - One-symbol-fixing PRFs, together with iO, imply a multiparty NIKE protocol with a bounded number of honest users (and hence also corruption queries) and group size, but an unbounded number of malicious users. This result starts by constructing a weaker NIKE protocol, and then applying our compilers.
- We construct adaptively secure 1-SF-PRFs from iO and DDH, thus obtaining multiparty NIKE from the same assumptions with bounded honest users.
- We give a direct construction of multiparty NIKE from iO and standard assumptions on groups, allowing for an unbounded number of honest users. The construction roughly follows the path above, but opens up the abstraction layers and makes crucial modifications to attain the stronger security notion. The main limitation is that there is still a bound on the number of users that the adversary can adaptively corrupt, as well as on the group size.

1.4 Technical Overview

We first briefly recall the types of queries an adversary can make:

- **Corrupt User.** The adversary selects an honest user's public key, and learns the secret key.
- **Shared Key.** The adversary selects a list of public keys, which may contain both honest users adversarially-generated users, and learns the shared key for the group of users. Since the adversary's public keys may be malformed, different users may actually arrive at different shared keys. So the query specifies which of the users' version of the shared key is revealed.
- **Challenge.** Here, the adversary selects a list of honest public keys, and tries to distinguish the shared key from random.

Upgrading NIKE. In addition to providing the first iO-based NIKE, Boneh and Zhandry [BZ14] also construct the first NIKE without a trusted setup, or crs. Their basic idea is to first design an iO-based protocol *with* a crs, but where the resulting crs is only needed to generate the shared keys, but not the individual public keys. Then they just have every user generate their own crs; when it comes time to compute the shared key for a group, the group arbitrarily selects a "leader" and uses the leader's crs.

The above works in the selective setting. However, in the adaptive setting, problems arise. The crs contains an obfuscated program that is run on the user's secret key. The adversary could therefore submit a Shared Key query on an adversarial public key containing a malicious crs. If that malicious user is selected as the leader for the group, honest users' secret keys will be fed into the malicious program, the output being revealed to the adversary, leading to simple attacks. Worse, in Rao's basic scheme with setup, the users need to know the crs in order to generate their public key. So in the setup-less scheme, each user would need to wait until the leader outputs their crs before they can publish their public key, resulting in an interactive protocol. Boneh and Zhandry and later Rao [Rao14] therefore devised more sophisticated techniques to remove the trusted setup.

Our first result sidesteps the above difficulties, by considering the setting where Shared Key queries are not allowed. In this setting, we can make the above strategy of having each party run their own trusted setup fully generic. To accommodate the case where the public keys may depend on the trusted setup, we actually have each user produce an obfuscation of a program that takes as input the crs, and samples a public key. In order to prove security, we also have the secret key for a user be an obfuscated program, which is analogous to the public key program except that is samples the corresponding secret key. In the reduction, this allows us to adaptively embed information in the secret key, which is needed to get the proof to work. See Section 3.2 for details.

Then we show how to generically lift any NIKE scheme that does not support Shared Key queries into one that does support them, without any additional assumptions. Combined with the previous compiler, we therefore eliminate the crs *and* add Shared Key queries to any scheme. The high-level idea is to give the reduction a random subset of the secret keys for honest users. The hope is that these keys will be enough to answer all Shared Key queries, while *not* allowing the reduction to answer the Challenge query. This requires care, as this will not be possible if some of the Shared Key queries have too much overlap with the Challenge query. See Section 3.3 for details.

Connection to Constrained PRFs. Multi-party NIKE already had a clear connection to constrained PRFs, with all iO-based NIKE crucially using constrained PRFs. In Section 4, we make this precise, showing that one symbol fixing (1-SF) PRFs are *equivalent* to NIKE, assuming iO.

One direction is straightforward: to build a 1-SF PRF from multiparty NIKE, create $n \times |\Sigma|$ users, which are arranged in an $|\Sigma| \times n$ grid. Each input in Σ^n then selects a single user from each column, and the value of the PRF is the shared key for the resulting set of n users. To constrain the ith symbol to be σ, simply reveal the secret key for user σ in column i.

The other direction is more complicated, and requires additionally assuming iO. The high-level idea is that the shared key for a group of users will be a PRF evaluated on the list of the users' public keys. If we pretend for the moment that user public keys come from a polynomial-sized set Σ, we could imagine using a 1-SF PRF for this purpose.

Following most iO-based NIKE protocols, we will then have a crs be an obfuscated program which takes as input the list of public keys, together with one of the users secret keys, and evaluates the PRF if the secret key is valid. Our novelty is how we structure the proof to attain adaptive security. Observe that user σ's secret key allows them to evaluate the PRF on any input that contains at least one σ. This is the union of the inputs that can be computed by keys that constrain symbol i to σ, as i ranges over all input positions.

We therefore switch to a hybrid where user σ has the aforementioned constrained keys covertly embedded in their secret key. In this hybrid, we crucially allow the reduction to generate the user's public key without knowing the constrained keys, and only later when the adversary makes a corruption query will it query for the constrained keys and construct the user's secret key. This strategy is our first step to overcoming Rao's impossibility result: the shared key is no longer information-theoretically determined by the public keys, and is only determined once the secret key with the embedded constrained key is specified. We note, however, that a version of Rao's impossibility still applies to the underlying adaptively secure constrained PRFs, which we will have to overcome later when constructing our PRF.

Moving to this hybrid is accomplished using a simplified version of delayed backdoor programming [HJK+16]. After switching the secret keys for each user, we switch the crs program to use the embedded constrained keys to evaluate the PRF, rather than the master key. At this point, adaptive NIKE security follows directly from adaptive 1-SF PRF security.

Of course, NIKE protocols cannot have public keys in a polynomial-sized set. Our actual protocol first generically compiles a 1-SF PRF into a more sophisticated constrained PRF where now Σ is exponentially large. By adapting the above sketch to this special kind of constrained PRF, we obtain the full proof. See Section 4 for details.

Constructing 1-SF PRFs. We turn to constructing a 1-SF PRF. As mentioned above, a version of Rao's impossibility result still applies even to constrained PRFs. Namely, an "admissible" reduction would commit at the beginning of the experiment to the PRF functionality it provides to the adversary. Such an admissible reduction cannot be used to prove adaptive security for constrained PRFs, for almost identical reasons as with Rao's impossibility. This means our reduction must actually have the PRF seen by the adversary be specified dynamically, where its outputs are actually dependent on prior queries made by the adversary.

One may be tempted to simply obfuscate a puncturable PRF. Boneh and Zhandry [BZ14] show that this gives a constrained PRF for any constraint, though only with selective security. Unfortunately, it appears challenging to to get adaptively secure constrained PRFs with this strategy. In particular, the punctured PRF specifies the value of the PRF at all points but one, which is problematic given that we need to dynamically determine the PRF function in order to circumvent Rao's impossibility.

We will instead use algebraic tools to achieve an adaptively secure construction. Our PRF will be Naor-Reingold [NR97], but adapted from a binary alphabet to a polynomial-sized alphabet. The secret key contains $n \times |\Sigma|$ random values $e_{j,\sigma}$, and the PRF on input $(x_1, \ldots, x_n) \in \Sigma^n$ outputs

$$\mathsf{F}(k, x) = h^{\prod_{i=1}^{n} e_{i,x_i}} ,$$

where h is a random generator of a cryptographic group. Without using any computational assumptions, F is already readily seen to be a 1-SF constrained PRF for *a single constrained key*. To constrain position i to σ, simply give out $e_{i,\sigma}$ and $e_{j,x}$ for all $x \in \Sigma$ and all $j \neq i$.

However, we immediately run into trouble even for two constrained keys, since constrained keys for two different i immediately yield the entire secret key. Instead, we constrain keys in this way, except that we embed the constrained keys in an obfuscated program. While this is the natural approach to achieve many-key security, it is a priori unclear how to actually prove security.

We show that obfuscating the constrained keys does in fact upgrade the single-key security of the plain scheme to many-time security. The proof is quite delicate. Essentially, we move to a hybrid where each constrained key uses its own independent h. The main challenge is that, since multiple keys will be able to compute the PRF at the same point, we need to ensure consistency between the keys. Our proof has each constrained key only use its particular h for inputs that cannot be computed by previous constrained keys. For outputs that can be computed by previous keys, the new constrained key will use the h for those keys.

Interestingly, this means that keys in this hybrid must actually contain the h's of all previous constrained keys, and the evaluation of the PRF will actually depend on the order constrained keys are queried. The salient point is that, when the ith constrained key query is made, we only commit to the structure of the PRF on the points that can be evaluated by the first i queries, but the PRF on the remaining part of the domain is unspecified. Structuring the proof in this way is the main insight that allows us to circumvent Rao's impossibility and prove adaptive security.

By careful iO arguments, we show that we are able to move to such a setting where the h for different pieces are random independent bases. The challenge query is guaranteed to be in its own piece, using a different h than all the constrained keys. Therefore, once we move to this setting the constrained keys do not help evaluate the challenge, and security follows. See the Section 5 for details. By combining with our compilers, we obtain the following:

Theorem 1 (Informal). *Assuming polynomial iO and DDH, there exist an adaptively secure multiparty NIKE where the number of honestly generated users is a priori bounded, but where the number of maliciously generated users is unbounded.*

In addition to improving to only polynomial hardness, the above improves on existing works by enhancing the security definition to allow an unbounded number of malicious users.

