How to Avoid Obfuscation Using Witness PRFs

Mark Zhandry
Stanford University, USA
mzhandry@stanford.edu

April 29, 2014

Abstract

Recently, program obfuscation has proven to be an extremely powerful tool and has been used to construct a variety of cryptographic primitives with amazing properties. However, current candidate obfuscators are far from practical and rely on unnatural hardness assumptions about multilinear maps. In this work, we bring several applications of obfuscation closer to practice by showing that a weaker primitive called witness pseudorandom functions (witness PRFs) suffices. Applications include multiparty key exchange without trusted setup, polynomially-many hardcore bits for any one-way function, and more. We then show how to instantiate witness PRFs from multilinear maps. Our witness PRFs are simpler and more efficient than current obfuscation candidates, and involve very natural hardness assumptions about the underlying maps.

1 Introduction

The goal of program obfuscation in cryptography is to scramble a program with the intention of hiding embedded secrets. Recently, Garg et al. [GGH+13b] gave the first candidate construction of a program obfuscator, which has sparked a flurry of research showing many exciting uses of obfuscation. Such uses include functional encryption [GGH+13b], short signatures and deniable encryption [SW14], multiparty key exchange and traitor tracing [BZ13], and much more [HSW14, GGHR14, BCP14, ABG+13, PPS13, KNY14].

While these results are exciting, instantiating these schemes with current candidate obfuscators [GGH+13b, BR13, BGK+14, PST13, AGIS14] has several drawbacks:

- First, these obfuscators only build obfuscation for *formulas*. Getting obfuscation for all circuits currently requires an expensive boosting step involving obfuscating the decryption algorithm for a fully homomorphic encryption scheme.

- Second, all of these constructions first convert the formula into a branching program that is either very long (in the case of [GGH+13b, BR13, BGK+14, PST13]) or very wide (in the case of [AGIS14]). Then, the branching program is encoded in a multilinear map. Long branching programs require a high level of multilinearity, and long or wide programs both require many group elements.

1.1 Our Results

In this work, we show that for several applications of obfuscation, a weaker primitive we call *witness pseudorandom functions* (witness PRFs) actually suffices. Informally, a witness PRF for an NP language L is a PRF F such that anyone with a valid witness that $x \in L$ can compute $\mathsf{F}(x)$, but for all $x \notin L$, $\mathsf{F}(x)$ is computationally hidden. More precisely, a witness PRF consists of the following three algorithms:

- $\mathsf{Gen}(\lambda, L, n)$ takes as input (a description of) an NP language L and instance length n, and outputs a secret function key fk and public evaluation key ek.
- $\mathsf{F}(\mathsf{fk}, x)$ takes as input the function key fk, an instance $x \in \{0,1\}^n$, and produces an output y
- $\mathsf{Eval}(\mathsf{ek}, x, w)$ takes the evaluation key ek, and instance x, and a witness w for x, and outputs $\mathsf{F}(\mathsf{fk}, x)$ if w is a valid witness, $\perp$ otherwise.

For security, we require that for any $x \in \{0,1\}^n \setminus L$, $\mathsf{F}(\mathsf{fk}, x)$ is pseudorandom, even given ek and polynomially many PRF queries to $\mathsf{F}(\mathsf{fk}, \cdot)$.

Witness PRFs are closely related to the concept of smooth projective hash functions (a comparison is given in Section 1.5), and can be seen as a generalization of constrained PRFs [BW13, KPTZ13, BGI13] to arbitrary NP languages. We first show how to replace obfuscation with witness PRFs for certain applications. We then show how to build witness PRFs from multilinear maps. Our witness PRFs are more efficient than current obfuscation candidates, and rely on very natural assumptions about the underlying maps. Below, we list our main results:

- We show how to realize the following primitives from witness PRFs
  - **Multiparty non-interactive key exchange without trusted setup**. The first such scheme is due to Boneh and Zhandry [BZ13], which is built from indistinguishability obfuscation (iO) and pseudorandom generators (PRGs). We give a closely related construction, where the obfuscator is replaced with a witness PRF, and prove that security still holds.
  - **Poly-many hardcore bits**. Bellare, Stepanovs, and Tessaro[BST13] construct a hardcore function of arbitrary output size for any one-way function. They require differing inputs obfuscation[BGI+01, BCP14, ABG+13], which is a form of knowledge assumption for obfuscators. We show how to replace the obfuscator with a witness PRFs that satisfies an extractability notion of security.
  - **Reusable Witness Encryption**. Garg, Gentry, Sahai, and Waters [GGSW13] define and build the first witness encryption scheme from multilinear maps. Later, Garg et al. [GGH+13b] show that indistinguishability obfuscation implies witness encryption. We show that witness PRFs are actually sufficient. We also define a notion of reusability for witness encryption, and give the first construction satisfying this notion.
  - **Rudich Secret Sharing for mNP**. Rudich secret sharing is a generalization of secret sharing to the case where the allowed sets are instances of a monotone NP (mNP) language, and an allowed set of shares plus the corresponding witness are sufficient for learning the secret. Komargodski, Naor, and Yogev [KNY14] give the first construction for all of mNP using iO. We give a simplification that uses only witness PRFs, and moreover is reusable.

– **Fully distributed broadcast encryption**. Boneh and Zhandry [BZ13] observe that certain families of key exchange protocols give rise to distributed broadcast encryption, where users generate their own secret keys. However, the notion has some limitations, which we discuss. We put forward the notion of *fully distributed* broadcast encryption which sidesteps these limitations, and give a construction where secret keys, public keys, and ciphertexts are short.

- Next, we show how to build witness PRFs from multilinear maps. We first define an intermediate notion of a subset-sum encoding, and construct such encodings from multilinear maps. We then show that subset-sum encodings imply witness PRFs.

1.2 Secure Subset-Sum Encodings

As a first step to building witness PRFs, we construct a primitive called a subset-sum encoding. Roughly, such an encoding corresponds to a set S of n integers, and consists of a secret encoding function, which maps integers y into encodings $\hat{y}$. Additionally, there is a public evaluation function which takes as input a subset $T \subseteq S$, and can compute the encoding $\hat{y}$ of the sum of the elements in T: $y = \sum_{i \in T} i$. For security, we ask that for any y that does not correspond to a subset-sum of elements of S, the encoding $\hat{y}$ is indistinguishable from a random element.

We provide a simple secure subset-sum encoding from asymmetric multilinear maps. Recall that an asymmetric multilinear map [BS03] consists of a sequence of "source" groups $\mathbb{G}_1, \dots, \mathbb{G}_n$, a target group $\mathbb{G}_T$, all of prime order p, along with generators $g_1, \dots, g_n, g_T$. There is also a multilinear operation $e : \mathbb{G}_1 \times \cdots \times \mathbb{G}_n \to \mathbb{G}_T$ such that

$$e(g_1^{a_1}, g_2^{a_2}, \dots, g_n^{a_n}) = g_T^{a_1 a_2 \dots a_n}$$

To generate a subset-sum encoding for a collection $S = \{v_1, \dots, v_n\}$ of n integers, choose a random $\alpha \xleftarrow{R} \mathbb{Z}_p$, and compute $V_i = g_i^{\alpha^{v_i}}$ for $i = 1, \dots, n$. Publish V_i for each i. α is kept secret.

The encoding of a target integer t is $\hat{t} = g_T^{\alpha^t}$. Given the secret α it is easy to compute $\hat{t}$. Moreover, if $t = \sum_{i \in T} i$ for some subset $T \subseteq S$, then given the public values V_i, it is also easy to compute $\hat{t}$: $\hat{t} = e(V_1^{b_1}, \dots, V_n^{b_n})$ where $b_i = 1$ if and only if $i \in T$, and $V_i^0 = g_i$. However, if t cannot be represented as a subset sum of elements in S, then there is no way to pair or multiply the V_i and g_i together to get $\hat{y}$. We conjecture that in this case, $\hat{t}$ is hard to compute. This gives rise to a new complexity assumption on multilinear maps: we say that the *multilinear subset-sum Diffie-Hellman assumption* holds for a multilinear map if, for any set of integers $S = \{v_1, \dots, v_n\}$ and any target t that cannot be represented as a subset-sum of elements in S, that $g_T^{\alpha^t}$ is indistinguishable from a random group element, even given the elements $\{g_i^{\alpha^{v_i}}\}_{i \in [n]}$[1].

Application to witness encryption Recall that in a witness encryption scheme as defined by Garg, Gentry, Sahai, and Waters [GGSW13], a message m is encrypted to an instance x, which may or may not be in some NP language L. Given a witness w that $x \in L$, it is possible to decrypt the ciphertext and recover m. However, if $x \notin L$, m should be computationally hidden.

Our subset-sum encodings immediately give us witness encryption for the language L of subset sum instances. Let (S, y) be a subset-sum instance. To encrypt a message m to (S, y), generate a

[1]We actually use an even stronger assumption, which also allows the adversary to adaptively ask for values $g_T^{\alpha^{t'}}$ for any $t' \neq t$.

subset-sum encoding for collection S. Then, using the secret encoding algorithm, compute $\hat{y}$. The ciphertext is the public evaluation function, together with $c = \hat{y} \oplus m$. To decrypt using a witness subset $T \subseteq S$, use the evaluation procedure to obtain $\hat{y}$, and then XOR with c to obtain m. Since subset-sum is NP-complete, we can use NP reductions to obtain witness encryption for any NP language L. Our scheme may be more efficient than [GGSW13] for languages L that have simpler reductions to subset-sum than exact set cover, which is used by [GGSW13].

We can also obtain a special case of Rudich secret sharing. Given a subset-sum instance (S, t), compute the elements $V_i, \hat{t}$ as above, and compute $c = \hat{t} \oplus s$ where s is the secret. Hand out share (V_i, c) to user i. Notice that a set U of users can learn s if they know a subset $T \subseteq U$ such that $\sum_{j \in T} j = t$. If no such subset exists, then our subset-sum Diffie-Hellman assumption implies that s is hidden from the group U of users.

1.3 Witness PRFs for NP

As defined above, witness PRFs are PRFs that can be evaluated on any input x for which the user knows a witness w that $x \in L$. For any $x \notin L$, the value of the PRF remains computationally hidden. Notice that subset-sum encodings *almost* give us witness PRFs for the subset-sum problem. Indeed, a subset-sum encoding instance only depends on the subset S of integers, and not the target value y. Thus, a subset-sum encoding for a set S gives us a witness PRF for the language L_S of all integers y that are subset-sums of the integers in S.

To turn this into a witness PRF for an arbitrary language, we give a reduction from any NP language L to subset-sum with the following property: the set S is determined entirely by the NP relation defining L (and the instance length), and the target y is determined by the instance x. Therefore, to build a witness PRF for any fixed NP relation R, run our reduction algorithm to obtain a set S_R, and then build a subset-sum encoding for S_R.

1.4 Replacing Obfuscation with Witness PRFs

We now explain how we use witness PRFs to remove obfuscation from certain applications.

Warm-up: No-setup non-interactive multiparty key exchange To illustrate our ideas, we discuss the application to key exchange. Consider the no-steup multiparty key exchange protocol of Boneh and Zhandry [BZ13]. Here, each party generates a seed s_i for a pseudorandom generator G, and publishes the corresponding output x_i. In addition, a designated master party builds the following program P:

- On input $x_1, \dots, x_n, s, i$, check if $\mathsf{G}(s) = x_i$.
- If the check fails, output $\perp$
- Otherwise, output $\mathsf{F}(x_1, \dots, x_n)$, where F is a pseudorandom function

The master party then publishes an obfuscation of P. Party i can now compute $K = \mathsf{F}(x_1, \dots, x_n)$ by feeding $x_1, \dots, x_n, s_i, i$ into the obfuscation of P. Thus, all parties establish the same shared key K. An eavesdropper meanwhile only gets to see the obfuscation of P and the x_i, and tries to determine K. He can do so in one of two ways:

- Run the obfuscation of P on inputs of his choice, hoping that one of the outputs is K.

- Inspect the obfuscation of P to try to learn K.

The one-wayness of G means the first approach is not viable. Boneh and Zhandry show that by obfuscating P, the value of K is still hidden, even if the adversary inspects the obfuscated code.

We now explain how witness PRFs actually suffice for this application. Notice that there are two parts to the input: the x_i, on which F is evaluated, and (s, i), which is essentially a witness that one of the x_i has a pre-image under G. We can therefore define an NP language L consisting of all tuples of x_i values where at least one of the x_i has a pre-image under G. Instead of obfuscating the program P, we can simply produce a witness PRF F for the language L, and set the shared key to be $\mathsf{F}(x_1, \ldots, x_n)$, which all the honest parties can compute since they know a witness.

To argue security, note that we can replace the x_i with random strings, and the security of G shows that the adversary cannot detect this change. Now, if the codomain of G is much larger than the seed space, then with overwhelming probability, none of the x_i have pre-images under G. This means, with overwhelming probability $(x_1, \ldots, x_n)$ is no longer in L. Therefore, the security of the witness PRF shows that the value $K = \mathsf{F}(x_1, \ldots, x_n)$ is computationally indistinguishable from random, as desired.

1.5 Other Related Work

Obfuscation. Barak et al. [BGI+01, BGI+12] begin the formal study of program obfuscation by giving several formalizations of program obfuscation, including virtual black box (VBB) obfuscation, indistinguishability obfuscation (iO), and differing inputs obfuscation (diO). They show that VBB obfuscation is impossible to achieve for general programs, though VBB obfuscation has since been achieved for very specific functionalities [CRV10]. Garg et al. [GGH+13b] give the first candidate construction of a general purpose indistinguishability obfuscator, which has been followed by several constructions [BR13, BGK+14, PST13] with improved security analyses. Boyle, Chung, and Pass [BCP14] and Ananth et al. [ABG+13] independently conjecture that current candidate indistinguishability obfuscators might actually differing inputs obfuscations (also referred to as extractability obfuscators in [BCP14]).

Smooth Projective Hash Functions. Cramer and Shoup [CS02] define the notion of a *smooth projective hash function* (SPHF), a concept is very similar to that of witness PRFs. However, SPHFs have been mostly studied for "languages of ciphertexts" for specific encryption schemes — no construction for general NP languages is known. Moreover, SPHF's for a language L are actually stronger than witness PRFs for L in that, for $x \notin L$, an SPHF requires that the value of F at x is *statistically* hidden.

Witness Encryption Garg, Gentry, Sahai, and Waters [GGSW13] define witness encryption and give the first candidate construction for the NP-Complete Exact Cover problem, whose security is based on the *multilinear no-exact-cover problem*, which they define. Goldwasser et al. [GKP+13] define a stronger notion, called extractable witness encryption, which stipulates that anyone who can distinguish the encryption of two messages relative to an instance x must actually be able to produce a witness for x. Subsequently, Garg, Gentry, Halevi, and Wichs [GGHW13] cast doubt on the plausibility of extractable witness encryption in general, though their results do not apply to most potential applications of the primitive.

Multiparty Key Exchange. The first key exchange protocol for $n = 2$ users is the celebrated Diffie-Hellman protocol. Joux [Jou04] shows how to use pairings to extend this to $n = 3$ users, and Boneh and Silverberg [BS03] show that multilinear maps give rise to n-user key exchange for any n. The first multilinear maps were constructed by Garg, Gentry, and Halevi [GGH13a] and by Coron, Lepoint, and Tibouchi [CLT13], giving the first n-user key exchange for $n > 3$. However, constructing these multilinear maps involves generating secrets, which translates to the key exchange protocols requiring a trusted setup assumption. Using obfuscation, Boneh and Zhandry [BZ13] give the first n user key exchange protocol for $n > 3$ that does not require a trusted setup.

Harcore bits. The Goldreich-Levin theorem [GL89] shows how to build a single hard-core bit for any one-way function. While this result can be extended to logarithmically-many bits, and polynomially-many hard-core bits have been constructed for *specific* one-way functions [HSS93, CGHG01], a general hard-core function outputting polynomially many bits for *all* one-way functions remained an open problem. The obfuscation-based hard-core function of Bellare and Stepanovs and Tessaro [BST13] is the first and only construction prior to this work.

Broadcast Encryption. There has been an enourmous body of work on broadcast encryption, and we only mention a few specific works. Boneh and Zhandry [BZ13] give a broadcast scheme from indistinguishability obfuscation which achieves very short secret keys and ciphertexts. Their broadcast scheme has the novel property of being distributed, where every user chooses their own secret key. However, their public keys are obfuscated programs, and are quite large (namely, linear in the number of users). Ananth et al. [ABG+13] show how to shrink the public key (while keeping secret keys and ciphertexts roughly the same size) at the expense of losing the distributed property. Boneh, Waters, and Zhandry [BWZ14] give a broadcast scheme whose concrete parameter sizes are much better directly from multilinear maps. However, this scheme is also not distributed.

Secret Sharing. The first secret sharing schemes due to Blakely [Bla79] and Shamir [Sha79] are for the *threshold* access structure, where any set of users of size at least some threashold t can recover the secret, and no set of size less than t can learn anything about the secret. In an unpublished work, Yao shows how to perform (computational) secret sharing where the allowable sets are decided by a polynomial-sized monotone circuit. Rudich raises the possibility of performing secret sharing where allowable sets are decided by a non-deterministic circuit. The first such scheme was built by Komargodski, Naor and Yogev [KNY14], and uses iO.

2 Preliminaries

2.1 Subset-Sum

Let $\mathbf{A} \in \mathbb{Z}^{m\times n}$ be an integer matrix, and $\mathbf{t} \in \mathbb{Z}^n$ be an integer vector. The *subset-sum* search problem is to find an $\mathbf{w} \in \{0,1\}^n$ such that $\mathbf{t} = \mathbf{A}\cdot\mathbf{w}$. The decision problem is to decide if such an $\mathbf{w}$ exists.

We define several quantities related to a subset-sum instance. Given a matrix $\mathbf{A} \in \mathbb{Z}^{m\times n}$, let $\mathsf{SubSums}(\mathbf{A})$ be the set of all subset-sums of columns of $\mathbf{A}$. That is, $\mathsf{SubSums}(\mathbf{A}) = \{\mathbf{A}\cdot\mathbf{w} : \mathbf{w} \in \{0,1\}^n\}$. Define $\mathsf{Span}(\mathbf{A})$ as the convex hull of $\mathsf{SubSums}(\mathbf{A})$. Equivalently, $\mathsf{Span}(\mathbf{A}) = \{\mathbf{A}\cdot\mathbf{w} : \mathbf{w} \in [0,1]^n\}$. We define the integer range of $\mathbf{A}$, or $\mathsf{IntRange}(\mathbf{A})$, as $\mathsf{Span}(\mathbf{A})\bigcap\mathbb{Z}^m$. We note that given an instance $(\mathbf{A}, \mathbf{t})$ of the subset-sum problem, it is efficiently decidable whether $\mathbf{t} \in \mathsf{IntRange}(\mathbf{A})$. Moreover, $\mathbf{t} \notin \mathsf{IntRange}(\mathbf{A})$ implies that $(\mathbf{A}, \mathbf{t})$ is unsatisfiable. The only "interesting" instances of the subset sum problem therefore have $\mathbf{t} \in \mathsf{IntRange}(\mathbf{A})$. From this point forward, we only consider $(\mathbf{A}, \mathbf{t})$ a valid subset sum instance if $\mathbf{t} \in \mathsf{IntRange}(\mathbf{A})$.

2.2 Multilinear Maps

An asymmetric multilinear map [BS03] is defined by an algorithm Setup which takes as input a security parameter λ, a multilinearity n, and a minimum group order p_{min}[2]. It outputs (the description of) $n+1$ groups $\mathbb{G}_1, \ldots, \mathbb{G}_n, \mathbb{G}_T$ of prime order $p \geq \max(2^\lambda, p_{min})$, corresponding generators $g_1, \ldots, g_n, g_T$, and a map $e : \mathbb{G}_1 \times \cdots \times \mathbb{G}_n \to \mathbb{G}_T$ satisfying

$$e(g_1^{a_1}, \ldots, g_n^{a_n}) = g_T^{a_1 \ldots a_n}$$

Approximate Multilinear Maps. Current candidate multilinear maps [GGH13a, CLT13] are only *approximate* and do not satisfy the ideal model outlined above. In particular, the maps are noisy. This has several implications. First, representations of group elements are not unique. Current map candidates provide an extraction procedure that takes a representation of a group element in the the target group $\mathbb{G}_T$ and outputs a canonical representation. This allows multiple users with different representations of the same element to arrive at the same value.

A more significant limitation is that noise grows with the number of multiplications and pairing operations. If the noise term grows too large, then there will be errors in the sense that the extraction procedure above will fail to output the canonical representation.

Lastly, and most importantly for our use, current map candidates do not allow regular users to compute g_i^α for any $\alpha \in \mathbb{Z}_p$ of the user's choice. Instead, the user computes a "level-0 encoding" of a random (unknown) $\alpha \in \mathbb{Z}_p$, and then pairs the "level-0 encoding" with g_i, which amounts computing the exponentiation g_i^α. To compute terms like $g_i^{\alpha^k}$ would require repeating this operation k times, resulting in a large blowup in the error. Thus, for large k, computing terms like $g_i^{\alpha^k}$ is infeasible for regular users. However, whomever sets up the map knows secret parameters about the map and *can* compute g_i^α for any $\alpha \in \mathbb{Z}_p$ without blowing up the error. Thus, the user who sets up the map can pick α, compute α^k in $\mathbb{Z}_p$, and then compute $g_i^{\alpha^k}$ using the map secrets. This will be critical for our constructions.

3 Witness PRFs

Informally, a witness PRF is a generalization of constrained PRFs to arbitrary NP relations. That is, for an NP language L, a user can evaluate the function F at an instance x only if $x \in L$ *and* the user can provide a witness w that $x \in L$. More formally, a witness PRF is the following:

Definition 3.1. A witness PRF is a triple of algorithms $(\mathsf{Gen}, \mathsf{F}, \mathsf{Eval})$ such that:

- Gen is a randomized algorithm that takes as input a security parameter λ and a circuit $R : \mathcal{X} \times \mathcal{W} \to \{0,1\}$[3], and produces a secret function key fk and a public evaluation key ek.

[2] It is easy to adapt multilinear map constructions [GGH13a, CLT13] to allow setting a minimum group order.

[3] By accepting relations as circuits, our notion of witness PRFs only handles instances of a fixed size. It is also possible to consider witness PRFs for instances of arbitrary size, in which case R would be a Turing machine.

- F is a deterministic algorithm that takes as input the function key fk and an input $x \in \mathcal{X}$, and produces some output $y \in \mathcal{Y}$ for some set $\mathcal{Y}$.
- Eval is a deterministic algorithm that takes as input the evaluation key ek and input $x \in \mathcal{X}$, and a witness $w \in \mathcal{W}$, and produces an output $y \in \mathcal{Y}$ or $\perp$.
- For correctness, we require $\mathsf{Eval}(\mathsf{ek}, x, w) = \begin{cases} \mathsf{F}(\mathsf{fk}, x) & \text{if } R(x,w) = 1 \\ \perp & \text{if } R(x,w) = 0 \end{cases}$ for all $x \in \mathcal{X}, w \in \mathcal{W}$.

3.1 Security

The simplest and most natural security notion we consider is a direct generalization of the security notion for constrained PRFs, which we call adaptive instance interactive security. Consider the following experiment $\mathtt{EXP}^R_{\mathcal{A}}(b, \lambda)$ between an adversary $\mathcal{A}$ and challenger, parameterized by a relation $R : \mathcal{X} \times \mathcal{W} \to \{0,1\}$, a bit b and security parameter λ.

- Run $(\mathsf{fk}, \mathsf{ek}) \xleftarrow{R} \mathsf{Gen}(\lambda, R)$ and give ek to $\mathcal{A}$.
- $\mathcal{A}$ can adaptively make queries on instances $x_i \in \mathcal{X}$, to which the challenger response with $\mathsf{F}(\mathsf{fk}, x_i)$.
- $\mathcal{A}$ can make a single challenge query on an instance $x^* \in \mathcal{X}$. The challenger computes $y_0 \leftarrow \mathsf{F}(\mathsf{fk}, x^*)$ and $y_1 \xleftarrow{R} \mathcal{Y}$, and responds with y_b.
- After making additional F queries, $\mathcal{A}$ produces a bit b'. The challenger checks that $x^* \notin \{x_i\}$, and that there is no witness $w \in \mathcal{W}$ such that $R(x, w) = 1$ (in other words, $x \notin L$)[4]. If either check fails, the challenger outputs a random bit. Otherwise, it outputs b'.

Define W_b as the event the challenger outputs 1 in experiment b. Let $\mathtt{WPRF.Adv}^R_{\mathcal{A}}(\lambda) = |\Pr[W_0] - \Pr[W_1]|$.

Definition 3.2. $\mathsf{WPRF} = (\mathsf{Gen}, \mathsf{F}, \mathsf{Eval})$ is *adaptive instance interactively secure* for a relation R if, for all PPT adversaries $\mathcal{A}$, there is a negligible function negl such that $\mathtt{WPRF.Adv}^R_{\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.

We can also define a weaker notion of *static instance* security where $\mathcal{A}$ commits to x^* before seeing ek or making any F queries. Independently, we can also define *non-interactive* security where the adversary is not allowed any F queries.

Fine-grained security notions. While adaptive instance interactive security will suffice for many applications, it is in some ways stronger than necessary. For example, for several applications, the witness is chosen by the reduction algorithm, not the adversary. Therefore, we aim to give more fine-grained notions of security, similar to the obfuscation-based notions of [BST13]. Such notions might be more plausible than the general purpose notion above, yet suffice for applications.

To that end, we define an *adaptive* R-instance sampler for WPRF as a PPT algorithm $\mathcal{D}$ that does the following. $\mathcal{D}$ is given ek derived as $(\mathsf{fk}, \mathsf{ek}) \xleftarrow{R} \mathsf{Gen}(\lambda, R)$, and is then allowed to make a polynomial number of queries on instances x_i, receiving $\mathsf{F}(\mathsf{fk}, x_i)$ in response. Finally, $\mathcal{D}$ produces an instance $x^* \notin \{x_i\}$ and potentially some auxiliary information Aux. We say that $\mathcal{D}$ is *static* if $\mathcal{D}$

[4] This check in general cannot be implemented in polynomial time, meaning our challenger is not efficient.

does not depend on ek and does not make any F queries (but may still depend on λ). Finally, $\mathcal{D}$ is semi-static if it does not make any F queries, but may depend on ek.

We now define and experiment $\mathtt{EXP}^R_{\mathcal{D},\mathcal{A}}(b,\lambda)$ between a challenger and an algorithm $\mathcal{A}$, parameterized by relation R, adaptive, semin-static, or static R-instance sampler $\mathcal{D}$, and bit b:

- $(\mathsf{fk},\mathsf{ek})\xleftarrow{R}\mathsf{Gen}(\lambda,R)$, $(x^*,\mathsf{Aux})\xleftarrow{R}\mathcal{D}^{\mathsf{F}(\mathsf{fk},\cdot)}(\mathsf{ek})$, $y_0\leftarrow\mathsf{F}(\mathsf{fk},x^*), y_1\xleftarrow{R}\mathcal{Y}$. Give $\mathsf{ek},x^*,\mathsf{Aux},y_b$ to $\mathcal{A}$
- $\mathcal{A}$ is allowed to make F queries on instances $x_i\in\mathcal{X}, x_i\neq x^*$, to which the challenger responds with $\mathsf{F}(\mathsf{sk},x_i)$.
- $\mathcal{A}$ eventually outputs a guess b'. If there is a witness $w\in\mathcal{W}$ such that $R(x^*,w)=1$ (in other words, if $x^*\in L$), then the challenger outputs a random bit. Otherwise, it outputs b'

We define W_b to be the event of outputting 1 in experiment b, and define the advantage of $\mathcal{A}$ to be $\mathtt{WPRF.Adv}_{\mathcal{D},\mathcal{A}}\mathcal{A}^{\mathbf{A}}(\lambda)=|\Pr[W_0]-\Pr[W_1]|$

We now define our main notion of security for witness PRFs:

Definition 3.3. $\mathsf{WPRF}=(\mathsf{Gen},\mathsf{F},\mathsf{Eval})$ is *interactively secure* for relation R and R-instance sampler $\mathcal{D}$ if, for all PPT adversaries $\mathcal{A}$, there is a negligible function negl such $\mathtt{WPRF.Adv}_(\mathcal{D},\mathcal{A})^R(\lambda)<\mathsf{negl}(\lambda)$.