Our Final Construction. Finally, we give another NIKE construction which further improves on the security attained in Theorem 1, at the cost of a slightly stronger group-based assumption:

Theorem 2 (Informal). *Assuming polynomial iO and the DDH-powers assumption, there exist an adaptively secure multiparty NIKE where the group size and number of corruptions is bounded, but otherwise the number of honest and malicious users unbounded.*

We note that bounding the number of corruptions is very natural, and has arisen in many cryptographic settings under the name "bounded collusions." Examples include traitor tracing [CFN94], Broadcast encryption [FN94], identity-based encryption [DKXY02] and its generalizations to functional encryption [GVW12], to name a few. Bounded collusions are often seen as a reasonable relaxation, and in many cases are stepping-stones to achieving full security. We view bounded collusion security for NIKE similarly, except that in some ways, bounded corruptions for NIKE is even stronger than bounded collusions, in that we allow the NIKE adversary to control an *unbounded* number of users, only limiting the number of users that can be corrupted adaptively.

In our construction, we no longer go through 1-SF-PRFs explicitly, but instead open up the layers of abstraction that gave Theorem 1 and make several crucial modifications to the overall protocol. The main technical challenge is that, in our proof of security for 1-SF-PRFs, we must hard-code all prior queries into each secret key. In the obtained NIKE scheme, this means hard-coding all the keys of users generated by the challenger. But as the number of hard-coded users can never be more than the bit-length of the secret key, this limits the number of honest users.

In our solution, we no longer explicitly hardcode the challenger-generated users, but switch to a hybrid where they are generated with a trigger. Only the obfuscated programs can detect this trigger so that they look like honestly generated users, and it moreover is impossible for the adversary to generate users with the trigger. By a delicate hybrid argument, we are able to mimic the security proof above using these triggers instead of the explicitly hardcoded public keys. See the Full Version [KWZ22] for details.

Note that the DDH-powers assumption is a q-type assumption, but this can be proved from a single assumption in the composite order setting, assuming appropriate subgroup decision assumptions [CM14].

1.5 Organization

Section 2 covers the definitions of multiparty NIKE and constrained PRFs that we will study. Section 3 gives our compilers for enhancing multiparty NIKE. Section 4 demonstrates the equivalence of 1-SF-PRFs and multiparty NIKE in the iO setting. Section 5 gives our construction of 1-SF-PRFs from iO. Due to lack of space, the proof of Theorem 2 removing the bound on the number of honest users is deferred to the Full Version [KWZ22].

2 Preliminaries

2.1 Multiparty NIKE

Here, we define the version of NIKE that we will be considering.

Definition 1 (Multiparty NIKE, Syntax). *A multiparty NIKE scheme with bounded honest users is a pair* $(\mathsf{Pub}, \mathsf{KeyGen})$ *with the following syntax:*

– $\mathsf{Pub}(1^\lambda, 1^\ell, 1^n, 1^c)$ *takes as input the security parameter* λ*, an upper bound* n *on the number of honest users, an upper bound* ℓ *on the number of users in a set, and an upper bound* c *on the number of corruptions. It outputs a public key* pk *and secret key* sk*.*
– $\mathsf{KeyGen}(U, \mathsf{sk})$ *takes as input a list* U *of* $t \leq \ell$ *public keys, plus the secret key for one of the public keys. It outputs a shared key. We have the following correctness guarantee: for any* $\ell, n, c > 0, t \in [\ell]$ *and any* $i, j \in [t]$*,*

$$\Pr[\mathsf{KeyGen}(\{\mathsf{pk}_1, \dots, \mathsf{pk}_t\}, \mathsf{sk}_i) = \mathsf{KeyGen}(\{\mathsf{pk}_1, \dots, \mathsf{pk}_t\}, \mathsf{sk}_j)] \geq 1 - \mathsf{negl}$$

where the probability is over $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda, 1^\ell, 1^n, 1^c)$ *for* $i = 1, \dots, t$*.*

Enhanced correctness notions. As a technical part of our compilers, we will also consider stronger variants of correctness. The first is *perfect correctness*, where the probability above is exactly 1. The second notion is *adversarial correctness*, which is defined via the following experiment with an adversary $\mathcal{A}$:

– On input 1^λ, $\mathcal{A}$ computes $1^\ell, 1^n, 1^c$.
– The challenger runs $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{Pub}(1^\lambda, 1^\ell, 1^n, 1^c)$ for $b = 0, 1$, and sends $\mathsf{pk}_0, \mathsf{pk}_1$ to $\mathcal{A}$
– $\mathcal{A}$ then computes a set U of public keys such that $|U| \leq \ell$ and $\mathsf{pk}_0, \mathsf{pk}_1 \in U$.
– The challenger computes $k_b = \mathsf{KeyGen}(U, \mathsf{sk}_b)$ for $b = 0, 1$. $\mathcal{A}$ wins if and only if $k_0 \neq k_1$.

A NIKE scheme is adversarially correct if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function ϵ such that the $\mathcal{A}$ wins with probability at most ϵ.

Definition 2 (Multiparty NIKE, Adaptive Security). *Consider the following experiment with an adversary* $\mathcal{A}$*:*

– *The challenger initializes empty tables T and U. T will contain records* $(\mathsf{pk}, \mathsf{sk}, b)$ *where* pk, sk *are the public key and secret key for a user, and b is a flag bit indicating if the user is honest (0) or corrupted (1). We will maintain that if the flag bit is 0, then* $\mathsf{sk} \neq \perp$. *U will contain sets of public keys. The challenger also stores a set S^*, initially set to $\perp$.*
– *$\mathcal{A}$ receives 1^λ, replies with $1^\ell, 1^n, 1^c$, and then makes several kinds of queries:*
  - **Register Honest User.** *Here, $\mathcal{A}$ sends nothing. The challenger runs* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Pub}(1^\lambda, 1^\ell, 1^n, 1^c)$. *If there is a record containing* pk *in T, the challenger replies with $\perp$. Otherwise, it adds* $(\mathsf{pk}, \mathsf{sk}, 0)$ *to T, and sends* pk *to $\mathcal{A}$. The total number of such queries is not allowed to exceed n.*
  - **Corrupt User.** *Here, $\mathcal{A}$ sends an* pk. *The challenger finds a record* $(\mathsf{pk}, \mathsf{sk}, 0)$ *in the table T. If no such record is found, or if a record is found but with flag bit set to 1, the challenger replies with $\perp$. Otherwise it replies with* sk. *It then updates the record in T to* $(\mathsf{pk}, \mathsf{sk}, 1)$. *The total number of such queries is not allowed to exceed c.*
  - **Register Malicious User.** *Here, $\mathcal{A}$ sends a public key* pk. *If there is no record in T containing* pk, *the challenger adds to T the record* $(\mathsf{pk}, \perp, 1)$. *There is no limit to the number of such queries.*
  - **Shared Key.** *The adversary sends an unordered set* $S = (\mathsf{pk}_1, \dots, \mathsf{pk}_t)$ *of up to $t \leq \ell$ distinct public keys, as well as an index $i \in [t]$. If $S^* \neq \perp$ and $S = S^*$, then the challenger replies with $\perp$. Otherwise, the challenger checks for each $j \in [t]$ if there is a* $(\mathsf{pk}_j, \mathsf{sk}_j, b_j) \in T$. *Moreover, it checks that* $\mathsf{sk}_i \neq \perp$. *If any of the checks fail, the challenger replies with $\perp$. If all the checks pass, the challenger replies with* $\mathsf{KeyGen}(S, \mathsf{sk}_i)$. *It adds the list S to U. There is no limit to the number of such queries.*
  - **Challenge.** *The adversary makes a single challenge query on an unordered list* $S = (\mathsf{pk}_1^*, \dots, \mathsf{pk}_t^*)$ *of up to $t \leq \ell$ distinct public keys. The challenger sets $S^* = S$. The challenger then checks for each $j \in [t]$ that there is a record* $(\mathsf{pk}_j^*, \mathsf{sk}_j^*, b_j^*)$ *in T such that $b_j^* = 0$. The challenger also checks that S^* is not in U. If any of the checks fails, the challenger immediately aborts and outputs a random bit.*

    If the checks pass, the challenger chooses a random bit $b^* \in \{0, 1\}$ and replies with k_{b^*} where $k_0 \leftarrow$ $\mathsf{KeyGen}(S^*, \mathsf{sk}_1)$ *and k_1 is uniformly random.*
– *$\mathcal{A}$ produces a guess b' for b^*. The challenger outputs 1 iff $b' = b^*$.*

A Multiparty NIKE is adaptively secure if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function ϵ such that the challenger outputs 1 with probability at most $\frac{1}{2} + \epsilon$.

Other security notions. We can also consider multiparty NIKE with *unbounded honest users*, where the input 1^n is ignored in Pub, and there is no limit to the number of Register Honest User. We can similarly consider multiparty NIKE with *unbounded corruptions* where there is no limit to the number of Corrupt User queries, and *unbounded set size*, where there is no limit to the set size t that can be inputted to KeyGen or queried in Shared Key or Challenger queries.

We can also consider NIKE that is "secure with out X queries", which means that security holds against all adversaries that do not make any type X queries.

Common Reference String. We can also consider a crs model, where there is a setup algorithm $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^n, 1^c)$. Then Pub is changes to have the syntax $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Pub}(\mathsf{crs})$. In the adaptive security experiment, we have the challenger run $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^n, 1^c)$ and give crs to $\mathcal{A}$. It then uses the updated Pub algorithm when registering honest users.

2.2 Constrained PRFs

A special case of bit-fixing PRFs. Here, we define a type of bit-fixing PRF.