We can also define non-interactive security where we do not allow $\mathcal{A}$ to make any F queries.

We can recast adaptive witness interactive security in this framework:

Fact 3.4. WPRF *is* adaptive witness interactively secure *for a relation R if it is interactively secure for R and all adaptive R-instance samplers $\mathcal{D}$.*

Extractable Witness PRFs. For some applications, we will need an extractable notion of witness PRF, which roughly states that $\mathsf{F}(x)$ is pseudorandom even for instances $x\in L$, unless the adversary "knows" a witness w for x.

Formally, we modify $\mathtt{EXP}^R_{\mathcal{D},\mathcal{A}}(b,\lambda)$ to get a new extracting experiment $\mathtt{EXP}^{e,R}_{\mathcal{D},\mathcal{A}}(b,\lambda)$ where we remove the check that $x\notin L$, and define $\mathtt{WPRF.Adv}^{e,R}_{\mathcal{D},\mathcal{A}}(\lambda)$ as the advantage of $(\mathcal{D},\mathcal{A})$ in this new game. We also define a second experiment for an extractor $\mathcal{E}$:

- $(\mathsf{fk},\mathsf{ek})\xleftarrow{R}\mathsf{Gen}(\lambda,R)$, $(x^*,\mathsf{Aux})\xleftarrow{R}\mathcal{D}^{\mathsf{F}(\mathsf{fk},\cdot)}(\mathsf{ek})$, $y^*\leftarrow\mathsf{F}(\mathsf{fk},x^*)$, $b'\xleftarrow{R}\mathcal{A}^{\mathsf{F}(\mathsf{fk},\cdot)}(\mathsf{ek},x^*,\mathsf{Aux},y^*)$
- Let $\{(x_i,y_i)\}$ be the F queries and responses made by $\mathcal{A}$ and r the random coins used by $\mathcal{A}$. Run $w^*\xleftarrow{R}\mathcal{E}(\mathsf{ek},x^*,\mathsf{Aux},y^*,\{x_i\},r)$. Output $R(x^*,w^*)$.

Define the advantage $\mathtt{EWPRF.Adv}^R_{\mathcal{D},\mathcal{E}}(\lambda)$ as the probability the challenger outputs 1.

Definition 3.5. $(\mathsf{Gen},\mathsf{F},\mathsf{Eval})$ is *extractable interactively secure* for a relation R and R-instance sampler $\mathcal{D}$ if, for all PPT adversaries $\mathcal{A}$ such that $\mathtt{WPRF.Adv}^R_{\mathcal{D},\mathcal{A}}(\lambda)>1/q_{\mathcal{A}}(\lambda)$ for some polynomial $q_{\mathcal{A}}$, there is an efficient extractor $\mathcal{E}$ and polynomial $q_{\mathcal{E}}$ such that $\mathtt{EWPRF.Adv}^R_{\mathcal{D},\mathcal{E}}(\lambda)>1/q_{\mathcal{E}}(\lambda)$.

Similarly, we define the relaxation to non-interactive security as we did for standard witness PRFs, where $\mathcal{A}$ is not allowed any F queries.

Remark 3.6. Notice that we've restricted the extractor to only making the same queries made by $\mathcal{A}$. This is potentially a stronger requirement than necessary for an extractable witness PRF. However, this restriction will become important in our constructions. For example, consider constructing an extractable witness PRF WPRF for a relation R by first building an extractable witness PRF WPRF' for an NP-Complete relation R', and then performing an NP reduction. To prove extractable security for WPRF, begin with an adversary $\mathcal{A}$ for WPRF. Use $\mathcal{A}$ and the NP reduction to construct an adversary $\mathcal{A}'$ for WPRF'. The existence of $\mathcal{A}'$ implies an extractor $\mathcal{E}'$ for WPRF'. The goal is to use this extractor to build an extractor $\mathcal{E}$ for WPRF. The problem is that, in the reduction, a legal query to WPRF' might not correspond to a legal query for WPRF. Thus if $\mathcal{E}'$ is allowed to make arbitrary queries, then there is no way for $\mathcal{E}$ to simulate them. However, if $\mathcal{E}'$ can only make queries made by $\mathcal{A}'$, this is no longer a problem since the queries made by $\mathcal{A}'$ will correspond to queries made by $\mathcal{A}$, which *are* legal WPRF queries.

We can now give a general extractability definition for witness PRFs:

Definition 3.7. A witness PRF is extractable static witness interactively secure for relation R if it is extractable interactively secure for R and any static R-instance sampler $\mathcal{D}$.

Remark 3.8. It is also possible to define semi-static or adaptive variants of the above. However, these variants are not attainable for many relations R. For example, consider a relation R where it is easy to sample instances in the language along with witnesses, but given only the instance, finding a witness is hard (an example of such a language is the language of outputs of a one-way function, where witnesses are the corresponding inputs). Then consider the following semi-static instance sampler: sample $x^* \in L$ along with witness w, and use w and ek to compute $y^* = \mathsf{Eval}(\mathsf{ek}, x^*, w) = \mathsf{F}(\mathsf{fk}, x^*)$. Output x^* as the instance and y^* as the auxiliary information. Clearly, given y^*, it is easy to distinguish $\mathsf{F}(\mathsf{fk}, x^*)$ from random. However, this is insufficient for extracting a witness w for x^*.

Remark 3.9. We will eventually show that the extractable witness PRFs imply extractable witness encryption. The recent work of Garg, Gentry, Halevi, and Wichs [GGHW13] shows that extractable witness encryption is problematic, casting some doubt on the plausability of of building extractable witness PRFs. The same doubt is cast on our extractable multilienar map assumptions and our intermediate notion of an extractable subset-sum encoding to be defined later. However, we stress that the results of [GGHW13] only apply to specific auxilliary inputs Aux, which turn out to be the obfuscations of certain programs. However, in many applications $\mathcal{D}$ will be determined by the reduction algorithm (that is, not the adversary) and Aux will be very simple or even non-existant. Therefore, the results of [GGHW13] will often not apply. While it may be impossible to build extractable witness PRFs for all $\mathcal{D}$, it seems plausible to build extractable witness PRFs for the specific applications we investigate.

Remark 3.10. We note the counter-intuitive property that extractable witness PRFs to not imply standard witness PRFs. Consider an R-instance sampler that outputs an instance $x \notin L$ with probability $1/2$, and outputs an instance $x \in L$ with an easy-to-compute witness with probability $1/2$. Extractability trvially holds, since it is possible to extract a witness with probability $1/2$. However, non-extracting security may not hold, as the cases where $x \in L$ are eliminated by the challenger.

4 Applications

In this section, we show several applications of obfuscation, the obfuscator can be replaced with witness PRFs. We break the applications into several categories:

- *Inherent sampler* constructions are those whose security is proven relative to a fixed instance sampler determined entirely by the construction. Of our constructions, these are the most plausible, since they will not be subject to the impossibility results in the literature [GGHW13]. Our constructions are CCA-secure encryption, non-interactive key exchange, and hardcore functions for any one-way function.

- *Parameterized sampler* constructions are those where the security definition of the primitive depends on an instance sampler $\mathcal{D}$, and security holds relative to $\mathcal{D}$ if the underlying witness PRF is secure for some other instance sampler $\mathcal{D}'$ derived from $\mathcal{D}$. Such constructions include (reusable) witness encryption, and (resuable) secret sharing for monotone NP. These constructions are likely to be secure for some samplers, but may not be secure for all samplers.

- *Restricted sampler class* constructions are those where the instance sampler depends on the adversary $\mathcal{A}$, meaning the witness PRF must be secure for a large class of samplers. However, we show that the sampler is still restricted, meaning security must only hold relative to a restricted set of samplers. Because of the restriction on instance samplers, it is still plausible that the construction is secure even considering impossibility results. Our fully-dynamic broadcast encryption scheme falls into this category.

4.1 CCA-secure Public Key Encryption

We demonstrate that the CCA-secure public key encryption of Sahai and Waters [SW14] can be instantiated from witness PRFs.

Construction 4.1. Let $\mathsf{WPRF} = (\mathsf{WPRF.Gen}, \mathsf{F}, \mathsf{Eval})$ be a witness PRF, and let $\mathsf{G} : \mathcal{S} \to \mathcal{Z}$ be a pseudorandom generator with $|\mathcal{S}|/|\mathcal{Z}| < \mathsf{negl}$. Build the following key encapsulation mechanism $(\mathsf{Enc.Gen}, \mathsf{Enc}, \mathsf{Dec})$:

- $\mathsf{Enc.Gen}(\lambda)$: Let $R(z, s) = 1$ if and only if $\mathsf{G}(s) = z$. In other words, R defines the language L of strings $z \in \mathcal{Z}$ that are images of G, and witnesses are the corresponding pre-images. Run $(\mathsf{fk}, \mathsf{ek}) \stackrel{R}{\leftarrow} \mathsf{WPRF.Gen}(\lambda, R)$. Set fk to be the secret key and ek to be the public key.

- $\mathsf{Enc}(\mathsf{ek})$: sample $s \stackrel{R}{\leftarrow} \mathcal{S}$ and set $z \leftarrow \mathsf{G}(s)$. Output z as the header and $k \leftarrow \mathsf{Eval}(\mathsf{ek}, z, s) \in \mathcal{Y}$ as the message encryption key.

- $\mathsf{Dec}(\mathsf{fk}, z)$: run $k \leftarrow \mathsf{F}(\mathsf{fk}, z)$.

Correctness is immediate. For security, we have the following:

Theorem 4.2. *If* WPRF *is interactively secure, then Construction 4.1 is a CCA secure key encapsulation mechanism. If* WPRF *is static instance non-interactively secure, then Construction 4.1 is CPA secure.*

Rather than prove Theorem 4.2, we instead prove security relative to a fine-grained security notion. Define the following static instance sampler $\mathcal{D}$: sample and output a random $z \in \mathcal{Z}$ and $\mathsf{Aux} = ()$.

Theorem 4.3. *If* G *is a secure pseudorandom generator and* WPRF *is interactively secure for relation* R *and* R*-instance sampler* $\mathcal{D}$*, then Construction 4.1 is a CCA secure key encapsulation mechanism. If* WPRF *is non-interactively secure, then Construction 4.1 is CPA secure.*

Proof. We prove the CCA case, the CPA case being almost identical. Let $\mathcal{B}$ be a CCA adversary with non-negligible advantage ϵ. Define **Game 0** as the standard CCA game, and define **Game 1** as the modification where the challenge header z^* is chosen uniformly at random in $\mathcal{Z}$. The security of G implies that $\mathcal{B}$ still has advantage negligibly-close to ϵ. Let **Game 2** be the game where z^* is chosen at random, but the game outputs a random bit and aborts if z^* is in the image space of G. Since $\mathcal{Z}$ is much larger than $\mathcal{S}$, the abort condition occurs with negligible probability. Thus $\mathcal{B}$ still has advantage negligibly close to ϵ in **Game 2**. Now we construct an adversary $\mathcal{A}$ for WPRF relative to sampler $\mathcal{D}$. $\mathcal{A}$ simulates $\mathcal{B}$, answering decryption queries using its F oracle. Finally, $\mathcal{B}$ makes a challenge query, and $\mathcal{A}$ responds with its input z^*. When $\mathcal{B}$ outputs a bit b', $\mathcal{A}$ outputs the same bit. $\mathcal{A}$ has advantage equal to that of $\mathcal{B}$ in **Game 2**, which is non-negligible, thus contradicting the security of WPRF. □

We can also relax the requirement on G to be a one-way function if we assume WPRF is extractable. Let $\mathcal{D}'$ be the following static instance sampler: sample $s \xleftarrow{R} \mathcal{S}$ and output $z = f(s)$ and $\mathsf{Aux} = ()$. Then we have the following theorem:

Theorem 4.4. *If* G *is a secure one-way function and* WPRF *is extractable interactively secure for relation* R *and* R*-instance sampler* $\mathcal{D}'$*, then Construction 4.1 is a CCA secure key encapsulation mechanism. If* WPRF *is extractable non-interactively secure, then Construction 4.1 is CPA secure.*

The proof is very similar to the proof of Theorem 4.3, and we omit the details.

4.2 Non-interactive Multiparty Key Exchange

A multiparty key exchange protocol allows a group of g users to simultaneously post a message to a public bulletin board, retaining some user-dependent secret. After reading off the contents of the bulletin board, all the users establish the same shared secret key. Meanwhile, and adversary who sees the entire contents of the bulletin board should not be able to learn the group key. More precisely, a multiparty key exchange protocol consists of:

- $\mathsf{Publish}(\lambda, g)$ takes as input the security parameter and the group order, and outputs a user secret s and public value pv. pv is posted to the bulletin board.
- $\mathsf{KeyGen}(\{\mathsf{pv}_j\}_{j\in[g]}, s_i, i)$ takes as input g public values, plus the corresponding user secret s_i for the ith value. It outputs a group key $k \in \mathcal{Y}$.

For correctness, we require that all users generate the same key:

$$\mathsf{KeyGen}(\{\mathsf{pv}_j\}_{j\in[g]}, s_i, i) = \mathsf{KeyGen}(\{\mathsf{pv}_j\}_{j\in[g]}, s_{i'}, i')$$

for all $(s_j, \mathsf{pv}_j) \xleftarrow{R} \mathsf{Publish}(\lambda, g)$ and $i, i' \in [g]$. For security, we have the following:

Definition 4.5. A non-interactive multiparty key exchange protocol is statically secure if the following distributions are indistinguishable:

$$\{\mathsf{pv}_j\}_{j\in[g]}, k \text{ where } (s_j, \mathsf{pv}_j) \xleftarrow{R} \mathsf{Publish}(\lambda, g) \forall j \in [g], k \leftarrow \mathsf{KeyGen}(\{\mathsf{pv}_j\}_{j\in[g]}, s_1, 1) \text{ and}$$
$$\{\mathsf{pv}_j\}_{j\in[g]}, k \text{ where } (s_j, \mathsf{pv}_j) \xleftarrow{R} \mathsf{Publish}(\lambda, g) \forall j \in [g], k \xleftarrow{R} \mathcal{Y}$$

Notice that our syntax does not allow a trusted setup, as constructions based on multilinear maps [BS03, GGH13a, CLT13] require. Boneh and Zhandry [BZ13] give the first multiparty key exchange protocol without trusted setup, based on obfuscation. We now give a very similar protocol using witness PRFs.

Construction 4.6. Let $\mathsf{G} : \mathcal{S} \to \mathcal{Z}$ be a pseudorandom generator with $|\mathcal{S}|/|\mathcal{Z}| < \mathsf{negl}$. Let $\mathsf{WPRF} = (\mathsf{Gen}, \mathsf{F}, \mathsf{Eval})$ be a witness PRF. Let $R_g : \mathcal{Z}^g \times (\mathcal{S} \times [g]) \to \{0, 1\}$ be a relation that outputs 1 on input $((z_1, \dots, z_g), (s, i))$ if and only if $z_i = \mathsf{G}(s)$. We build the following key exchange protocol:

- $\mathsf{Publish}(\lambda, g)$: compute $(\mathsf{fk}, \mathsf{ek}) \xleftarrow{R} \mathsf{Gen}(\lambda, R_g)$. Also pick a random seed $s \xleftarrow{R} \mathcal{S}$ and compute $z \leftarrow \mathsf{G}(s)$. Keep s as the secret and publish (z, ek).
- $\mathsf{KeyGen}(\{(z_i, \mathsf{ek}_i)\}_{i\in[g]}, s)$. Each user sorts the pairs (z_i, ek_i) by z_i, and determines their index i in the ordering. Let $\mathsf{ek} = \mathsf{ek}_1$, and compute $k = \mathsf{Eval}(\mathsf{ek}, (z_1, \dots, z_g), (s, i))$

Correctness is immediate. For security, we have the following:

Theorem 4.7. *If* WPRF *is static witness non-interactively secure, the Construction 4.6 is statically secure.*

Rather than prove Theorem 4.7, we instead prove security relative to a fine-grained security notion. Let $\mathcal{D}_g$ be the following R_g-instance sampler: choose random $z_i \xleftarrow{R} \mathcal{Z}$ for $i \in [g]$, and output $(z_1, \dots, z_g), \mathsf{Aux} = ()$.

We have the following theorem:

Theorem 4.8. *If* WPRF *is non-interactively secure for relation* R_g *and* R*-instance sampler* $\mathcal{D}_g$*, and* G *is a secure PRG, then* $(\mathsf{Publish}, \mathsf{KeyGen})$ *is a statically secure NIKE protocol.*

We can also trade a stronger notion of security for the witness PRF in exchange for a weaker security requirement for G. Let $\mathcal{D}'_g$ be the following static R_g-instance sampler: choose random $s_i \xleftarrow{R} \mathcal{S}$, and set $z_i \leftarrow \mathsf{G}(s_i)$ and output $(z_1, \dots, z_g), \mathsf{Aux} = ()$

Theorem 4.9. *If* WPRF *is extracting non-interactively secure for relation* R_g *and* R*-instance sampler* $\mathcal{D}'_g$*, and* G *is a secure one-way function, then* $(\mathsf{Publish}, \mathsf{KeyGen})$ *is a statically secure NIKE protocol.*

Proof. We prove Theorem 4.8, the proof of Theorem 4.9 being similar. Let $\mathcal{B}$ be an adversary for the key exchange protocol with non-negligible advantage. The $\mathcal{B}$ sees $\{(z_i, \mathsf{ek}_i)\}_{i\in[g]}$ where $z_i \leftarrow \mathsf{G}(s_i)$ for a random $s_i \xleftarrow{R} \mathcal{S}$, as well as a key $k \in \mathcal{Y}$, and outputs a guess b' for whether $k = \mathsf{F}(\mathsf{ek}_1, \{(z_i)\}_{i\in[g]}$ or $k \xleftarrow{R} \mathcal{Y}$. Call this **Game 0**. Define **Game 1** as the modification where $z_i \xleftarrow{R} \mathcal{Z}$. The security of G implies that **Game 0** and **Game 1** are indistinguishable. Next define **Game 2** as identical to **Game 1**, except that the challenger outputs a random bit and aborts if any of the z_i are in the range of G. Since $|\mathcal{S}|/|\mathcal{Z}| < \mathsf{negl}$, this abort condition occurs with negligible probability, meaning $\mathcal{B}$ still has non-negligible advantage in **Game 2**. We construct an adversary $\mathcal{A}$ for WPRF relative to sampler $\mathcal{D}_g$ as follows: $\mathcal{A}$, on input $\mathsf{ek}, \{z_i\}_{i\in[g]}, k$ (where $\{z_i\}\stackrel{R}{\leftarrow}\mathcal{D}_g$), sorts the z_i, and then sets $\mathsf{ek}_1 = \mathsf{ek}$. For $i > 1$, $\mathcal{A}$ runs $(\mathsf{fk}_i, \mathsf{ek}_i)\stackrel{R}{\leftarrow}\mathsf{Gen}(\lambda, R_g)$. It then gives $\mathcal{A}$ $\{(z_i, \mathsf{ek}_i)\}_{i\in[g]}, k$. Note that for key generation, $\mathsf{ek}_1 = \mathsf{ek}$ is chosen. Also, $(z_1, \ldots, z_g)$ is chosen at random in $\mathcal{Z}^g$, and $\mathcal{A}$'s challenger aborts if any of the z_g are in the range of G (that is, if $(z_1, \ldots, z_g)$ has a witness under R_g). Therefore, the view of $\mathcal{B}$ as a subroutine of $\mathcal{A}$ and the view of $\mathcal{B}$ in **Game 2** are identical. Therefore, the advantage of $\mathcal{A}$ is also non-negligible, a contradiction. □

Adaptive Security. In semi-static or active security (defined by Boneh and Zhandry [BZ13]), the same published values pv_j are used in many key exchanges, some involving the adversary. Obtaining semi-static or adaptive security from even the strongest forms of witness PRFs is not immediate. The issue, as noted by Boneh and Zhandry in the case of obfuscation, is that, even in the semi-static setting, the adversary may see the output of Eval on honest secrets, but using a malicious key ek. It may be possible for a malformed key to leak the honest secrets, thereby allowing the scheme to be broken. In more detail, consider an adversary $\mathcal{A}$ playing the role of user i, and suppose the maximum number of users in any group is 2. $\mathcal{A}$ generates and publishes params_i in a potentially malicious way (and also generates and publishes some z_i). Meanwhile, an honest user j publishes an honest ek_j and $z_j = G(s_j)$. Now, if $z_i < z_j$, user j computes the shared key for the group $\{i, j\}$ as $\mathsf{Eval}(\mathsf{ek}_i, (z_i, z_j), s_j, 2)$. While an honest ek_i would cause Eval to be independent of the witness, it may be possible for a dishonest ek_i to cause Eval to leak information about the witness.

Boneh and Zhandry circumvent this issue by using a special type of signature scheme, and only inputting signatures into Eval. Even if the entire signature leaks, it will not help the adversary produce the necessary signature to break the scheme. Unfortunately, their special signature scheme requires obfuscation to build, and it is not obvious that such signatures can be built from witness PRFs. Therefore, we leave obtaining an adaptive notion of security from witness PRFs as an interesting open problem.

We note that it is straightforward to give a semi-static scheme that requires a trusted setup from witness PRFs. The idea is to make generation of ek the responsibility of a trusted authority and make all groups use ek to derive the shared secret. This sidesteps the issues outlined above. The adaptive witness interactive security of the witness PRF then implies the semi-static security of the scheme. We omit the details.

4.3 Poly-many hardcore bits for any one-way function

A hardcore function for a function $f : \mathcal{S} \to \mathcal{Z}$ is a function $h : \mathcal{S} \to \mathcal{Y}$ such that $(f(s), h(s))$ for a random $s\stackrel{R}{\leftarrow}\mathcal{S}$ is indistinguishable from $(f(s), y)$ for a random $s\stackrel{R}{\leftarrow}\mathcal{S}$ and random $y\stackrel{R}{\leftarrow}\mathcal{Y}$. We now give our construction, based on the construction of [BST13]:

Construction 4.10. Let $f : \mathcal{S} \to \mathcal{Z}$ be any one-way function. Let $\mathsf{WPRF} = (\mathsf{Gen}, \mathsf{F}, \mathsf{Eval})$ be a witness PRF. We build a function $h : \mathcal{S} \to \mathcal{Y}$ as follows:

- Define $R_f(x, s) = 1$ if and only if $x = f(s)$.
- Run $(\mathsf{fk}, \mathsf{ek})\stackrel{R}{\leftarrow}\mathsf{Gen}(\lambda, R)$.
- Define $h(s) = \mathsf{Eval}(\mathsf{ek}, f(s), s)$.

For security, let $\mathcal{D}_f$ be the following R_f-instance sampler: choose a random $s^* \in \mathcal{S}$, compute $z^* = f(s^*)$, and output $(z^*, \mathsf{Aux} = ())$.

Theorem 4.11. *If f is a one-way function and $(\mathsf{Gen}, \mathsf{F}, \mathsf{Eval})$ is extractable non-interactively secure for relation R_f and sampler $\mathcal{D}_f$, then h in Construction 4.10 is hardcore for f.*

Proof. Let $\mathcal{A}$ be an adversary that distinguishes h from random with inverse polynomial probability $1/q_{\mathcal{A}}$. That is, given $f(s^*)$ for a random s^*, $\mathcal{A}$ is able to distinguish $h(s^*)$ from a random string. Then $\mathcal{A}$ is actually a non-interactive adversary for WPRF relative to relation R and instance sampler $\mathcal{D}_f$. In other words, $\texttt{WPRF.Adv}^R_{\mathcal{D},\mathcal{A}}(\lambda) \geq 1/q_{\mathcal{A}}$. The extracting security of the witness PRF implies that there is an extractor $\mathcal{E}$ and polynomial $q_{\mathcal{E}}$ such that $f(\mathcal{E}(\mathsf{ek}, s^*)) = s^*$. In otherwords, $\mathcal{E}$ breaks the one-wayness of f, reaching a contradiction. $\square$

4.4 Witness Encryption

We show how to build witness encryption from witness PRFs. A witness encryption scheme is parameterized by a relation $R : \mathcal{X} \times \mathcal{W} \to \{0, 1\}$, and consists of the following algorithms:

- $\mathsf{Enc}(\lambda, x, m)$ outputs a ciphertext c
- $\mathsf{Dec}(x, w, c)$ outputs a message m or $\perp$. For correctness, we require that if $R(x, w) = 1$, then $\mathsf{Dec}(x, w, \mathsf{Enc}(\lambda, x, m)) = m$ and if $R(x, w) = 0$, $\mathsf{Dec}(x, w, c) = \perp$.

For security, we use a notion similar to Bellare and Hoang [BH13], but with a minor modification. We have the notion of an R-instance sampler $\mathcal{D}$, which takes the security parameter λ and samples an instance x and auxilliary information Aux[5]. Let $\texttt{EXP}^R_{\mathcal{D},\mathcal{A}}(b, \lambda)$ denote the following experiment on an adversary $\mathcal{A}$: Run $(x, \mathsf{Aux}) \xleftarrow{R} \mathcal{D}(\lambda)$, and then run $\mathcal{A}(x, \mathsf{Aux})$. At some point, $\mathcal{A}$ produces a pair of messages (m_0, m_1), to which the challenger responds with $\mathsf{Enc}(\lambda, x, m_b)$. $\mathcal{A}$ then outputs a guess b' for b. The challenger checks if there is a w such that $R(x, w) = 1$ (that is, checks if $x \in L$), and if so, outputs a random bit. Otherwise, the challenger outputs b'. We define W_b to be the event of outputting 1 in experiment b, and define the advantage of $\mathcal{A}$ to be $\texttt{WENC.Adv}^R_{\mathcal{D},\mathcal{A}}(\lambda) = |\Pr[W_0] - \Pr[W_1]|$.

Definition 4.12. A witness encryption scheme is soundness secure for an R-instance sampler $\mathcal{D}$ if, for all adversaries $\mathcal{A}$, there is a negligible function negl such that $\texttt{WENC.Adv}^R_{\mathcal{D},\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.

We can also define an extractability definition where we remove the check that $x \in L$, and require that any distinguishing adversary gives rise to an extractor that can find a witness.

Our construction is the following:

Construction 4.13. Let R be a relation, and let $(\mathsf{Gen}, \mathsf{F}, \mathsf{Eval})$ be a witness PRF for R. We define a witness encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ where:

- $\mathsf{Enc}(\lambda, x, m)$ computes $(\mathsf{fk}, \mathsf{ek}) \xleftarrow{R} \mathsf{Gen}(\lambda, R)$, $K \leftarrow \mathsf{F}(\mathsf{fk}, x)$, and $c = K \oplus m$. Output the ciphertext (ek, c).

[5] In [BH13], the sampler also outputs the challenge messages m_0, m_1. We let the adversary produce m_0, m_1. As Aux can contain the challenge, our notion is potentially stronger

- $\mathsf{Dec}(x, w, (\mathsf{ek}, c))$ checks that $R(x, w) = 1$, and aborts otherwise. Then it computes $K \leftarrow \mathsf{Eval}(\mathsf{ek}, x, w)$, and outputs $c \oplus K$.

Correctness is immediate from the correctness of $(\mathsf{Gen}, \mathsf{F}, \mathsf{Eval})$. Moreover, we have the following straightforward security theorem:

Theorem 4.14. *If* $\mathsf{WPRF} = (\mathsf{Gen}, \mathsf{F}, \mathsf{Eval})$ *is non-interactively secure for relation R and R-instance generator $\mathcal{D}$, then Construction 4.13 is soundness secure for relation R and R-instance $\mathcal{D}$ (treated as a static instance generator for* WPRF*). Moreover, if* WPRF *is extracting, then so is Construction 4.13.*

We omit the proof, and instead present a stronger variant of witness encryption that we will prove secure.