Definition 3 (1-Symbol-Fixing PRF, Syntax). *1-SF-PRF is a tuple* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Constr}, \mathsf{EvalC})$ *with the following syntax:*

- $\mathsf{Gen}(1^\lambda, 1^{|\Sigma|}, 1^\ell)$ *takes as input a security parameter λ, an alphabet size $|\Sigma|$, and an input length ℓ, all represented in unary. It outputs a key k.*
- $\mathsf{Eval}(k, x)$ *is the main evaluation algorithm, which is deterministic and takes as input a key k and $x \in \Sigma^\ell$, and outputs a string.*
- $\mathsf{Constr}(k, i, z)$ *is a potentially randomized algorithm that takes as input a key k, index $i \in [\ell]$, and symbol $z \in \Sigma$. It outputs a constrained key $k_{i,z}$.*
- $\mathsf{EvalC}(k_{i,z}, x)$ *takes as input a constrained key $k_{i,z}$ for an index/symbol pair (i, z), and an input x. It outputs a string. We have the correctness guarantee:*

$$\mathsf{EvalC}(k_{i,z}, x) = \begin{cases} \perp & \text{if } x_i \neq z \\ \mathsf{Eval}(k, x) & \text{if } x_i = z \end{cases}$$

Definition 4 (1-SF-PRF, Adaptive Security). *Consider the following experiment with an adversary $\mathcal{A}$:*

- *$\mathcal{A}$ on input 1^λ, produces $1^{|\Sigma|}, 1^\ell$. The challenger runs $k \leftarrow \mathsf{Gen}(1^\lambda, 1^{|\Sigma|}, 1^\ell)$. It returns nothing to $\mathcal{A}$.*
- *Then $\mathcal{A}$ can adaptively make the following types of queries:*
  - **Constrain.** *$\mathcal{A}$ sends i, z, and receives $k_{i,z} \leftarrow \mathsf{Constr}(k, i, z)$. The challenger records each (i, z) in a table C. There is no limit to the number of constrain queries.*
  - **Eval.** *$\mathcal{A}$ sends an input x, and receives $\mathsf{Eval}(k, x)$. The challenger records each x in a table E. There is no limit to the number of Eval queries.*
  - **Challenge.** *$\mathcal{A}$ can make a* single *challenge query on an input $x^* \in \Sigma^\ell$. The challenger flips a random bit $b \in \{0, 1\}$ and replies with $y^* = y_b$ where $y_0 = \mathsf{Eval}(k, x)$ and y_1 is sampled uniformly and independently.*

  If at any time, $x_i^* = z$ for some $(i, z) \in C$ or $x^* \in E$, the challenger immediately aborts and outputs a random bit.
- *The adversary outputs bit b'. The challenger outputs 1 if $b = b'$, 0 otherwise.*

A 1-SF-PRF is adaptively secure *if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function ϵ such that the challenger outputs 1 with probability at most $\frac{1}{2} + \epsilon$. It is* adaptively secure without Eval queries *if this holds for all $\mathcal{A}$ that make no Eval queries.*

A 1-SF-PRF scheme is said to be adaptively secure against unique-query adversaries *if the above holds for any adversary $\mathcal{A}$ that makes unique constrained key queries to the challenger.*

3 Enhancing Multi-party NIKE

Here give some compilers for multi-party NIKE, which allow for simplifying the task of designing new NIKE protocols built from iO. Our ultimate goal is to show that one can safely ignore Shared Key and Register Malicious User queries, and also employ a trusted setup. Our compilers then show how to lift such a scheme into one secure under all types of queries and without a trusted setup.

3.1 Achieving Adversarial Correctness

First, we convert any NIKE that is perfectly correct into one with adversarial correctness. While adversarial correctness is not a particular design goal in multiparty NIKE, this step will be needed in order to apply our later compilers.

Theorem 3. *Assume there exists a multi-party NIKE with perfect correctness, potentially in the crs model. Assume additionally there exists a NIZK. Then there exists a multi-party NIKE with both perfect and adversarial correctness in the crs model. If the perfectly correct scheme has unbounded honest users, corruptions, and/or set size, then so does the resulting adversarially correctscheme.*

Theorem 3 follows from a standard application of NIZKs, and is similar to a theorem used in the context of two-party NIKE by [HHK18]. The proof is given in the Full Version [KWZ22].

3.2 Removing the CRS

Next, we use iO to remove the common reference string (crs) from any multi-party NIKE. A side-effect of this transformation, however, is that we only achieve security without Register Malicious User queries.

Theorem 4. *Assuming there exists iO an adaptively secure multi-party NIKE in the common reference string (crs) model, then there also exists adaptively multi-party NIKE in the plain model that is secure without Register Malicious User queries. If the crs scheme has unbounded honest users, corruptions, and/or set size, or has perfect and/or adversarial correctness, or only has secure without X queries for some X, then the same is true of the resulting plain model scheme.*

Proof. Theorem 4 formalizes the ad hoc techniques for removing the CRS in iO-based constructions starting from Boneh and Zhandry [BZ14]. The proofs of the bounded/unbounded cases and perfect/adversarial correctness cases are essentially the same, so we focus on the case where everything is bounded. We will let $(\mathsf{Setup}, \mathsf{Pub}', \mathsf{KeyGen}')$ be a multi-party NIKE with setup.

Let F be a puncturable PRF. F can be constructed from any one-way function, which are in turn implied by any NIKE scheme. We construct a new mutliparty NIKE $(\mathsf{Pub}, \mathsf{KeyGen})$ without setup as follows:

- $\mathsf{Pub}(1^\lambda, 1^\ell, 1^n, 1^c)$: Run $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^n, 1^c)$. Sample a random PRF key k for F. Let $\mathsf{PKey}_k, \mathsf{SKey}_k$ be the programs in Figures 1 and 2, and let $\widehat{\mathsf{PKey}} = \mathsf{iO}(\mathsf{PKey}_k), \widehat{\mathsf{SKey}} = \mathsf{iO}(\mathsf{SKey}_k)$. $\mathsf{pk} = (\mathsf{crs}, \widehat{\mathsf{PKey}})$ and $\mathsf{sk} = \widehat{\mathsf{SKey}}$.
- $\mathsf{KeyGen}(S, \mathsf{sk}_i)$: Let $\mathsf{pk}^* \in S$ be the minimal $\mathsf{pk} \in S$ according to some ordering; we will call pk^* the distinguished public key.
  Write $\mathsf{pk}^* = (\mathsf{crs}^*, \widehat{\mathsf{PKey}}^*)$. Let S' be derived from S, where for each $\mathsf{pk} = (\mathsf{crs}, \widehat{\mathsf{PKey}}) \in S$, we include $\mathsf{pk}' = \widehat{\mathsf{PKey}}(\mathsf{crs}^*)$ in S'. Also let $\mathsf{sk}_i = \widehat{\mathsf{SKey}}_i$, and run $\mathsf{sk}'_i = \widehat{\mathsf{SKey}}_i(\mathsf{crs}^*)$. Then run and output $\mathsf{KeyGen}'(\mathsf{crs}, S', \mathsf{sk}'_i)$.

Fig. 1: The program PKey_k.

Inputs: crs
Constants: k

```
1. $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{Pub}'(\mathsf{crs};\ \mathsf{F}(k, \mathsf{crs}))$
2. Output $\mathsf{pk}'$
```

Fig. 2: The program SKey_k.

Inputs: crs
Constants: k

```
1. $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{Pub}'(\mathsf{crs};\ \mathsf{F}(k, \mathsf{crs}))$
2. Output $\mathsf{sk}'$
```

Correctness: Correctness follows from the correctness of the underlying scheme:

$$\begin{aligned}\mathsf{KeyGen}(S, \mathsf{sk}_i) &= \mathsf{KeyGen}'(\mathsf{crs}, S', \mathsf{sk}'_i) = \mathsf{KeyGen}'(\mathsf{crs}, S', \mathsf{sk}'_j) \\ &= \mathsf{KeyGen}(S, \mathsf{sk}_j)\end{aligned}$$

Security: Security is proved in the Full Version [KWZ22], following a careful application of iO techniques.

3.3 Adding Shared Key Queries

The final compiler generically convert a NIKE scheme whose security does *not* support shared key queries into one that does.

Theorem 5. *Assume there exists a multi-party NIKE with adversarial correctness and adaptive security without Shared Key or Register Malicious User queries. Then there exists a multi-party NIKE with adversarial correctness and adaptive security (*with *Shared Key and Register Malicious User queries). If the original scheme is also perfectly correct, then so is the resulting scheme. If the original scheme has unbounded honest users, corruptions, and/or set size, then so does the resulting scheme. The resulting scheme is in the CRS model if and only if the original scheme is.*

Note the requirement that the underlying NIKE protocol have adversarial correctness. The proof of Theorem 5 exploits the structure of multiparty NIKE, together with combinatorial tricks, to ensure that the reduction can answer all

Shared Key queries (even on sets involving malicious users) while not being able to answer the challenge query.

In slightly more detail, the rough idea is to randomly give the reduction some of the secret keys for users. We give the reduction enough secret keys so that with non-negligible probability it will be able to answer all shared key queries, while simultaneously being *unable* to answer the challenge query.

The main challenge is that shared key queries can be very "close" to the challenge query, potentially differing in only a single user. In order to be able to answer the shared key query but not the challenge query, we must give out the secret key for exactly the differing user, which we do not know in advance. In our solution, every user will actually contain many sub-users. The shared key for a group of users is then the shared key for some collection of the sub-users. The collections of sub-users will be chosen so that the collections for each group are "far" apart. The proof is given in the Full Version [KWZ22].

3.4 Putting It All Together

We can combine Theorems 3, 4, and 5 together, to get the following corollary:

Corollary 1. *Assume there exists iO and perfectly correct multi-party NIKE in the crs model with adaptive security without Shared Key or Register Malicious User queries. Then there exists perfectly correct (and also adversarially correct) multi-party NIKE in the plain model with adaptive security (under both Shared Key and Register Malicious User queries). If the original scheme has unbounded honest users, corruptions, and/or set size, then so does the resulting scheme.*

Corollary 1 shows that, for multiparty NIKE from iO, it suffices to work in the CRS model and ignore Shared Key and Register Malicious User queries.

4 The Equivalence of Multiparty NIKE and 1-SF-PRF

In this section, we show that NIKE is equivalent to a 1-SF-PRF. In the Full Version [KWZ22], we show that NIKE implies 1-SF-PRF, following a simple combinatorial construction. Here, we focus on the other direction.