4.4.1 Reusable Witness Encryption

All current witness encryption schemes, including ours above, have long ciphertexts and relatively inefficient encryption algorithms. This is due to the inefficient setup procedure for the underlying multilinear maps. In this section, we explore the setting where various messages are being witness encrypted to multiple instances, and try to amortize the computation and ciphertext length over many ciphertexts. More precisely, we define a reusable witness key encapsulation mechanism:

Definition 4.15. A private key (resp. public key) witness encryption scheme is a triple of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ where:

- Gen takes as input a security parameter λ and a relation R, and outputs public parameters params as well as a master decryption key dk.

- Enc takes as input an instance x and the parameters params. It outputs a header Hdr and message encryption key k.

- Dec takes as input an instance x, header Hdr, witness w, and parameters params. It outputs a message encryption key k or $\perp$.

- Alternatively, Dec takes as input the master decryption key dk, instance x, and header Hdr (no witness), and outputs k

For correctness, we require for all (Hdr, k) outputted by $\mathsf{Enc}(\mathsf{params}, x)$, and all w such that $R(x, w) = 1$, that $\mathsf{Dec}(\mathsf{params}, x, \mathsf{Hdr}, w) = k$. We also require that $\mathsf{Dec}(\mathsf{dk}, x, \mathsf{Hdr}) = k$.

We observe that from a functionality perspective, if we ignore the master decryption key, witness encryption and reusable witness encryption are equivalent concepts: any witness encryption scheme is a reusable with a Gen algorithm that does nothing but output the security parameter, and any reusable witness encryption scheme can be converted into a regular witness encryption scheme by having the encryption procedure run Gen, and output the public parameters with the ciphertext. However, if the Gen procedure is significantly more inefficient than encryption, or if the public parameters are much longer than the ciphertext, reusable witness encryption will result in less computation and communication that standard witness encryption. Therefore, we focus on building reusable witness encryption where the ciphertext are short and encryption procedures are relatively efficient.

We now give a security definition for reusable witness encryption. Let $\mathcal{D}$ be a PPT algorithm that takes as input parameters $\mathsf{params} \overset{R}{\leftarrow} \mathsf{Gen}(\lambda, R)$, is allowed to make decryption queries $\mathsf{Dec}(\mathsf{sk}, \cdot, \cdot)$, and outputs an instance x^* along with auxilliary information Aux. We call $\mathcal{D}$ an R-instance sampler for WENC.

For any instance sampler $\mathcal{D}$, let $\mathtt{EXP}^R_{\mathcal{D},\mathcal{A}}(b,\lambda)$ denote the following experiment on a PPT algorithm $\mathcal{A}$: run $\mathsf{params} \overset{R}{\leftarrow} \mathsf{Gen}(\lambda, R)$ and $(x^*, \mathsf{Aux}) \overset{R}{\leftarrow} \mathcal{D}(\mathsf{params})$. Let $(\mathsf{Hdr}^*, k_0) \overset{R}{\leftarrow} \mathsf{Enc}(\mathsf{params}, x)$ and $k_1 \overset{R}{\leftarrow} \mathcal{Y}$. Run $b' \overset{R}{\leftarrow} \mathcal{A}^{\mathsf{Dec}(\mathsf{sk},\cdot,\cdot)}(\mathsf{params}, x^*, \mathsf{Aux}, \mathsf{Hdr}^*, k_b)$, with the requirement that the oracle $\mathsf{Dec}(\mathsf{sk},\cdot,\cdot)$ outputs $\perp$ on query (x^*, Hdr^*). If there is a witness w such that $R(x^*, w) = 1$ (in other words, if $x^* \in L$), then output a random bit and abort. Otherwise, output b'.

We define W_b to be the event of outputting 1 in experiment b, and define the advantage of $\mathcal{A}$ to be $\mathtt{WENC.Adv}^R_{\mathcal{D},\mathcal{A}}(\lambda) = |\Pr[W_0] - \Pr[W_1]|$.

Definition 4.16. $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a CCA secure reusable witness encryption scheme for a relation R and instance sampler $\mathcal{D}$ if, for all PPT adversaries $\mathcal{A}$, there is a negligible function negl such that $\mathtt{WENC.Adv}^R_{\mathcal{D},\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.

We can also get a CPA definition is we do not allow $\mathcal{A}$ to make decryption queries[6]. We can similarly get an extractable definition, where we remove the check that $x^* \in L$, and instead have any $\mathcal{A}$ with non-negligible advantage imply an extractor that can find a witness that $x^* \in L$.

Our Construction Our construction of reusable witness encryption is the following:

Construction 4.17. Let $(\mathsf{WPRF.Gen}, \mathsf{F}, \mathsf{Eval})$ be a witness PRF, and let $\mathsf{G} : \mathcal{S} \to \mathcal{Z}$ be a pseudorandom generator with $|\mathcal{S}|/|\mathcal{Z}| < \mathsf{negl}$. Construct the following public key witness encryption scheme $(\mathsf{WENC.Gen}, \mathsf{Enc}, \mathsf{Dec})$:

- $\mathsf{WENC.Gen}(\lambda, R)$: Suppose $R : \mathcal{X} \times \mathcal{W} \to \{0,1\}$. Assume that $\mathcal{S}$ and $\mathcal{W}$ are disjoint. Let $\mathcal{X}' = \mathcal{X} \times \mathcal{Z}$ and $\mathcal{W}' = \mathcal{W} \bigcup \mathcal{S}$. Finally, let $R' : \mathcal{X}' \times \mathcal{W}' \to \{0,1\}$ be the following function:

$$R'((x,z), w') = \begin{cases} R(x,w) & \text{if } w' = w \in \mathcal{W} \\ 1 & \text{if } w' = s \in \mathcal{S} \text{ and } \mathsf{G}(s) = z \\ 0 & \text{if } w' = s \in \mathcal{S} \text{ and } \mathsf{G}(s) \neq z \end{cases}$$

  That is, if w' is a witness for R, R' checks if w is valid for x. Otherwise, w' is a seed for G, and R' checks if the seed generates z.

  Now, run $(\mathsf{fk}, \mathsf{ek}) \overset{R}{\leftarrow} \mathsf{WPRF.Gen}(\lambda, R')$ and output $\mathsf{params} = \mathsf{ek}$ and $\mathsf{dk} = \mathsf{fk}$.

- $\mathsf{Enc}(\mathsf{ek}, x)$: Choose a random seed $s \overset{R}{\leftarrow} \mathcal{S}$, and let $z \leftarrow \mathsf{G}(s)$. Run $k \leftarrow \mathsf{Eval}(\mathsf{ek}, (x,z), s)$. Output z as the header, and k as the message encryption key.

- $\mathsf{Dec}(\mathsf{ek}, x, z, w) = \mathsf{Eval}(\mathsf{ek}, (x,z), w)$

- $\mathsf{Dec}(\mathsf{fk}, x, z) = \mathsf{F}(\mathsf{fk}, (x,z))$

Ciphertext size is $|z| + |m|$. Thus, the ciphertext size is equal to the length of the message plus a term proportional to the security parameter, which is essentially optimal for public key encryption schemes.

[6] If we still consider samplers that can make decryption queries, this notion is similar to CCA1 security for standard public key encryption.

Notice that for $z = \mathsf{G}(s)$, s is always a witness for (x, z) relative to R'. Moreover, if w is a witness for x relative to R, it is also a witness for (x, z) relative to R' for all z. Correctness immediately follows. For security, note that a valid encryption is indistinguishable from an encryption generated by choosing a random $z \stackrel{R}{\leftarrow} \mathcal{Z}$ and computing $k \leftarrow \mathsf{F}(\mathsf{sk}_R, (x, z))$. However, assuming $\mathcal{Z}$ is much bigger than $\mathcal{S}$, with high probability z will not have a pre-image under G. thus, if x has no witness relative to R, (x, z) will have no witness relative to R', meaning k is indistinguishable from random. Security follows. Before proving the statement, we define the following algorithm:

Let $\mathcal{D}_{\mathsf{WENC}}$ be an R-instance sampler for WENC. We construct a R'-instance sampler for WPRF, called $\mathcal{D}_{\mathsf{WPRF}}$, as follows. On input ek, run $\mathcal{D}_{\mathsf{WENC}}$ on $\mathsf{params} = \mathsf{ek}$. When $\mathcal{D}_{\mathsf{WENC}}$ makes a CCA query an instance $x \in \mathcal{X}$ and header $\mathsf{Hdr} = z$, make a F query on (x, z), and respond with the resulting value. When $\mathcal{D}_{\mathsf{WENC}}$ outputs an instance $x^* \in \mathcal{X}$ and auxiliary information Aux, produce a random string $z^* \in \mathcal{Z}$ and output the instance (x^*, z^*) and Aux.

Theorem 4.18. *If* G *is a secure pseudorandom generator and* $\mathsf{WPRF} = (\mathsf{WPRF.Gen}, \mathsf{F}, \mathsf{Eval})$ *is adaptive witness interactively (resp. non-interactively) secure for relation* R' *and instance sampler* $\mathcal{D}_{\mathsf{WPRF}}$*, then Construction 4.17 is a CCA (resp. CPA) secure re-usable witness encryption scheme for relation* R *and* R*-instance sampler* $\mathcal{D}_{\mathsf{WENC}}$*. Moreover, if* WPRF *is extracting, then so is* WENC*.*

Proof. We prove the CCA non-extracting case, the others being similar. Let $\mathcal{B}$ be a CCA adversary for $(\mathsf{WENC.Gen}, \mathsf{Enc}, \mathsf{Dec})$ with non-negligible advantage. We define **Game 0** as the standard attack game, and define **Game 1** as the alternate attack game where y^* is chosen as a random string. If $\mathcal{B}$ can distinguish the two cases, then we could construct an adversary breaking the security of G. Now we use $\mathcal{B}$ to build an adversary $\mathcal{A}$ for WPRF and instance sampler $\mathcal{D}_{\mathsf{WPRF}}$. On input $(\mathsf{ek}, (x^*, z^*), \mathsf{Aux}, k)$, $\mathcal{A}$ simulates $\mathcal{B}$ on input $(\mathsf{ek}, x^*, \mathsf{Hdr} = z^*, \mathsf{Aux}, k)$. Whenever $\mathcal{B}$ makes a CCA query on (x, z), $\mathcal{A}$ makes a F query on (x, z), and forwards the response to $\mathcal{B}$. $\mathcal{A}$ outputs the output of $\mathcal{B}$. Notice that the view of $\mathcal{B}$ is identical to that in **Game 1**. Moreover, with overwhelming probability, z^* is not in the image of G, so (x^*, z^*) is a valid instance for relation R' exactly when x^* is a valid instance for relation R. Therefore, the advantage of $\mathcal{A}$ is negligibly close to the advantage of $\mathcal{B}$, and is therefore non-negligible. $\square$

4.5 Secret Sharing for mNP

We define the notion of re-usable secret sharing for mNP. mNP is the class of *monotone* NP languages, meaning that if $\mathbf{x} \in \{0,1\}^n$ is in L with witness w, and $\mathbf{x}' \in \{0,1\}^n$ is an instance such that $x_i = 1 \Rightarrow x'_i = 1$, then $\mathbf{x}'$ is also in L and w is also a witness for $\mathbf{x}'$. Such languages are characterized by a relation $R : \{0,1\}^n \times \mathcal{W} \to \{0,1\}$ such that there are no NOT gates in any of the paths from the first set of input wires to the output, and $\mathbf{x} \in L$ if and only if there is a w such that $R(\mathbf{x}, w) = 1$.

Intuitively, in re-usable secret sharing for an mNP language L, a trusted party publishes parameters params, which allows anyone to secret share to sets of users in the language L. More precisely, we define the notion of a re-usable secret sharing key encapsulation mechanism.

Definition 4.19. A resuable secret sharing scheme for mNP is a triple of PPT algorithms $(\mathsf{Gen}, \mathsf{Share}, \mathsf{Recon})$ where:

- $\mathsf{Gen}(\lambda, R)$ takes as input a relation accepting n-bit instances, produces a public key params and secret key sk.

- $\mathsf{Share}(\mathsf{params})$ produces shares s_i for user i, and a secret encryption key $k \in \mathcal{Y}$.
- $\mathsf{Recon}(\mathsf{params}, \{s_i\}_{i \in S}, w)$ Outputs either $\perp$ or a key $k \in \mathcal{Y}$. For correctness, we require that if $R(y, x) = 1$ where $y_i = 1$ if and only if $i \in S$, then Recon outputs the correct k, and for $R(y, x) = 0$, Recon outputs $\perp$.

Let $\mathcal{D}$ be an algorithm that, on input security parameter λ, outputs an instance $\mathbf{x} \in \{0,1\}^n$ and auxilliary information Aux. Associate $\mathbf{x}$ with the set $S \subseteq [n]$ where $i \in S$ if and only if $x_i = 1$. We call $\mathcal{D}$ an R-instance sampler. Let $\mathtt{EXP}^R_{\mathcal{D},\mathcal{A}}(b, \lambda)$ be the following experiment on a PPT adversary $\mathcal{A}$: run $(\mathbf{x}, \mathsf{Aux}) \xleftarrow{R} \mathcal{D}(\lambda)$ and $\mathsf{params} \xleftarrow{R} \mathsf{Gen}(\lambda, R)$ and $(\{s_i\}_{i \in [n]}, k_0) \xleftarrow{R} \mathsf{Share}(\mathsf{params})$ and $k_1 \xleftarrow{R} \mathcal{Y}$, and give $\mathcal{A}$ the shares $\{s_i\}_{i \in S}$, Aux and the key k_b. $\mathcal{A}$ produces a guess b'. Let W_b be the event of outputting 1 in experiment b. Define $\mathtt{SS.Adv}^R_{\mathcal{D},\mathcal{A}}(\lambda) = |\Pr[W_0] - \Pr[W_1]|$.

Definition 4.20. $(\mathsf{Gen}, \mathsf{Share}, \mathsf{Recon})$ is secure for a relation R and instance sampler $\mathcal{D}$ if, for all adversaries $\mathcal{A}$, there is a negligible function negl such that $\mathtt{SS.Adv}^R_{\mathcal{D},\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.

Construction. For simplicity, we will assume that for every input length n, the language L contains a string $x \in \{0,1\}^n$ and valid witness w that are both easy to compute. It is straightforward to adapt our scheme to the setting where this is not the case.

Construction 4.21. Let $\mathsf{G} : \mathcal{S} \to \mathcal{Z}$ be a pseudorandom generator, and $(\mathsf{WPRF.Gen}, \mathsf{F}, \mathsf{Eval})$ be a witness PRF.

- $\mathsf{SS.Gen}(\lambda, R,)$: for a mNP relation $R : \{0,1\}^n \times \mathcal{W} \to \{0,1\}$, let $R' : \mathcal{Z}^n \times (\mathcal{S}^n \times \mathcal{W}) \to \{0,1\}$ be the following NP relation: on instance $\{z_i\}_{i \in [n]}$ and witness $\{s_i\}_{i \in [n]}, w$, compute $\mathbf{x} \in \{0,1\}^n$ where $x_i = \begin{cases} 1 & \text{if } \mathsf{G}(s_i) = z_i \\ 0 & \text{if } \mathsf{G}(s_i) \neq z_i \end{cases}$. Then compute $R'(x, w)$.

  Run $(\mathsf{fk}, \mathsf{ek}) \xleftarrow{R} \mathsf{WPRF.Gen}(\lambda, R')$. Output $\mathsf{params} = \mathsf{ek}$.

- $\mathsf{Share}(\mathsf{params})$: Sample $s_i \xleftarrow{R} \mathcal{S}$ for $i \in [n]$ and compute $z_i = \mathsf{G}(s_i)$. Compute some instance $\mathbf{x}$ and witness w for R, and compute $k \leftarrow \mathsf{Eval}(\mathsf{ek}, \{z_i\}_{x_i=1}, w)$. The share for user i is $(s_i, \{z_j\}_{j \in [n]})$, and the secret encryption key is k.

- $\mathsf{Recon}(\mathsf{ek}, \mathbf{x}, \{z_i\}_{i \in [n]}, \{s_i\}_{x_i=1}, w)$: check that $R(\mathbf{x}, w) = 1$ and that $\mathsf{G}(s_i) = z_i$ for each i where $x_i = 1$. For each i where $x_i = 0$, let $s'_i = \perp$, and let $s'_i = s_i$ for all other i. Let $w' = (\{s'_i\}_{i \in [n]}, w)$, and compute $k \leftarrow \mathsf{Eval}(\mathsf{ek}, \{x_i\}_{i \in [n]}, w')$.

Note that the size of a share is $|s| + n|z| \in O(n\lambda)$. However, the $\{z_i\}_{i \in [n]}$ are shared among all users and can therefore be transmitted in a single broadcast. In this way, the amortized share size per user is only $O(\lambda)$.

For security, let $\mathcal{D}_{\mathsf{SS}}$ be an R-instance sampler for $\mathsf{SS} = (\mathsf{SS.Gen}, \mathsf{Share}, \mathsf{Recon})$. Define the following static R'-instance sampler $\mathcal{D}_{\mathsf{WPRF}}$ for WPRF. Run $\mathcal{D}_{\mathsf{SS}}$ to obtain (x, Aux). Let $S \subseteq [n]$ be the set where $i \in S$ if and only if $y_i = 1$. For $i \in S$, sample random $s_i \xleftarrow{R} \mathcal{S}$ and $z_i \leftarrow \mathsf{G}(s_i)$. For all other i, let $z_i \xleftarrow{R} \mathcal{Z}$. Output $(\{z_i\}_{i \in [n]}, \mathsf{Aux}' = (\mathsf{Aux}, \{s_i\}_{i \in S}))$.

Theorem 4.22. *If WPRF is static witness non-interactively secure for relation R' and instance sampler $\mathcal{D}_{\mathsf{WPRF}}$, then SS is secure for relation R and instance sampler $\mathcal{D}_{\mathsf{SS}}$.*

Proof. Let $\mathcal{B}$ be am adversary for SS. Construct the following adversary $\mathcal{A}$ for WPRF. On input $\mathsf{ek}, \{x_i\}_{i\in[n]}, \mathsf{Aux}, \{s_i\}_{i\in S}, k$, $\mathcal{A}$ runs $\mathcal{B}$ on input $\mathsf{ek}, \{x_i\}_{i\in[n]}, \mathsf{Aux}, \{s_i\}_{i\in S}, k$. When $\mathcal{B}$ outputs a bit b', $\mathcal{A}$ outputs b'. Notice that the view of $\mathcal{B}$ as a subroutine of $\mathcal{A}$ is indistinguishable from the correct view of $\mathcal{B}$: the only difference are the x_i for $i \notin S$, which are generated randomly instead of pseudorandomly. Therefore, the advantage of $\mathcal{A}$ is negligible close to the advantage of $\mathcal{B}$, so the security of WPRF implies that they must both be negligible. □

4.6 Distributed Broadcast Encryption

Boneh and Zhandry [BZ13] show that key exchange with small parameters gives a form of distributed broadcast encryption with short ciphertexts. In distributed broadcast encryption, each user generates their own secret key rather than having the secret key generated by a trusted authority. However, their system had large public keys. Ananth et al. [ABG+13] show how to reduce the public key size as well, but at the cost of losing the distributed property of the system. Achieving small public keys for a distributed broadcast scheme seems problematic, as each user must publish some value dependent on their secret key, so the total amount of public data is at least linear in the number of users. Another drawback of the distrubted encryption definition of Boneh and Zhandry is that part of the public key still needs to be computed by a trusted party.

We now put forward the notion of a *fully-distributed broadcast scheme*. In such a scheme, all parties are stateful, and keep a small secret key. There is also a small global public key, posted to some public bulletin board. When a new user joins the system, the user reads off the global public key, and generates their own secret key. Then the user publishes a user public key. All of the existing users use the new user public key to update their secret keys. Finally, the global public key is updated to incorporate the new user. Anyone is able to update the global public key. In this system, there is no a priori bound on the number of users.

Definition 4.23. Fully-dynamic broadcast encryption.

- $\mathsf{Init}(\lambda)$ outputs an initial global public key $\mathsf{params}^{(0)}$.
- $\mathsf{Join}(\mathsf{params}^{(n)})$ generates a user secret key $\mathsf{sk}_{n+1}^{(n+1)}$ and user public key pk_{n+1} for user $n+1$. The user then publishes pk_{n+1}.
- $\mathsf{Update}(\mathsf{sk}_i^{(n)}, \mathsf{pk}_{n+1})$ generates a new user secret key $\mathsf{sk}_i^{(n+1)}$ for user i.
- $\mathsf{Inc}(\mathsf{params}^{(n)}, \mathsf{pk}_{n+1})$ produces an updated global public key $\mathsf{params}^{(n+1)}$.
- $\mathsf{Enc}(\mathsf{params}^{(n)}, S)$ takes as input a subset $S \subseteq [n]$, and produces a header Hdr and message encryption key k.
- $\mathsf{Dec}(\mathsf{sk}_i^{(n)}, S, \mathsf{Hdr})$ checks if $i \in S$, and if so outputs the key k. Otherwise, output $\perp$.

For security, we consider an adaptive notion. In this notion, the adversary can control arbitrary subsets of users, and can adaptively corrupt them. We do not allow the adversary to alter the public key $\mathsf{params}^{(0)}$, except by joining new users to the system. This is a reasonable requirement, as any of the other users could keep a copy of the global public key and always make sure the key is correct and not tampered with.

Consider the following experiment $\mathtt{EXP}_{\mathcal{A}}(b, \lambda)$, played between an adversary $\mathcal{A}$ and a challenger:

- The challenger runs $\mathsf{Init}(\lambda)$ to obtain a global public key $\mathsf{params}^{(0)}$, which it then provides to $\mathcal{A}$. It also initializes a counter n to 0, and a set T to $\{\}$.
- $\mathcal{A}$ can make *register honest* queries on empty input. The challenger runs the following:

$$\begin{aligned}
(\mathsf{sk}_{n+1}^{(n+1)}, \mathsf{pk}_{n+1}) &\xleftarrow{R} \mathsf{Join}(\mathsf{params}^{(n)}) \\
\mathsf{sk}_i^{(n+1)} &\leftarrow \mathsf{Update}(\mathsf{sk}_i^{(n)}, \mathsf{pk}_{n+1}) \text{ for } i \in T \\
\mathsf{params}^{(n+1)} &\leftarrow \mathsf{Inc}(\mathsf{params}^{(n)}, \mathsf{pk}_{n+1}) \\
T &\leftarrow T \bigcup \{n+1\} \\
n &\leftarrow n+1
\end{aligned}$$

  The challenger then supplies the new public key $\mathsf{params}^{(n+1)}$ and the user public key $\mathsf{pk}_{n+1}^{(n+1)}$ to $\mathcal{A}$

- $\mathcal{A}$ can make *register corrupt* queries on input pk_{n+1}. The challenger runs the following:

$$\begin{aligned}
\mathsf{sk}_i^{(n+1)} &\leftarrow \mathsf{Update}(\mathsf{sk}_i^{(n)}, \mathsf{pk}_{n+1}) \text{ for } i \in T \\
\mathsf{params}^{(n+1)} &\leftarrow \mathsf{Inc}(\mathsf{params}^{(n)}, \mathsf{pk}_{n+1}) \\
n &\leftarrow n+1
\end{aligned}$$

  The challenger then supplies the new public key $\mathsf{params}^{(n+1)}$ to the adversary.

- $\mathcal{A}$ can make *corrupt user* queries on input $i \in T$. The challenger responds by setting $T \leftarrow T \setminus \{i\}$, and giving $\mathsf{sk}_i^{(n)}$ to the adversary.
- $\mathcal{A}$ can make a single *challenge query* on target set $S \subseteq T$. The challenger then computes $(\mathsf{Hdr}^*, k_0) \xleftarrow{R} \mathsf{Enc}(\mathsf{params}^{(n)}, S)$, and lets $k_1 \xleftarrow{R} \mathcal{K}$. The challenger gives $\mathcal{A}$ the pair $(\mathsf{Hdr}^*, k^* = k_b)$.
- Finally, $\mathcal{A}$ outputs a guess b' for b.

We define W_b to be the event of outputting 1 in experiment b, and define the advantage to be $\mathtt{BE.Adv}_{\mathcal{A}}(\lambda) = |\Pr[W_0] - \Pr[W_1]|$.

Definition 4.24. A fully-distributed broadcast encryption scheme is adaptively secure if, for all adversaries $\mathcal{A}$, there is a negligible function negl such that $\mathtt{BE.Adv}_{\mathcal{A}}(\lambda) < \mathsf{negl}(\lambda)$.

Our Construction The idea behind our construction is as follows. Each user of the system will generate a random input s to a one-way function f, and publish the corresponding output z. Clearly, if the public parameters contained all published outputs, the parameters would be linear in the size of the users. Instead, similar to the scheme of Ananth et al. [ABG+13], we use a Merkle hash tree to hash down the public parameters to a small hash. In particular, we divide the n users into at most $\lceil \log n \rceil$ groups S_j where $|S_j| = 2^j$. For each group, we compute the Merkle tree hash of the public values of that group. The public parameters are then the hashes h_j for each group G_j. The secret key for a user in S_j will be their random s, as well as a "proof" that the corresponding output x was one of the leaves in the hash tree for h_j. The proof will consist of the nodes in the path from x to h_j in the Merkle tree, as well as all of the neighbors. Any false proof will lead directly to a collision for the underlying hash function.

Adding a user is simple: the user computes a random input s to f and publishes the output x. Add x as a new hash to the public parameters. As long as there are two hashes corresponding to Merkle trees of the same height, merge the two together by hashing their roots, and replacing the two values with the new hash. Merge the corresponding groups together as well. This can all be done publicly.

If a user belongs to a group that was merged, it is easy to update their proof by merging the path to the old root with the path from the old root to the new root.

We now give the construction in more detail:

Construction 4.25. Let $f : \mathcal{S} \to \mathcal{X}$ be a one-way function, $(\mathsf{Gen}, \mathsf{F}, \mathsf{Eval})$ be a witness PRF, and $H : \mathcal{X}^2 \to \mathcal{X}$ a collision-resistant hash function.

- $\mathsf{Init}(\lambda)$: initializes an empty list $L = ()$, and outputs $\mathsf{params}^{(0)} = (\lambda, n = 0, L)$. n will be the number of users, and L will be a list of hashes. Let $n = \sum_{j \in T} 2^j$ where $T \subseteq [0, \lfloor \log n \rfloor]$. Then $[n]$ will be divided into $|T|$ different sets $S_j, j \in T$, where $|S_j| = 2^j$ (the sets T and S_j can be inferred from n and do not need to be stored in the public parameters). L will contain a hash h_j, which will be the Merkle tree hash using H of the public values of the users in S_j.

- $\mathsf{Join}(\mathsf{params}^{(n)})$: $s \xleftarrow{R} \mathcal{S}$, $x \leftarrow f(s)$. Publish $\mathsf{pk}_{n+1} = x$. Create a local copy of L, and let j_{min} be the minimum non-negative integer not in T. Define $h_{j,L} = h_j$ for $j = 0, ..., j_{min} - 1$. Create a local tree π with $h_{0,R} = x$ initially as the root. For $j = 0, ..., j_{min} - 1$, create a new root $h_{j+1,R} = H(h_{j,L}, h_{j,R})$ with $h_{j,L}$ as the left child and $h_{j,R}$ as the right child. Mark x as the "target leaf." Note that π consists of a single path between the "target leaf" and the root, plus a sibling for every non-root node. We call such a tree a *rooted binary caterpillar* (RBC) tree. Define $h_{j_{min}} = h_{j_{min},R}$, the root of the RBC tree.

  Set $T = (T \setminus \{0, ..., j_{min} - 1\}) \bigcup \{j_{min}\}$. Remove all hashes h_j for $j = 0, \ldots, j_{min} - 1$ from L, and add the hash $h_{j_{min}}$. The secret key for user $n + 1$ is L, π, x, s.

- $\mathsf{Update}(\mathsf{sk}_i^{(n)}, \mathsf{pk}_{n+1})$: Update the local copy of L as in Join, obtaining an RBC tree π. If the root of π_i (the current tree for user i) became one of the leaves on π, prune the opposite branch in π, and then merge the two trees together to obtain a new RBC tree π_i. The target leaf of π_i is still the original target leaf from π_i.

- $\mathsf{Inc}(\mathsf{params}^{(n)}, \mathsf{pk}_{n+1})$: Update L as in Join, and then discard the tree π.