4.1 From 1-SF-PRF to Special Constrained PRF

Here, we define an intermediate notion of constrained PRF, which enhances a 1-SF-PRF. The idea is that the symbol space Σ is now exponentially large. However, at the beginning a polynomial-sized set S is chosen, and a punctured key is revealed that allows for evaluating the PRF on any point *not* in S. The points in S then behave like the symbol space for a plain 1-SF-PRF, where it is possible to generate keys that fix any given position to some symbol in S.

Looking ahead to our NIKE construction, the set S will correspond to the public keys of the honest users of the system, while the rest of Σ will correspond to maliciously-generated keys. The abstraction of our special constrained PRF in this section is the missing link to formalize the connection between 1-SF-PRFs and NIKE as outlined in Section 1.

Definition 5 (Special Constrained PRF, Syntax). *SC-PRF is a tuple of algorithms* (Gen, Eval, Punc, EvalP, Constr, EvalC) *with the following syntax:*

- $\mathsf{Gen}(1^\lambda, |\Sigma|, 1^\ell, 1^n)$ *takes as input a security parameter* λ*, an alphabet size* $|\Sigma|$*, an input length* ℓ*, and a maximal set size* n*. Here,* $|\Sigma|$ *is represented in binary (thus allowing exponential-sized* Σ*), but everything else in unary.*
- $\mathsf{Eval}(k, x)$ *is the main evaluation algorithm, which is deterministic and takes as input a key* k *and* $x \in \Sigma^\ell$*, and outputs a string.*
- $\mathsf{Punc}(k, S)$ *is a randomized puncturing algorithm that takes as input a key* k *and set* $S \subseteq \Sigma$ *of size at most* n*. It outputs a punctured key* k_S*.*
- $\mathsf{EvalP}(k_S, x)$ *takes as input an* $x \in \Sigma^\ell$*, and outputs a value such that*

$$\mathsf{EvalP}(k_S, x) = \begin{cases} \perp & \text{if } x \in S^n \\ \mathsf{Eval}(k, x) & \text{if } x \notin S^n \end{cases}$$

- $\mathsf{Constr}(k, S, i, z)$ *is a potentially randomized constraining algorithm that takes as input a set* S*, a key* k*, an index* $i \in [\ell]$*, and symbol* $z \in S$*. It outputs a constrained key* $k_{S,i,z}$*.*
- $\mathsf{EvalC}(k_{S,i,z}, x)$ *takes as input a constrained key* $k_{S,i,z}$ *for a set/index/symbol triple* (S, i, z)*, and input* x*. It outputs a string. The correctness guarantee is:*

$$\mathsf{EvalC}(k_{S,i,z}, x) = \begin{cases} \perp & \text{if } x_i \neq z \\ \mathsf{Eval}(k, x) & \text{if } x_i = z \end{cases}$$

Definition 6 (Special Constrained PRF, Adaptive Security). *Consider the following experiment with an adversary* $\mathcal{A}$*:*

- $\mathcal{A}$ *on input* 1^λ*, outputs* $|\Sigma|, 1^\ell, 1^n$*, and set* S *of size at most* n*. The challenger runs* $k \leftarrow \mathsf{Gen}(1^\lambda, |\Sigma|, 1^\ell, 1^n)$ *and* $k_S \leftarrow \mathsf{Punc}(k, S)$*. It sends* k_S *to* $\mathcal{A}$*.*
- *Then* $\mathcal{A}$ *can adaptively make the following types of queries:*
  - **Constrain.** $\mathcal{A}$ *sends* i, z*, and receives* $k_{S,i,z} \leftarrow \mathsf{Constr}(k, S, i, z)$*. The challenger records each* (i, z) *in a table* C*.*
  - **Eval.** $\mathcal{A}$ *sends an input* x*, and receives* $\mathsf{Eval}(k, x)$*. The challenger records each* x *in a table* E*. There is no limit to the number of Eval queries.*
  - **Challenge.** $\mathcal{A}$ *can make a* single *challenge query on an input* $x^* \in S^\ell$*. The challenger flips a random bit* $b \in \{0,1\}$ *and replies with* $y^* = y_b$ *where* $y_0 = \mathsf{Eval}(k, x)$ *and* y_1 *is sampled uniformly and independently.*

  If at any time, $x_i^* = z$ *for some* $(i, z) \in C$ *or* $x^* \in E$*, the challenger immediately aborts and outputs a random bit.*
- *The adversary outputs bit* b'*. The challenger outputs 1 if* $b = b'$*, 0 otherwise.*

A Special Constrained PRF is adaptively secure if, for all PPT adversaries $\mathcal{A}$*, there exists a negligible function* ϵ *such that the challenger outputs 1 with probability at most* $\frac{1}{2} + \epsilon$*.*

Theorem 6. *If 1-SF-PRFs exist, then so do Special Constrained PRFs.*

The proof of Theorem 6 use purely combinatorial techniques. The idea is to set the symbol space Σ for the Special Constrained PRF to be codewords over the symbol space for the 1-SF-PRF, where the code is an error correcting code with certain properties. We defer the details to the Full Version [KWZ22].

4.2 From Special Constrained PRF to Multiparty NIKE with Setup

As a warm up, we construct multiparty NIKE in the common reference string model. We will need the following ingredients:

Definition 7. *A* single-point binding (SPB) signature *is a quadruple of algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{GenBind})$ *where* $\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}$ *satisfy the usual syntax of a signature scheme. Additionally, we have the following:*

- $(\mathsf{vk}, \sigma) \leftarrow \mathsf{GenBind}(1^\lambda, m)$ *takes as input a message m, and produces a verification key* vk *and signature σ.*
- *For any messages $m, m' \neq m$, with overwhelming probability over the choice of* $(\mathsf{vk}, \sigma) \leftarrow \mathsf{GenBind}(1^\lambda, m)$, $\mathsf{Ver}(\mathsf{vk}, m', \sigma') = \perp$ *for any σ'. That is, there is no message $m' \neq m$ where there is a valid signature of m' relative to* vk.
- *For any m,* $\mathsf{GenBind}(1^\lambda, m)$ *and* $(\mathsf{vk}, \mathsf{Sign}(\mathsf{sk}, m))$ *are indistinguishable, where* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. *Note that this property implies that* $\mathsf{Ver}(\mathsf{vk}, m, \sigma)$ *accepts, when* $(\mathsf{vk}, \sigma) \leftarrow \mathsf{GenBind}(1^\lambda, m)$.

Definition 8. *A* multi-point binding (MPB) hash function *is a triple of algorithms* $(\mathsf{Gen}, H, \mathsf{GenBind})$ *where:*

- $\mathsf{Gen}(1^\lambda, 1^n)$ *takes as input the security parameter λ, and an upper bound n on the number of inputs to bind. It produces a hashing key* hk.
- $H(\mathsf{hk}, x)$ *deterministically produces a hash h.*
- $\mathsf{GenBind}(1^\lambda, 1^n, S^*)$ *takes as input λ, n, and also a set S^* of inputs of size at most n. It produces a hashing key* hk *with the property that, with overwhelming probability over the choice of* $\mathsf{hk} \leftarrow \mathsf{GenBind}(1^\lambda, 1^n, S^*)$, *for any $x \in S^*$ and any $x' \neq x$ (which may or may not be in S^*),* $H(\mathsf{hk}, x) \neq H(\mathsf{hk}, x')$.
- *For any n and any set S^* of size at most n,* $(S^*, \mathsf{Gen}(1^\lambda, 1^n))$ *is computationally indistinguishable from* $(S^*, \mathsf{GenBind}(1^\lambda, 1^n, S^*))$.

A single-point *binding (SPB) hash function is as above, except we fix $n = 1$.*

We will rely on the following Lemmas of Guan, Wichs, and Zhandry [GWZ22]:

Lemma 1 ([GWZ22]). *Assuming one-way functions exist, so do single-point binding signatures.*

[GWZ22] show how to construct single-point binding hash functions. We adapt their construction to multi-point binding hashes:

Lemma 2. *Assuming one-way functions and iO exist, then so do multi-point binding hash functions.*

This lemma is proved in the Full Version [KWZ22], following almost identical ideas to the proof as [GWZ22].

We use multi-point binding hash functions in order to statistically bind to a set of inputs S^* with a hash that is much smaller than the inputs. Such hash functions will contain many collisions, but the point binding guarantee means that there is no collision with S^*. The SPB signature is used for similar reasons.

Our NIKE Construction. We don't bound collusion queries c (that is, the number of corruption queries), but bound the number of honest users, which implicitly bounds the collusion queries at n.

- $\mathsf{Setup}(1^\lambda, 1^\ell, 1^n)$: Run $\mathsf{hk} \leftarrow \mathsf{Gen}_{Hash}(1^\lambda, 1^\ell)$. Let $\mathcal{Y}$ be the range of H. Also sample $k \leftarrow \mathsf{Gen}_{PRF}(1^\lambda, |\mathcal{Y}|, 1^\ell, 1^n)$. Let $\mathsf{KGen}_{\mathsf{hk},k}$ be the program given in Figure 3, padded to the maximum size of the programs in Figures 3 and 4, and let $\widehat{\mathsf{KGen}} = \mathsf{iO}(\mathsf{KGen}_{\mathsf{hk},k})$. Output $\mathsf{crs} = \widehat{\mathsf{KGen}}$.
- $\mathsf{Pub}(\mathsf{crs})$: Sample a random message m and run $(\mathsf{vk}, \sigma) \leftarrow \mathsf{GenBind}_{Sig}(1^\lambda, m)$. Output $\mathsf{pk} = \mathsf{vk}$ and $\mathsf{sk} = (m, \sigma)$.
- $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{pk}_1, \ldots, \mathsf{pk}_\ell, i, \mathsf{sk}_i)$: assume the pk_j are sorted in order of increasing pk according to some fixed ordering; if the pk_j are not in order sort them, and change i accordingly. Write $\mathsf{crs} = \widehat{\mathsf{KGen}}$, $\mathsf{pk}_j = \mathsf{vk}_j$ and $\mathsf{sk}_i = (m_i, \sigma_i)$. Then output $\widehat{\mathsf{KGen}}(\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell, i, m_i, \sigma_i)$.