- $\mathsf{Enc}(\mathsf{params}^{(n)}, S)$: Let $\ell = \lfloor \log n \rfloor$. Define $R_{\leq \ell}$ to be the relation where instances are tuples $(j, h_j, h_S^{(j)})$ with $j \leq \ell$. A witness for (j, h_j, h_S) is a tuple (π, τ, s) where π, τ are RBC trees of height j with identical structure (including the same target leaf) such that the root of π is h_j, the root of τ is h_S, the target leaf of π is $f(s)$, and the target leaf of h_S is 1.

  Choose a random key k. For each $j \in T$, do the following:

  - Run $(\mathsf{fk}_j, \mathsf{ek}_j) \xleftarrow{R} \mathsf{Gen}(\lambda, R_{\leq \ell})$.

– Let $\mathbf{v}^{(j)} \in \mathcal{X}^{2^j}$ be the vector defined as follows: iterate through the 2^j identities $i \in S_j$, and set $\mathbf{v}_i^{(j)} = 1$ if $i \in S$ and 0 otherwise.
– Let $h_S^{(j)}$ be the Merkle tree hash of $\mathbf{v}^{(j)}$ computed by successively hashing pairs of elements together.
– Let $K_j = k \oplus \mathsf{F}(\mathsf{fk}_j, (j, h_j, h_S^{(j)})$.

Set k to be the message encryption key, and the header to be $\mathsf{Hdr} = \{j, K_j, \mathsf{ek}_j\}_{j \in T}$

- $\mathsf{Dec}(\mathsf{sk}_i^{(n)}, S, \mathsf{Hdr})$: If $i \notin S$, output $\perp$. Otherwise, let $i \in S_j$ for some $j \in T$. Compute the Merkle hash $h_S^{(j)}$ of $\mathbf{v}^{(j)}$ as above. Next prune the tree down to a tree τ_i that has structure is identical to π_i, including the target leaf at position i. If $i \in S$, the the target leaf will have value 1. Next, run $k \leftarrow K_j \oplus \mathsf{Eval}(\mathsf{ek}_j, (j, h_j, h_S^{(j)}), (\pi_i, \tau_i, s))$.

Correctness is immediate from the construction. Also notice that the list L contains at most ℓ pairs, so the total size ignoring the security parameter is $O(\log n)$. The secret key for a user contains L, as well as a BCP tree π. π has depth at most ℓ, and has at most two nodes per level. therefore, the size is $O(\log n)$. Finally, each header component contains an evaluation key ek_j for a witness PRF for relation $R_{\leq \ell}$ where $\ell \leq \log n$. The size of $R_{\leq \ell}$ is polynomial in ℓ, so the size of ek_j is $\mathsf{poly}(\log n)$. In addition, there are at most ℓ header components, so the total header size is $\mathsf{poly}(\log n)$.

For security, we define a class of instance samplers, called *acceptable*, such that the instance $(j, h_j, h_S^{(j)})$ and auxilliary information Aux computed by $\mathcal{D}$ satisfies the following:

- h_j is computed as the Merkle tree hash of some $x_i \in \mathcal{X}$ for $i \in [2^j]$ for some j.
- Each of the x_i above are computed as $x_i = f(s_i)$ for some $s_i \in \mathcal{S}$
- $h_S^{(j)}$ is computed as the Merkle tree hash of some $\mathbf{v}^{(j)} \in \mathcal{X}^{2^j}$ where $v_i^{(j)}$ is either 0 or 1 for all j.
- For each i where $v_i^{(j)} = 1$, Aux does not depend on s_i (though it may be computed from x_i).

This class of instance samplers is very restricting. In particular, Aux is computed independently of any of the "easy-to-compute" witnesses for $(j, h_j, h_S^{(j)})$ (that is, those witness containing the correct proofs, and not false proofs that lead to collisions). This makes it unlikely the counter example of [GGHW13] can be adapted to apply to this setting.

Theorem 4.26. *Suppose H is collision resistant, f is one-way, and* $(\mathsf{Gen}, \mathsf{F}, \mathsf{Eval})$ *is an extracting non-interactive witness PRF for all static acceptable instance samplers $\mathcal{D}$. Then Construction 4.25 is adaptively secure.*

We note that if we require interactive security for WPRF, we can modify Construction 4.25 so that all header components use a single witness PRF key, and security will be maintained. This will save roughly a $\log n$ factor in the length of the ciphertexts.

We now prove Theorem 4.26:

Proof. Let $\mathcal{B}$ be an adaptive adversary for this scheme. Taking only a polynomial hit in the security parameter, we can assume without loss of generality that there is some polynomial p such that $\mathcal{B}$ always makes the challenge query when exactly $p(\lambda)$ users are registered. We break $\mathcal{B}$ into two parts: $\mathcal{B}_0$, which stops after submitting the challenge query and returns some state state, and $\mathcal{B}_1$ which takes as input state, the challenge set S, the challenge header Hdr^* and the challenge key k^*, and ultimately outputs a bit b'.

Let $\mathcal{D}$ be the following instance sampler. $\mathcal{D}$ simulates $\mathcal{B}_0$, and plays the role of challenger to $\mathcal{B}_0$. When $\mathcal{B}_0$ outputs a challenge set S, $\mathcal{D}$ picks a random $j \xleftarrow{R} L$, computes $h_S^{(j)}$ and h_j as above. It outputs the instance $(j, h_j, h_S^{(j)})$, and the auxiliary information $\mathsf{Aux} = \mathsf{state}$. We assume that state contains all of the queries made by $\mathcal{B}_0$. We also assume that, before the challenge query on S, $\mathcal{B}_0$ has asked for all of the secret keys for users outside of S. We note that $\mathcal{D}$ is static and acceptable.

Let $\mathcal{A}$ be the following adversary for $(\mathsf{Gen}, \mathsf{F}, \mathsf{Eval})$, derived from $\mathcal{B}_1$. On input the tuple $(j, h_j, h_S^{(j)}), \mathsf{state}, \mathsf{ek}, K$, $\mathcal{A}$ chooses a random $k \xleftarrow{R} \mathcal{K}$, and sets $K_j = k \oplus K$ and $\mathsf{ek}_j = \mathsf{ek}$. For all $j' \in T \setminus \{j\}$, run $(\mathsf{fk}_{j'}, \mathsf{ek}_{j'}) \xleftarrow{R} \mathsf{Gen}(\lambda, R_{\leq \ell})$. If $j' < j$, set $\mathsf{Hdr}_{j'} = (K_{j'}, \mathsf{ek}_{j'})$ where $K_{j'} \xleftarrow{R} \mathcal{K}$. For all tuples with $j' > j$, compute $K_{j'} \leftarrow k \oplus \mathsf{F}(\mathsf{fk}_{j'}, (h_{j'}, h_S^{(j')}))$ where $h^{j'}$ and $h_S^{(j')}$ are computed as above. Then it provides $\mathcal{A}_1$ the header $\{K_{j'}, \mathsf{ek}_{j'}\}$, state state, and key k. Then $\mathcal{A}$ runs $\mathcal{B}_1$, playing the role of challenger to $\mathcal{B}_1$.

To analyze the advantage of $\mathcal{A}$, we define the following j hybrids. In hybrid h_j, $K_{j'}$ for $j' \leq j$ are chosen randomly for the challenge header, whereas $K_{j'}$ for $j' > j$ are chosen correctly. Then Hybrid 0 corresponds to experiment 0, and hybrid ℓ corresponds to experiment 1.

Let j be the value chosen by $\mathcal{D}$. Then, in experiment 0, $\mathcal{A}$ perfectly simulates the view of $\mathcal{B}$ in Hybrid $j-1$. Meanwhile, in experiment 1, $\mathcal{A}$ perfectly simulates the view of $\mathcal{B}$ in Hybrid j. Therefore, it is straightforward to show that the advantage of $\mathcal{A}$ is at least $1/\ell$ times the advantage of $\mathcal{B}$.

If $\mathcal{B}$ had non-negligible advantage, this implies that $\mathcal{A}$ has non-negligible advantage. But then there is an extractor $\mathcal{E}$ that, on input $(j, h_j, h_S^{(j)}), \mathsf{state}, \mathsf{ek}$, is able to find a witness for $j, h^j, h^{(j)}$. We can use such a witness $w = (\pi, \tau, s)$ to break the collision resistance of H or the one-wayness of f. Let $\mathbf{v}^{(j)}$ be the values that $\mathcal{D}$ hashed to obtain h^j. Let π' be the result of pruning the Merkle tree for h^j down to the same structure as π. Let τ' be defined similarly. There are several cases:

- π and π' have different values. This means, π and π' contain a collision for H.
- τ and τ' have different values. This means τ and τ' contain a collision for H.
- $f(s)$ is equal to the target leaf in π. Given that neither of the above holds, this means s is a pre-image of one of the user public keys.

The collision resistance of H implies that the first two cases above happen with at most negligible probability. Moreover, it is straightforward to use the third case above to invert f, meaning the third case occurs with negligible probability. This means the extractor must actually succeed with only non-negligible probability, a contradiction. □

5 An Abstraction: Subset-Sum Encoding

Now that we have seen many applications of witness PRFs, we begin our construction. In this section, we give an abstraction of functionality we need from multilinear maps. Our abstraction is called a *subset-sum* encoding. Roughly, a subset sum encoding is a way to encode vectors $\mathbf{t}$ such that (1) the encoding of $\mathbf{t} = \mathbf{A} \cdot \mathbf{w}$ for $\mathbf{w} \in \{0,1\}^n$ is efficiently computable given $\mathbf{w}$ and (2) the encoding of $\mathbf{t} \notin \mathsf{SubSums}(\mathbf{A})$ is indistinguishable from a random string. More formally, a subset-sum encoding is the following:

Definition 5.1. A *subset-sum encoding* is a triple of efficient algorithms $(\mathsf{Gen}, \mathsf{Encode}, \mathsf{Eval})$ where:

- Gen takes as input a security parameter λ and an integer matrix $\mathbf{A} \in \mathbb{Z}^{m \times n}$, and outputs an encoding key sk and an evaluation key ek.
- Encode takes as input the secret key sk vector $\mathbf{t} \in \mathbb{Z}^m$, and produces an encoding $\hat{\mathbf{t}} \in \mathcal{Y}$. Encode is deterministic.
- Eval takes as input the encoding key ek and a bit vector $\mathbf{w} \in \{0,1\}^n$, and outputs a value $\hat{\mathbf{t}}$ satisfying $\hat{\mathbf{t}} = \mathsf{Encode}(\mathsf{sk}, \mathbf{t})$ where $\mathbf{t} = \mathbf{A} \cdot \mathbf{w}$.

Security Notions. The security notions we define for subset-sum encodings are very similar to those for witness PRFs. Consider the following experiment $\mathtt{EXP}_{\mathcal{A}}^{\mathbf{A}}(b, \lambda)$ between an adversar $\mathcal{A}$ and challenger, parameterized by a matrix $\mathbf{A} \in \mathbb{Z}^{m \times n}$, a bit b, and a security parameters λ:

- Run $(\mathsf{sk}, \mathsf{ek}) \xleftarrow{R} \mathsf{Gen}(\lambda, \mathbf{A})$, and give ek to $\mathcal{A}$
- $\mathcal{A}$ can adaptively make queries on targets $\mathbf{t}_i \in \{0,1\}^m$, to which the challenger responds with $\hat{\mathbf{t}_i} \leftarrow \mathsf{Encode}(\mathsf{sk}, \mathbf{t}_i) \in \mathcal{Y}$.
- $\mathcal{A}$ can make a single challenge query on a target $\mathbf{t}^*$. The challenger computes $y_0 = \hat{\mathbf{t}^*} \leftarrow \mathsf{Encode}(\mathsf{sk}, \mathbf{t}^*)$ and $y_1 \xleftarrow{R} \mathcal{Y}$, and responds with y_b.
- After making additional Encode queries, $\mathcal{A}$ produces a bit b'. The challenger checks that $\mathbf{t}^* \notin \{\mathbf{t}_i\}$ and $\mathbf{t}^* \notin \mathsf{SubSums}(\mathbf{A})$. If either check fails, the challenger outputs a random bit. Otherwise, it outputs b'.

Define W_b as the event the challenger outputs 1 in experiment b. Let $\mathtt{SS.Adv}_{\mathcal{A}}^{\mathbf{A}}(\lambda) = |\Pr[W_0] - \Pr[W_1]|$.

Definition 5.2. $(\mathsf{Gen}, \mathsf{Encode}, \mathsf{Eval})$ is *adaptive target interactively secure* for a matrix $\mathbf{A}$ if, for all adversaries $\mathcal{A}$, there is a negligible function negl such that $\mathtt{SS.Adv}_{\mathcal{A}}^{\mathbf{A}}(\lambda) < \mathsf{negl}(\lambda)$.

We can also define a weaker notion of *static target* security where $\mathcal{A}$ commits to $\mathbf{t}^*$ before seeing ek or making any Encode queries. Independently, we can also define *non-interactive* security where the adversary is not allowed to make any Encode queries.

Fine-grained security notions. Similar to witness PRFs, it is straight forward to define fine-grained security notions, where security holds relative to a specific instance sampler. We omit the details.

5.1 A simple instantiation from multilinear maps.

We now construct subset-sum encodings from asymmetric multilinear maps.