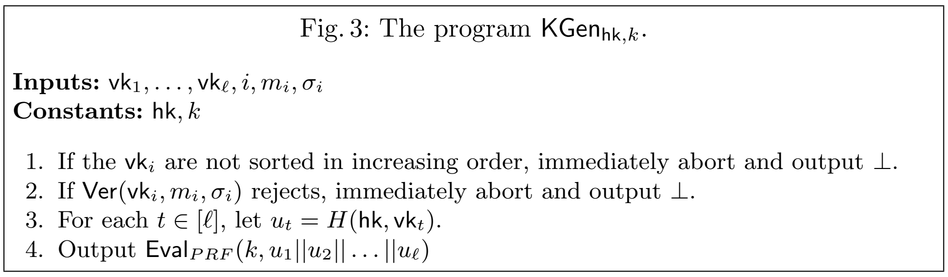
Fig. 3: The program $\mathsf{KGen}_{\mathsf{hk},k}$.

Inputs: $\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell, i, m_i, \sigma_i$
Constants: hk, k

```
1. If the $\mathsf{vk}_i$ are not sorted in increasing order, immediately abort and output $\perp$.
2. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
3. For each $t \in [\ell]$, let $u_t = H(\mathsf{hk}, \mathsf{vk}_t)$.
4. Output $\mathsf{Eval}_{PRF}(k, u_1||u_2||\ldots||u_\ell)$
```

Correctness. We need for any n and $i, j \in [\ell]$, that $\mathsf{KeyGen}(\mathsf{crs}, \{\mathsf{pk}_j\}_j, i, \mathsf{sk}_i)$ outputs a value equal to $\mathsf{KeyGen}(\mathsf{crs}, \{\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell\}, j, \mathsf{sk}_j)$ with overwhelming probability. This follows from the correctness of the signature scheme. With overwhelming probability, $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ $=\mathsf{Ver}(\mathsf{vk}_j, m_j, \sigma_j)$$=1$. Once the signature check passes, the outputs are identical.

Security. We will prove security via a sequence of hybrid experiments.

- $\mathsf{Game}_{\text{real}}$: This corresponds to the security game.

- **Setup Phase:**
  The challenger samples $\mathsf{hk} \leftarrow \mathsf{Gen}_{Hash}(1^\lambda, 1^\ell)$.
  Next, it samples $k \leftarrow \mathsf{Gen}_{PRF}(1^\lambda, |\mathcal{Y}|, 1^\ell, 1^n)$.
  The challenger computes $\widehat{\mathsf{KGen}} = \mathsf{iO}(\mathsf{KGen}_{\mathsf{hk},k})$ and sends $\mathsf{crs} = \widehat{\mathsf{KGen}}$ to the adversary. It also maintains a table T which is initially empty.
- **Pre-challenge Queries** The adversary makes the following queries:
  * *Honest user registration query*: For the i^{th} registration query, the challenger chooses m_i^*, computes $(\mathsf{vk}_i^*, \sigma_i^*) \leftarrow \mathsf{GenBind}_{Sig}(1^\lambda, m_i^*)$, sets vk_i^* as the public key and (m_i^*, σ_i^*) as the secret key. It adds $(\mathsf{vk}_i^*, (m_i^*, \sigma_i^*), 0)$ to the table T.
  * *Corruption query*: On receiving a corruption query for vk_i^*, the challenger sends (m_i^*, σ_i^*) to the adversary, and updates the i^{th} entry in T to $(\mathsf{vk}_i^*, (m_i^*, \sigma_i^*), 1)$.
  * *Registering Malicious user*: On receiving pk, the challenger adds $(\mathsf{pk}, \perp, 1)$ to T.
- **Challenge Query** On receiving $(\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell)$, the challenger checks the table T contains a $(\mathsf{vk}_i, (m_i, \sigma_i), 0)$ for each $i \in [\ell]$. If so, it chooses a random bit $b \leftarrow \{0,1\}$. If $b = 0$, it sends $\mathsf{Eval}_{PRF}(k, u_1 || \ldots || u_\ell)$, where $u_i = H(\mathsf{hk}, \mathsf{vk}_i)$. Else it sends a uniformly random string.
- **Post-challenge Queries** Same as pre-challenge queries.
- **Guess** Finally, the adversary sends its guess b', and wins if $b = b'$.

– Game_1: This experiment is identical to Game_0, except that the challenger chooses the n pairs $(\mathsf{vk}^*, \sigma^*)$ and m^* during setup. These are used to answer registration queries. The distribution of all components is identical to that in the previous experiment.

– Game_2: In this experiment, the challenger uses the honest users' verification keys to sample a hash key that is binding to all the verification keys. That is, it replaces $\mathsf{hk} \leftarrow \mathsf{Gen}_{Hash}(1^\lambda, 1^\ell)$ in Game_0 and Game_1 with $\mathsf{hk} \leftarrow \mathsf{GenBind}_{Hash}\left(1^\lambda, \{\mathsf{vk}_i^*\}_{i\in[n]}\right)$.

– Game_3: In this game, the challenger uses a different (but functionally identical) program ($\mathsf{KGenAlt}$, defined in Figure 4) for computing the CRS. The Setup phase is now the following, with the changes from Game_2 in yellow:
- **Setup Phase:**
  For $j \in [n]$, sample m_j^* and $(\mathsf{vk}_j^*, \sigma_j^*) \leftarrow \mathsf{GenBind}_{Sig}(1^\lambda, m_j^*)$.
  The challenger samples $\mathsf{hk} \leftarrow \mathsf{GenBind}_{Hash}\left(1^\lambda, \left\{\mathsf{vk}_j^*\right\}_{j\in[n]}\right)$.
  Next, it samples $k \leftarrow \mathsf{Gen}_{PRF}(1^\lambda, |\mathcal{Y}|, 1^\ell, 1^n)$.
  The challenger computes $u_j^* = H(\mathsf{hk}, \mathsf{vk}_j^*)$ and sets $S = \{u_j^*\}_{j\in[n]}$.
  It computes $K_S \leftarrow \mathsf{Punc}(k, S)$ and constrained keys
  $K_j^* = \left(\ \mathsf{Constr}(k, S, t, u_j^*)\ \right)_{t\in[\ell]}$. It sets $v_j^* = m_j^* \oplus K_j^*$ for each $j \in [n]$.
  The challenger computes $\widehat{\mathsf{KGenAlt}} = \mathsf{iO}\left(\mathsf{KGenAlt}_{\mathsf{hk},\{u_j^*, v_j^*, K_j^*\}, K_S}\right)$ and sends $\mathsf{crs} = \widehat{\mathsf{KGenAlt}}$ to the adversary. It also maintains a table T which is initially empty.

Fig. 4: The program $\mathsf{KGenAlt}_{\mathsf{hk},\{u_j^*,v_j^*,K_j^*\},K_S}$.

```
Inputs: $\mathsf{vk}_1,\dots,\mathsf{vk}_\ell, i, m_i, \sigma_i$
Constants: Hash key $\mathsf{hk}$
           $S = \{u_j^*\}_{j\in[n]}$
           $\{v_j^*\}_{j\in[n]}$
           Punctured key $K_S$

1. If the $\mathsf{vk}_i$ are not sorted in increasing order, immediately abort and output $\perp$.
2. If $\mathsf{Ver}(\mathsf{vk}_i, m_i, \sigma_i)$ rejects, immediately abort and output $\perp$.
3. For each $t \in [\ell]$, let $u_t = H(\mathsf{hk}, \mathsf{vk}_t)$.
4. If $u_i \in \{u_j^*\}_{j\in[n]}$, compute $K_j^* = \left(\ K_{j,t}^*\ \right)_{t\in[\ell]} = m_i \oplus v_i^*$,
   then output $\mathsf{EvalC}(K_{j,i}^*, u_1||u_2||\dots||u_\ell)$. Else output $\mathsf{EvalP}(K_S, u_1||u_2||\dots||u_\ell)$.
```

- Game_4: In this experiment, during setup, the challenger replaces $(\mathsf{vk}_j^*, \sigma_j^*) \leftarrow \mathsf{GenBind}_{Sig}(1^\lambda, m_j^*)$ from Game_3 with $(\mathsf{sk}_j^*, \mathsf{vk}_j^*) \leftarrow \mathsf{Gen}_{Sig}(1^\lambda)$ and $\sigma_j^* \leftarrow \mathsf{Sign}(\mathsf{sk}_j^*, m_j^*)$.
- Game_5: This game represents a syntactic change. Instead of choosing m_j^* first and then computing v_j^*, the challenger chooses uniformly random v_j^*, and sets $m_j^* = v_j^* \oplus K_j^*$. In terms of the adversary's view, this experiment is identical to the previous one.
  Now the constrained keys are not needed during setup, and can instead be generated adaptively during the corruption queries, which are now answered as follows (changes from Game_4 in yellow): On receiving a corruption query for vk_i^*, the challenger computes $K_i^* = \left(\ \mathsf{Constr}(k, S, t, u_j^*)\ \right)_{t\in[\ell]}$. It then computes $m_j^* = v_j^* \oplus K_j^*$ and sends (m_i^*, σ_i^*) to the adversary, and updates the i^{th} entry in T to $(\mathsf{vk}_i^*, (m_i^*, \sigma_i^*), 1)$.

In the Full Version [KWZ22], we analyse the adversary's advantage in each of these experiments, showing these games are computationally indistinguishable.

5 Construction of 1-SF-PRFs

The previous section worked to distill adaptively secure NIKE to the more basic primitive of constrained PRFs for one symbol fixing. While these transformations simplify the problem, the central barriers to proving adaptive security still remain. In this section we address these head on.

We review the main issues for adaptivity. Consider an adversary $\mathcal{A}$ that first makes several constrained key queries $(\mathsf{index}_1, \mathsf{sym}_1), \dots, (\mathsf{index}_Q, \mathsf{sym}_Q)$. Next the $\mathcal{A}$ submits a challenge input x^* such that $x_i^* \neq z$ for any pre-challenge key query (i, z) and receives back the challenge output from the challenger. Before submitting its guess, $\mathcal{A}$ will first perform some consistency checks on the constrained keys it received. For example, it can run the evaluation algorithm on multiple points that are valid for different sets of constrained keys and verify that it receives the same output from each. If not, it aborts and makes no guess.

Dealing with such an attacker is difficult for multiple reasons. First, a reduction cannot simply guess x^* or which index/symbol pairs will be queried without an exponential loss. Second, it cannot issue constrained keys that are deviate much from each other less this be detected by $\mathcal{A}$'s consistency checks.

We overcome these issues by having the challenger gradually issues constrained keys that deviate from a canonical PRF which is used to evaluate on the challenge input. However, we endeavor to keep all subsequent issued keys consistent with any introduced deviation so that this will avoid being detected.

Diving deeper our construction will use constrained keys which are obfuscated programs. Initially, the obfuscated program will simply check if an input x is consistent with the single symbol fixing of the key. If so, it evaluates the canonical PRF which is a Naor-Reingold style PRF.

The proof will begin by looking at the first key that is issued by the challenger for some query $(\mathsf{index}_1, \mathsf{sym}_1)$. For this key the obfuscated program will branch off and evaluate any inputs x where $x_{\mathsf{index}_1} = \mathsf{sym}_1$ in a different, but functionally equivalent way to the canonical PRF. By the security of iO this will not be detected. Moreover, this alternative evaluation for when $x_{\mathsf{index}_1} = \mathsf{sym}_1$ will be adopted by all further issued keys. Once this alternative pathway is set for all keys, we can change the evaluation on such inputs to be inconsistent with the canonical PRF, but mutually consistent with all issued keys. This follows from the DDH assumption. The proof can then proceed to the transforming the second issued key in a similar way such that there is a separate pathway for all inputs x where $x_{\mathsf{index}_2} = \mathsf{sym}_2$. The one exception is that the second and all future keys will give prioritization to the first established pathway whenever we have an input x where both $x_{\mathsf{index}_1} = \mathsf{sym}_1$ and $x_{\mathsf{index}_2} = \mathsf{sym}_2$.

The proof continues on in this way where each new key issued will establish an alternative evaluation which will be used except when it is pre-empted by an earlier established alternative. In this manner the constrained keys issued will always be mutually consistent on inputs, even while they gradually deviate from the canonical PRF. Finally, at the end of the proof all issued keys will use some alternative pathway for *all* evaluations. At this point we can use indistinguishability obfuscation again to remove information about the canonical PRF from the obfuscated programs since it is never used. With this information removed no attacker can distinguish a canonical PRF output from a random value.

We remark that in order to execute our proof strategy, our initial obfuscated program must be as large as any program used in the proof. In particular, it must be large enough to contain an alternative evaluation programming for all corrupted keys. Thus our constrained PRF keys must grow in size proportional to $\ell \cdot |\Sigma|$ and our resulting NIKE is parameterized for a set number of collusions.

5.1 Construction

- $\mathsf{Gen}(1^\lambda, \Sigma, 1^\ell)$: The key generation algorithm first runs $\mathcal{G}(1^\lambda)$ to compute $(p, \mathbb{G})$. Next, it chooses $v \leftarrow \mathbb{G}$, exponents $e_{j,w} \leftarrow \mathbb{Z}_p$ for each $j \in [\ell]$, $w \in \Sigma$. The PRF key K consists of $(v, \{e_{j,w}\})$.
- $\mathsf{Eval}(\mathsf{K}, x)$: Let $\mathsf{K} = (v, \{e_{j,w}\})$ and $x = (x_1, \ldots, x_\ell) \in \Sigma^\ell$. The PRF evaluation on input x is v^t, where $t = \left(\prod_{j \le n} e_{j,x_j}\right)$.
- $\mathsf{Constr}(\mathsf{K}, i, z)$: The constrained key is an obfuscation of the program $\mathsf{ConstrainedKey}_{\mathsf{K},i,z}$ (defined in Figure 5). The program is sufficiently padded to ensure that it is of the same size as the programs $\mathsf{ConstrainedKeyAlt}$, $\mathsf{ConstrainedKeyAlt}'$ (defined in Figure 6, 7) as well as an additional program that is used in the security proof. This additional program is specified in the Full Version [KWZ22].
  It outputs $\mathsf{K}_{i,z} \leftarrow \mathsf{iO}(1^\lambda, \mathsf{ConstrainedKey}_{\mathsf{K},i,z})$ as the constrained key.

$\mathsf{ConstrainedKey}_{\mathsf{K},i,z}$

Input: $x = (x_1, \ldots, x_\ell) \in \Sigma^\ell$

Constants:	Group element v
	Exponents $\{e_{j,w}\}_{j \in [\ell], w \in \Sigma}$
	Constraining index/symbol $i \in [\ell], z \in \Sigma$

1. If $x_i \neq z$ output $\perp$.
2. Compute $t = \left(\prod_{j \le \ell} e_{j,x_j}\right)$.
3. Output v^t.

Fig. 5: Program ConstrainedKey

- $\mathsf{EvalC}(\mathsf{K}_{i,z}, x)$: The constrained key $\mathsf{K}_{i,z}$ is an obfuscated program. The evaluation algorithm outputs $\mathsf{K}_{i,z}(x)$.

5.2 Security Proof

We will prove that the above construction satisfies security against unique-query adversaries, via a sequence of hybrid games. The first game corresponds to the original security game (security against *unique query adversary*). Next, we define Q hybrid games $\{\mathsf{Game}_y\}_{y \in [Q]}$, where Q is a bound on the total number of constrained key queries by the adversary.

- $\mathsf{Game}_{\mathrm{real}}$:
  - **Setup Phase:** The challenger chooses $v \leftarrow \mathbb{G}$, $e_{j,w} \leftarrow \mathbb{Z}_p$ for each $j \in [\ell], w \in \Sigma$. Let $\mathsf{K} = (v, (e_{j,w})_{j,w})$.
    The challenger also maintains an ordered list L of $(\mathsf{index}, \mathsf{sym})$ pairs. This list is initially empty, and for each (new) query, the challenger adds a tuple to L.

- **Pre-challenge queries:** Next, the challenger receives pre-challenge constrained key queries. Let $(\mathsf{index}_j, \mathsf{sym}_j)$ be the j^{th} constrained key query. The challenger adds $(\mathsf{index}_j, \mathsf{sym}_j)$ to L.
  The challenger computes the constrained key
  $\mathsf{K}_j \leftarrow \mathsf{iO}(1^\lambda, \mathsf{ConstrainedKey}_{\mathsf{K},\mathsf{index}_j,\mathsf{sym}_j})$ and sends K_j to the adversary.
- **Challenge Phase:** Next, the adversary sends a challenge x^* such that $x_i^* \neq z$ for any pre-challenge key query (i, z). The challenger chooses $b \leftarrow \{0, 1\}$. If $b = 0$, the challenger computes $t = \prod_i e_{i,x_i^*}$ and sends v^t. If $b = 1$, the challenger sends a uniformly random group element in $\mathbb{G}$.
- **Post-challenge queries:** The post-challenge queries are handled similar to the pre-challenge queries.
- **Guess:** Finally, the adversary sends the guess b' and wins if $b = b'$.

– Game_y: In this game, the challenger uses an altered program for the first y constrained keys. It makes the following changes to $\mathsf{Game}_{\text{real}}$:
  - **Setup Phase:** The challenger additionally samples $h_j \leftarrow \mathbb{G}$ for all $j \in [y]$. Let $H = (h_j)_{j\in[y]}$.
  - **Pre-challenge queries:** Let $(\mathsf{index}_j, \mathsf{sym}_j)$ be the j^{th} constrained key query. The challenger adds $(\mathsf{index}_j, \mathsf{sym}_j)$ to L. Let $s = \min(y, j)$, and let L_s (resp. H_s) denote the first s entries in L (resp. H). The challenger computes the key $\mathsf{K}_j \leftarrow \mathsf{iO}(1^\lambda, \mathsf{ConstrainedKeyAlt}_{s,L_s,H_s,v,(e_{j,w}),\mathsf{index}_j,\mathsf{sym}_j})$ and sends K_j to the adversary.

$\mathsf{ConstrainedKeyAlt}_{s,L_s,H_s,v,(e_{j,w}),i,z}$

Input: $x = (x_1, \ldots, x_\ell) \in \Sigma^\ell$

Constants: $s \in \ell \cdot |\Sigma|$
List $L_s = \left((\mathsf{index}_j, \mathsf{sym}_j)\right)_{j\in[s]}$
$H_s = (h_j)_{j\in[s]}$
Group element v,
Exponents $(e_{j,w})_{j,w}$,
Constraining index/symbol $i \in [\ell], z \in \Sigma$

```
1. If $x_i \neq z$ output $\perp$.
2. Compute $t = \left(\prod_{j\leq\ell} e_{j,x_j}\right)$.
3. Find the smallest $j \in [s]$ such that $x_{\mathsf{index}_j} = \mathsf{sym}_j$.
   (a) If such $j$ exists, then output $h_j^t$.
   (b) Else output $v^t$.
```

Fig. 6: Program $\mathsf{ConstrainedKeyAlt}$

Analysis We will now show that $\mathsf{Game}_{\mathrm{real}}$ and Game_y are computationally indistinguishable for all $y \in [Q]$. Finally, we will show that no polynomial time adversary has non-negligible advantage in Game_Q, showing that the scheme is secure against *unique query adversaries*. For any adversary $\mathcal{A}$, let $\mathsf{adv}_{\mathcal{A},\mathrm{real}}$ denote $\mathcal{A}$'s advantage in $\mathsf{Game}_{\mathrm{real}}$, and let $\mathsf{adv}_{\mathcal{A},y}$ denote $\mathcal{A}$'s advantage in Game_y.

Lemma 3. *Assuming* iO *is secure, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for all* λ, $|\mathsf{adv}_{\mathcal{A},\mathrm{real}} - \mathsf{adv}_{\mathcal{A},0}| \le \mathsf{negl}(\lambda)$.

Proof. For $y = 0$, the lists L_y and H_y are empty, and as a result, the programs are functionally identical. On any input x, both programs output v^t. Therefore, their obfuscations are computationally indistinguishable.

Lemma 4. *Fix any* $y \in [Q]$*. Assuming* DDH *and security of* iO*, for any PPT adversary* $\mathcal{A}$ *making at most* Q *queries, there exists a negligible function* negl *such that for all* λ, $|\mathsf{adv}_{\mathcal{A},y} - \mathsf{adv}_{\mathcal{A},y+1}| \le \mathsf{negl}(\lambda)$.

Proof. We will define hybrid games to show that Game_y and Game_{y+1} are computationally indistinguishable. The main difference in the two games is with regard to the last $Q - y$ constrained key queries. Note that the first y constrained keys are identical in both experiments. For each of the last $Q - y$ constrained keys, if (i, z) is the constrained key query, then the adversary receives an obfuscation of

- $P_{y,i,z} \equiv \mathsf{ConstrainedKeyAlt}_{y,L_y,H_y,v,(e_{j,w}),i,z}$ in Game_y,
- $P_{y+1,i,z} \equiv \mathsf{ConstrainedKeyAlt}_{y+1,L_{y+1},H_{y+1},v,(e_{j,w}),i,z}$ in Game_{y+1}

Note that the programs $P_{y,i,z}$ and $P_{y+1,i,z}$ only differ on inputs x where $x_i = z$ (in one case the output is v^t, while in the other case the output is h^t_{y+1}). We will prove that these two hybrid games are indistinguishable, using a sequence of sub-hybrid experiments defined below.

- $\mathsf{Game}_{y,a}$: This security game is similar to Game_y, except that the challenger guesses the $(y+1)^{\mathrm{th}}$ query in the setup phase.
  - **Setup Phase:** The challenger chooses $v \leftarrow \mathbb{G}$, $h_j \leftarrow \mathbb{G}$ for all $j \in [y]$ and $e_{j,w} \leftarrow \mathbb{Z}_p$ for all $j \in [\ell], w \in \Sigma$. Let $H_y = (h_j)_{j\in[y]}$.
    The challenger maintains an ordered list L of $(\mathsf{index}, \mathsf{sym})$ pairs which is initially empty.
    The challenger also chooses $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1}) \leftarrow [\ell] \times \Sigma$.
  - **Pre-challenge queries:** Next, the challenger receives pre-challenge constrained key queries. Let $(\mathsf{index}_q, \mathsf{sym}_q)$ be the q^{th} constrained key query. The challenger adds $(\mathsf{index}_q, \mathsf{sym}_q)$ to L.
    If the $(y+1)^{\mathrm{th}}$ query is not $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$, then the challenger aborts. The adversary wins with probability 1/2.
    Let $s = min(y, q)$, and let L_s denote the first s entries in L. The challenger computes the constrained key
    $\mathsf{K}_q \leftarrow \mathsf{iO}(1^\lambda, \mathsf{ConstrainedKeyAlt}_{s,L_s,H_s,v,(e_{j,w}),\mathsf{index}_q,\mathsf{sym}_q})$ and sends K_q to the adversary.

- **Challenge Phase:** Next, the adversary sends a challenge x^* such that $x_i^* \neq z$ for any pre-challenge key query (i, z). The challenger chooses $b \leftarrow \{0,1\}$. If $b = 0$, the challenger computes $t = \prod_i e_{i,x_i^*}$ and sends v^t. If $b = 1$, the challenger sends a uniformly random group element in $\mathbb{G}$.
- **Post-challenge queries:** The post-challenge queries are handled similar to the pre-challenge queries.
- **Guess:** Finally, the adversary sends the guess b' and wins if $b = b'$.

– $\mathsf{Game}_{y,b}$: This security game is similar to $\mathsf{Game}_{y,a}$, except that the challenger chooses the h_j constants and one of the $e_{j,w}$ exponents differently. However, the distribution of these components is identical to their distribution in the previous game.

- **Setup Phase:** The challenger chooses $g \leftarrow \mathbb{G}$, $b \leftarrow \mathbb{Z}_p$, $c_j \leftarrow \mathbb{Z}_p$ for all $j \in [y]$. It sets $v = g^b$, $h_j = g^{c_j}$.
  The challenger maintains an ordered list L of $(\mathsf{index}, \mathsf{sym})$ pairs which is initially empty.
  The challenger also chooses $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1}) \leftarrow [\ell] \times \Sigma$.
  It chooses $e_{j,w} \leftarrow \mathbb{Z}_p$ for all $j \in [n], w \in \Sigma$, $(j, w) \neq (\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$.
  It chooses $a \leftarrow \mathbb{Z}_p$ and sets $e_{\mathsf{index}_{y+1},\mathsf{sym}_{y+1}} = a$, $A = g^a$ and $T = v^a$.
  Note that the terms A and T are not used in this experiment; they will be used in some of the following hybrid experiments. Let $H_y = (h_j)_{j\in[y]}$.

– $\mathsf{Game}_{y,c}$: In this security game, the challenger computes the constrained keys differently. Instead of sending an obfuscation of $\mathsf{ConstrainedKeyAlt}$ (with appropriate hardwired constants), the challenger computes an obfuscation of $\mathsf{ConstrainedKeyAlt}'$ (with appropriate hardwired constants). The program $\mathsf{ConstrainedKeyAlt}'$ is defined in Figure 7, and is padded to be of the same size as $\mathsf{ConstrainedKey}$, $\mathsf{ConstrainedKeyAlt}$ and $\mathsf{ConstrainedKeyEnd}$.
The main difference is that $\mathsf{ConstrainedKeyAlt}'$ does not contain the exponent $e_{\mathsf{index}_{y+1},\mathsf{sym}_{y+1}}$ (recall $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$ is the $(y+1)^{\text{th}}$ constrained key query, and the challenger guesses this query during setup). Instead, the program contains $g^{e_{\mathsf{index}_{j+1},\mathsf{sym}_{j+1}}}$ and $v^{e_{\mathsf{index}_{j+1},\mathsf{sym}_{j+1}}}$. As a result, the final output is computed differently (although the outputs are identical).
We will show that the two programs are functionally identical, and therefore their obfuscations are computationally indistinguishable.

- **Pre-challenge queries:** Let $(\mathsf{index}_q, \mathsf{sym}_q)$ be the q^{th} constrained key query. The challenger adds $(\mathsf{index}_q, \mathsf{sym}_q)$ to L. Let L_j denote the first j entries in L.
  If $q \leq y$, the challenger computes
  $\mathsf{K}_q \leftarrow \mathsf{iO}(1^\lambda, \mathsf{ConstrainedKeyAlt}_{q,L_q,H_q,v,(e_{j,w}),\mathsf{index}_q,\mathsf{sym}_q})$ and sends K_q to the adversary.
  If the $(y+1)^{\text{th}}$ query is not $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$,[4] then the challenger aborts. The adversary wins with probability $1/2$.

[4] Recall $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$ is chosen during the setup phase.

If $q > y$, the challenger sends an obfuscation of the program:
$\mathsf{ConstrainedKeyAlt}'_{y,L_y,(\mathsf{index}_{y+1},\mathsf{sym}_{y+1}),\{c_y\},g,v,B,T,(e_{j,w})_{(j,w)},\mathsf{index}_q,\mathsf{sym}_q}$

```
ConstrainedKeyAlt'

Input: $x = (x_1, \dots, x_\ell) \in \Sigma^\ell$

  Constants:  $y \in [\,\ell \cdot |\Sigma|\,]$
              List of first $y$ queries $L_y = \big((\mathsf{index}_j, \mathsf{sym}_j)\big)_{j\in[y]}$
              $(y+1)^{\text{th}}$ query $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$
              exponents for computing $(h_j)_j : (c_j)_{j\in[y]}$
              Group elements $g, v, A, T$
              PRF eval exponents $= (e_{j,w})_{(j,w)\neq(\mathsf{index}_{y+1},\mathsf{sym}_{y+1})}$
              Constraining index/symbol $i \in [\ell], z \in \Sigma$

  1. If $x_i \neq z$ output $\perp$.
  2. Compute $t$ as follows:
     (a) If $x_{\mathsf{index}_{y+1}} = \mathsf{sym}_{y+1}$ then set $t = \left(\prod_{j\neq \mathsf{index}_{y+1}} e_{j,x_j}\right)$
     (b) Else $t = \left(\prod_j e_{j,x_j}\right)$
  3. Find the smallest $j \in [y]$ such that $x_{\mathsf{index}_j} = \mathsf{sym}_j$.
     (a) If such $j$ exists and $x_{\mathsf{index}_{y+1}} = \mathsf{sym}_{y+1}$ then output $(A)^{t\cdot c_j}$
     (b) If such $j$ exists and $x_{\mathsf{index}_{y+1}} \neq \mathsf{sym}_{y+1}$ then output $\left(g_j^c\right)^t$
     (c) Else if no such $j$ exists and $x_{\mathsf{index}_{y+1}} = \mathsf{sym}_{y+1}$ output $(T)^t$.
     (d) Else if no such $j$ exists and $x_{\mathsf{index}_{y+1}} \neq \mathsf{sym}_{y+1}$ output $v^t$.
```

Fig. 7: Program $\mathsf{ConstrainedKeyAlt}'$

- $\mathsf{Game}_{y,d}$: In this security game, the challenger sets T to be a uniformly random element in $\mathbb{G}$ instead of $T = v^a$.
- $\mathsf{Game}_{y,e}$: This security game represents a syntactic change. We choose $h_{j+1} \leftarrow \mathbb{G}$ and set $T = h_{j+1}^a$. The element h_{j+1} is not used anywhere else.
- $\mathsf{Game}_{y,f}$: In this experiment, the challenger uses $\mathsf{ConstrainedKeyAlt}$ for the last $Q - y$ constrained key queries. On receiving query (i, z), the challenger sends an obfuscation of $\mathsf{ConstrainedKeyAlt}_{y+1,L_{y+1},H_{y+1},v,(e_{k,w}),i,z}$. Here L_{y+1} and H_{y+1} are defined as in $\mathsf{Game}_{y,e}$.
- $\mathsf{Game}_{y,g}$: This security game is identical to $\mathsf{Game}_{y,f}$, and the changes in this game are syntactic. Instead of sampling exponents c_j and setting $h_j = g^{c_j}$, the challenger chooses $h_j \leftarrow \mathbb{G}$. Similarly, the challenger samples $v \leftarrow \mathbb{G}$, and samples all the exponents $e_{j,w} \leftarrow \mathbb{Z}_p$. Note that this experiment is identical to Game_{y+1}, except that the challenger guesses $(\mathsf{index}_{y+1}, \mathsf{sym}_{y+1})$ in the setup phase.

Claim 1 *For any $y \in [Q]$, and any adversary $\mathcal{A}$ making at most Q constrained key queries,* $|\mathsf{adv}_{\mathcal{A},y} - \mathsf{adv}_{\mathcal{A},y+1}| = \frac{1}{\ell\cdot|\Sigma|}\left(|\mathsf{adv}_{\mathcal{A},y,a} - \mathsf{adv}_{\mathcal{A},y,g}|\right)$.

Proof. Note that the only difference between $\mathsf{Game}_{y,a}$ and Game_y is that the challenger guesses the $(y+1)^{\text{th}}$ constrained key query in the setup phase. Similarly, the only difference between $\mathsf{Game}_{y,g}$ and Game_{y+1} is that the challenger guesses the $(y+1)^{\text{th}}$ constrained key query. This guess is correct with probability $1/(\ell \cdot |\Sigma|)$, and therefore $|\mathsf{adv}_{\mathcal{A},y} - \mathsf{adv}_{\mathcal{A},y+1}| = \frac{1}{\ell \cdot |\Sigma|}\left(|\mathsf{adv}_{\mathcal{A},y,a} - \mathsf{adv}_{\mathcal{A},y,g}|\right)$.

Therefore, it suffices to show that $\mathsf{Game}_{y,a}, \ldots, \mathsf{Game}_{y,g}$ are computationally indistinguishable. This is proved in the Full Version [KWZ22]. Proving the indistinguishability of these hybrids completes the proof of Lemma 4.

Acknowledgements

We thank Rachit Garg and George Lu for helpful feedback on an earlier draft of our work.

References

AJ15. Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 308–326. Springer, Heidelberg, August 2015.

AMN+18. Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Constrained PRFs for NC^1 in traditional groups. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 543–574. Springer, Heidelberg, August 2018.

BFP+15. Abhishek Banerjee, Georg Fuchsbauer, Chris Peikert, Krzysztof Pietrzak, and Sophie Stevens. Key-homomorphic constrained pseudorandom functions. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 31–60. Springer, Heidelberg, March 2015.

BGI14. Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

BGK+18. Dan Boneh, Darren B. Glass, Daniel Krashen, Kristin E. Lauter, Shahed Sharif, Alice Silverberg, Mehdi Tibouchi, and Mark Zhandry. Multiparty non-interactive key exchange and more from isogenies on elliptic curves. *Journal of Mathematical Cryptology*, 14:5 – 14, 2018.

BJLS16. Christoph Bader, Tibor Jager, Yong Li, and Sven Schäge. On the impossibility of tight cryptographic reductions. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 273–304. Springer, Heidelberg, May 2016.

BS02. Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90, 2002.

BTVW17. Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 264–302. Springer, Heidelberg, November 2017.

BV15a. Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *56th FOCS*, pages 171–190. IEEE Computer Society Press, October 2015.

BV15b. Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 1–30. Springer, Heidelberg, March 2015.

BW13. Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

BZ14. Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 480–499. Springer, Heidelberg, August 2014.

CC17. Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 446–476. Springer, Heidelberg, April / May 2017.

CFN94. Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 257–270. Springer, Heidelberg, August 1994.

CKS08. David Cash, Eike Kiltz, and Victor Shoup. The twin Diffie-Hellman problem and applications. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 127–145. Springer, Heidelberg, April 2008.

CM14. Melissa Chase and Sarah Meiklejohn. Déjà Q: Using dual systems to revisit q-type assumptions. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 622–639. Springer, Heidelberg, May 2014.

CRV16. Nishanth Chandran, Srinivasan Raghuraman, and Dhinakaran Vinayagamurthy. Reducing depth in constrained PRFs: From bit-fixing to $\mathbf{NC}^1$. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 359–385. Springer, Heidelberg, March 2016.

DH76. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

DKN$^+$20. Alex Davidson, Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Adaptively secure constrained pseudorandom functions in the standard model. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 559–589. Springer, Heidelberg, August 2020.

DKW16. Apoorvaa Deshpande, Venkata Koppula, and Brent Waters. Constrained pseudorandom functions for unconstrained inputs. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 124–153. Springer, Heidelberg, May 2016.

DKXY02. Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Key-insulated public key cryptosystems. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 65–82. Springer, Heidelberg, April / May 2002.

FHKP13. Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 254–271. Springer, Heidelberg, February / March 2013.

FKPR14. Georg Fuchsbauer, Momchil Konstantinov, Krzysztof Pietrzak, and Vanishree Rao. Adaptive security of constrained PRFs. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 82–101. Springer, Heidelberg, December 2014.

FN94. Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 480–491. Springer, Heidelberg, August 1994.

GGH13. Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, Heidelberg, May 2013.

GPSZ17. Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 156–181. Springer, Heidelberg, April / May 2017.

GVW12. Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 162–179. Springer, Heidelberg, August 2012.

GWZ22. Jiaxin Guan, Daniel Wichs, and Mark Zhandry. Incompressible cryptography. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 700–730. Springer, Heidelberg, May / June 2022.

HHK18. Julia Hesse, Dennis Hofheinz, and Lisa Kohl. On tightly secure non-interactive key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 65–94. Springer, Heidelberg, August 2018.

HHKL21. Julia Hesse, Dennis Hofheinz, Lisa Kohl, and Roman Langrehr. Towards tight adaptive security of non-interactive key exchange. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 286–316. Springer, Heidelberg, November 2021.

HJK$^+$16. Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal samplers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 715–744. Springer, Heidelberg, December 2016.

HKKW14. Dennis Hofheinz, Akshay Kamath, Venkata Koppula, and Brent Waters. Adaptively secure constrained pseudorandom functions. Cryptology ePrint Archive, Report 2014/720, 2014. https://eprint.iacr.org/2014/720.

HKW15. Susan Hohenberger, Venkata Koppula, and Brent Waters. Adaptively secure puncturable pseudorandom functions in the standard model. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 79–102. Springer, Heidelberg, November / December 2015.

Hof14. Dennis Hofheinz. Fully secure constrained pseudorandom functions using random oracles. Cryptology ePrint Archive, Report 2014/372, 2014. https://eprint.iacr.org/2014/372.

Jou00. Antoine Joux. A one round protocol for tripartite diffie–hellman. In Wieb Bosma, editor, *Algorithmic Number Theory*, pages 385–393, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

KPTZ13. Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

KRS15. Dakshita Khurana, Vanishree Rao, and Amit Sahai. Multi-party key exchange for unbounded parties from indistinguishability obfuscation. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 52–75. Springer, Heidelberg, November / December 2015.

KWZ22. Venkata Koppula, Brent Waters, and Mark Zhandry. Adaptive multiparty nike (full version), 2022.

LOS$^+$10. Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 62–91. Springer, Heidelberg, May / June 2010.

LW14. Allison B. Lewko and Brent Waters. Why proving HIBE systems secure is difficult. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 58–76. Springer, Heidelberg, May 2014.

MZ17. Fermi Ma and Mark Zhandry. Encryptor combiners: A unified approach to multiparty NIKE, (H)IBE, and broadcast encryption. Cryptology ePrint Archive, Report 2017/152, 2017. https://eprint.iacr.org/2017/152.

MZ18. Fermi Ma and Mark Zhandry. The MMap strikes back: Obfuscation and new multilinear maps immune to CLT13 zeroizing attacks. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 513–543. Springer, Heidelberg, November 2018.

NR97. Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997.

Rao14. Vanishree Rao. Adaptive multiparty non-interactive key exchange without setup in the standard model. Cryptology ePrint Archive, Report 2014/910, 2014. https://eprint.iacr.org/2014/910.

STW96. Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman key distribution extended to group communication. In Li Gong and Jacques Stern, editors, *ACM CCS 96*, pages 31–37. ACM Press, March 1996.

SW14. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484. ACM, 2014.

Wat09. Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 619–636. Springer, Heidelberg, August 2009.