Construction 5.3. Let Setup be the generation algorithm for an asymmetric multilinear map. We build the following subset-sum encoding:

- $\mathsf{Gen}(\lambda, \mathbf{A})$: on input a matrix $\mathbf{A} \in \mathbb{Z}^{m \times n}$, let $B = \|\mathbf{A}\|_\infty$, and $p_{min} = 2nB + 1$. Run $\mathsf{params} \xleftarrow{R} \mathsf{Setup}(\lambda, n, p_{min})$ to get the description of a multilinear map $e : \mathbb{G}_1 \times \cdots \times \mathbb{G}_n \to \mathbb{G}_T$ on groups of prime order p, together with generators $g_1, \ldots, g_m, g_T$. Choose random $\boldsymbol{\alpha} \in (\mathbb{Z}_p^*)^m$. Denote by $\boldsymbol{\alpha}^{\mathbf{v}}$ the product $\prod_{i \in [m]} \alpha_i^{v_i}$ (since each component of $\boldsymbol{\alpha}$ is non-zero, this operation is well-defined for all integer vectors $\mathbf{v}_i$). Let $V_i = g_i^{\boldsymbol{\alpha}^{\mathbf{v}_i}}$ where $\mathbf{v}_i$ are the columns of $\mathbf{A}$. Publish $\mathsf{ek} = (\mathsf{params}, \{V_i\}_{i \in [n]})$ as the public parameters and $\mathsf{sk} = \boldsymbol{\alpha}$
- $\mathsf{Encode}(\mathsf{sk}, \mathbf{t}) = g_T^{\boldsymbol{\alpha}^{\mathbf{t}}}$, where $\mathbf{t} \in \mathsf{IntRange}(\mathbf{A})$.
- $\mathsf{Eval}(\mathsf{ek}, \mathbf{w}) = e(V_1^{w_1}, V_2^{w_2}, \ldots, V_n^{w_n})$ where we define $V_i^0 = g_i$

For correctness, observe that

$$e(v_1^{w_1}, v_2^{w_2}, \ldots, V_n^{w_n}) = e(g_1^{\boldsymbol{\alpha}^{vv_1 w_1}}, \ldots, g_n^{\boldsymbol{\alpha}^{vv_n w_n}}) = g_T^{\boldsymbol{\alpha}^{\sum_{i \in [n]} \mathbf{v}_i w_i}} = g_T^{\boldsymbol{\alpha}^{\mathbf{A} \cdot \mathbf{w}}} = \mathsf{Encode}(\mathsf{sk}, \mathbf{A} \cdot \mathbf{w})$$

Security. We assume the security of our subset-sum encodings, which translates to a new security assumption on multilinear maps, which we call the *(adaptive target interactive) multilinear subset-sum Diffie Hellman assumption.* For completeness, we formally define the assumption as follows. Let $\mathtt{EXP}_{\mathcal{A}}^{\mathbf{A}}(b, \lambda)$ be the following experiment between an adversary $\mathcal{A}$ and challenger, parameterized by a matrix $\mathbf{A} \in \mathbb{Z}^{m \times n}$, a bit b, and a security parameter λ:

- Let $B = \|\mathbf{A}\|_\infty$, and $p_{min} = 2nB + 1$. Run $\mathsf{params} \xleftarrow{R} \mathsf{Setup}(\lambda, n, p_{min})$.
- Choose a random $\boldsymbol{\alpha} \in \mathbb{Z}_p^m$, and let $V_i = g_i^{\boldsymbol{\alpha}^{\mathbf{v}_i}}$ where $\mathbf{v}_i$ are the columns of $\mathbf{A}$. Give $(\mathsf{params}, \{V_i\}_{i \in [n]})$ to $\mathcal{A}$.
- $\mathcal{A}$ can make oracle queries on targets $\mathbf{t}_i \in \mathsf{IntRange}(\mathbf{A})$, to which the challenger responds with $g_T^{\boldsymbol{\alpha}^{\mathbf{t}_i}}$.
- $\mathcal{A}$ can make a single challenge query on a target $\mathbf{t}^* \in \mathsf{IntRange}(\mathbf{A})$. The challenger computes $y_0 = g_T^{\boldsymbol{\alpha}^{\mathbf{t}^*}}$ and $y_1 = g_T^r$ for a random $r \xleftarrow{R} \mathbb{Z}_p$, and responds with y_b.
- After making additional Encode queries, $\mathcal{A}$ produces a bit b'. The challenger checks that $\mathbf{t}^* \notin \{t_i\}$ and $t^* \notin \mathsf{SubSums}(\mathbf{A})$. If either check fails, the challenger outputs a random bit. Otherwise, it outputs b'.

Define W_b as the event that the challenger outputs 1 in experiment b. Let $\mathtt{SSDH.Adv}_{\mathcal{A}}^{\mathbf{A}}(\lambda) = |\Pr[W_0] - \Pr[W_1]|$.

Definition 5.4. The adaptive target interactive multilinear subset-sum Diffie Hellman (SSDH) assumption holds relative to Setup if, for all adversaries $\mathcal{A}$, there is a negligible function negl such that $\mathtt{SSDH.Adv}_{\mathcal{A}}^{\mathbf{A}}(\lambda) < \mathsf{negl}(\lambda)$.

Security of our subset-sum encodings immediately follows from the assumption:

Fact 5.5. *If the adaptive target interactive multilinear SSDH assumptions holds for* Setup*, the Construction 5.3 is an adaptive target interactively secure subset-sum encoding.*

We can also consider fine-grained SSDH assumptions and obtain the corresponding fine-grained security notions:

Fact 5.6. *If the (extracting) interactive/non-interactive subset-sum Diffie-Hellman assumption holds relative to* Setup *for a matrix* $\mathbf{A}$ *and an instance sampler* $\mathcal{D}$*, then* $(\mathsf{Gen}, \mathsf{Encode}, \mathsf{Eval})$ *is (extracting) interactively/non-interactively secure for matrix* $\mathbf{A}$ *and instance sampler* $\mathcal{D}$*.*

Flattening The Encodings We can convert any subset-sum encoding for $m = 1$ into a subsetsum encoding for any m. Let $\mathbf{A} \in \mathbb{Z}^{m \times n}$ and define $B = \|\mathbf{A}\|_\infty$. Then, for any $\mathbf{w} \in \{0,1\}^n$, $\|\mathbf{A} \cdot \mathbf{w}\|_\infty \leq nB$. Therefore, we can let $\mathbf{A}' = (1, nB+1, (nB+1)^2, \ldots, (nB+1)^{m-1}) \cdot \mathbf{A}$ be a single row, and run $\mathsf{Gen}(\lambda, \mathbf{A}')$ to get $(\mathsf{sk}, \mathsf{ek})$. To encode an element $\mathbf{t}$, compute $\mathbf{t}' = (1, nB, (nB)^2, \ldots, (nB)^{m-1}) \cdot \mathbf{t}$, and encode $\mathbf{t}'$. Finally, to evaluate on vector $\mathbf{w}$, simply run $\mathsf{Eval}(\mathsf{ek}, \mathbf{w})$.

Security translates since left-multiplying by $(1, nB, (nB)^2, \ldots, (nB)^{m-1})$ does not introduce any collisions. Therefore, we can always rely on subset-sum encodings, and thus the subset-sum Diffie-Hellman assumption, for $m = 1$. However, we recommend *not* using this conversion for two reasons:

- To prevent the exponent from "wrapping" mod $p-1$, $p-1$ needs to be larger than the maximum L_1-norm of the rows of $\mathbf{A}$. In this conversion, we are multiplying rows by exponential factors, meaning p needs to correspondingly be set much larger.
- In Appendix A, we prove the security of our encodings in the generic multilinear map model. Generic security is only guaranteed if $\|\mathbf{A}\|_\infty / p$ is negligible. This means for security, p will have to be substantially larger after applying the conversion.

5.2 Limited Witness PRFs

We note that subset-sum encodings immediately give us witness PRFs for restricted classes. In particular, for a matrix $\mathbf{A}$, a subset-sum encoding is a witness PRF for the language $\mathsf{SubSums}(\mathbf{A})$. The various security notions for subset-sum encodings correspond exactly to the security notions for witness PRFs.

Our goal in the next section is to turn this into a witness PRF for any language L. In essence, we provide a reduction from an instance x to a subset-sum instance $\mathbf{A}, \mathbf{t}$, where $\mathbf{A}$ is determined entirely by the relation R defining L, and is independent of x (except for its length). Thus, $\mathsf{SubSums}(\mathbf{A})$ corresponds exactly with L. This means the subset-sum encoding for $\mathbf{A}$ is actually a witness PRF for L.

6 Witness PRFs from Subset-Sum Encodings

We now show how to build witness PRFs from secure subset-sum encodings, completing our construction.

6.1 Verifying Computation as Subset Sum

Given a circuit C and strings $\mathbf{x}, \mathbf{y}$, we would like to prove that $C(\mathbf{x}) = \mathbf{y}$. However, we want to restrict the verification algorithm to be a subset-sum instance, where $(\mathbf{x}, \mathbf{y})$ is a satisfying assignment only if $C(\mathbf{x}) = \mathbf{y}$. More formally, we desire the following procedures:

- $\mathsf{Convert}(C)$ takes as input a circuit of g gates mapping in bits to out bits, and outputs a matrix $\mathbf{C} \in \mathbb{Z}^{m \times (out+in+r)}$ and vector $\mathbf{b} \in \mathbb{Z}^m$, plus some parameter params_C, where m, r and $B = \|\mathbf{A}\|_\infty$ are polynomial in g, in, out. We call $\mathbf{C}, \mathbf{b}$ an enforcer for C.
- $\mathsf{Prove}(\mathsf{params}_C, \mathbf{x}, \mathbf{y})$ constructs a proof $\boldsymbol{\pi} \in \{0,1\}^r$ such that

$$\mathbf{C} \cdot \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \\ \boldsymbol{\pi} \end{pmatrix} = \mathbf{b}$$

We require two properties: completeness, which means that Prove always succeeds, and soundness, which says that if $\mathbf{y} \neq C(\mathbf{x})$, then for all proofs $\boldsymbol{\pi} \in \{0,1\}^r$,

$$\mathbf{C} \cdot \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \\ \boldsymbol{\pi} \end{pmatrix} \neq \mathbf{b}$$

Our construction is relatively straightforward. We first describe the scheme for single gate circuits, and then show how to piece together many gates to form a circuit.

- AND gates: let $\mathbf{C}_{\mathsf{AND}}$ consist of a single row $(-2, 1, 1, -1)$ and $\mathbf{b}_{\mathsf{AND}} = 0$. Prove, on input $\mathbf{x} = (x_1, x2)$ and $y = x_1 \mathsf{AND} x_2 = x_1 x_2$ computes $\pi = x_1 \mathsf{XOR} x_2 = x_1 + x_2 - 2x_1x_2$. Observe that
$$\mathbf{C}_{\mathsf{AND}} \cdot (y, x_1, x_2, \pi) = -2x_1x_2 + x_1 + x_2 - (x + 1 + x_2 - 2x_1x_2) = 0 = \mathbf{b}$$
For $y = \mathsf{NOT}(x_1 \mathsf{AND} x_2) = 1 - x_1x_2$ and any $\pi \in \{0,1\}$,
$$\mathbf{C}_{\mathsf{AND}} \cdot (y, x_1, x_2, \pi) = -2 + 2x_1x_2 + x_1 + x_2 - \pi = -2 + 4x_1x_2 + (x_1 \mathsf{XOR} x_2) - \pi$$
$-2 + 4x_1x_2 \in \{-2, 2\}$ and $x_1 \mathsf{XOR} x_2 \in \{0,1\}$, so $-2 + 4x_1x_2 + (x_1 \mathsf{XOR} x_2) \in \{-2, -1, 2, 3\}$. Therefore, no setting of $\pi \in \{0,1\}$ will cause $\mathbf{C}_{\mathsf{AND}} \cdot (y, x_1, x_2, \pi)$ to evaluate to 0.
- OR gates: let $\mathbf{C}_{\mathsf{OR}}$ consist of a single row $(-2, 1, 1, 1)$ and $\mathbf{b}_{\mathsf{OR}} = 0$. Prove, on input $\mathbf{x} = (x_1, x_2)$ and $y = x_1 \mathsf{OR} x_2 = x_1 + x_2 - x_1x_2$ computes $\pi = x_1 \mathsf{XOR} x_2 = x_1 + x_2 - 2x_1x_2$. Observe that
$$\mathbf{C}_{\mathsf{OR}} \cdot (y, x_1, x_2, \pi) = -2(x_1 + x_2 - x_1x_2) + x_1 + x_2 + (x_1 + x_2 - 2x_1x_2) = 0 = \mathbf{b}$$
Similarly, for $y = \mathsf{NOT}(x_1 \mathsf{OR} x_2)$, it is straightforward to show that $\mathbf{C}_{\mathsf{OR}} \cdot (y, x_1, x_2, \pi) \neq 0$ for all $\pi \in \{0,1\}$.
- NOT gates: let $\mathbf{C}_{\mathsf{NOT}}$ consist of a single row $(1 \;\; 1)$ and $\mathbf{b}_{\mathsf{NOT}} = 1$. Prove is trivial and outputs nothing. Verifying the correctness is straightforward.
- PASS gates: a pass gate is a fan-in one gate that just outputs its input. This gate will be useful for our construction, and we give two implementations:

- The *simple* pass gate. Let $\mathbf{C}_{\mathsf{PASS},0}$ consist of a single row $(1,-1)$ and $\mathbf{b}_{\mathsf{PASS},0} = 0$. Prove is trivial and outputs nothing. Verifying the correctness is straightforward. This is more direct than implementing the pass gate as two not gates.
- The *cheating* pass gate. Let $\mathbf{C}_{\mathsf{PASS},1}$ consist of a single row $(1,1,-2)$ and $\mathbf{b}_{\mathsf{PASS},1} = 0$. Prove, on input x and $y = x$ computes $\pi = x$. Correctness is straightforward. Notice that for this gate, if we allow π to be a real number in $[0,1]$, then we can can set $\pi = 1/2$ and $y = \mathsf{NOT}x$ and "prove" the wrong output. This will be important for our construction.

- Other fan-in two gates: it is straightforward to build all fan-in two gates using the above ideas. Each gate requires just a single row in $\mathbf{C}$, and a single proof bit π. We omit the details. The advantage of implementing all fan-in two gates is that we can absorb NOT gates into the fan-in two gates, and therefore get NOT gates for free.

To handle arbitrary circuits, we evaluate the circuit gate by gate. For each gate, we assign a row of $\mathbf{C}$ which will enforce that the output of that row is correct using the above single-gate enforcers. We will also assign at most two columns, one for the actual result, and possibly one column for the proof. For any invalid computation with potential proof $\boldsymbol{\pi}$, $\boldsymbol{\pi}$ will give an assignment to the wires. Since the result is incorrect, there must be some gate where the input wires are correct, but the output wire is incorrect. Assume the gate has two inputs, the one input gate being similar. For this gate, look at the corresponding row of $\mathbf{C}$, the two input columns, the output column, and the proof column. Also look at the row in $\mathbf{b}$. Also look at the restriction of $\boldsymbol{\pi}$ to these four components. This will correspond to a single gate enforcer, and the invalid proof $\boldsymbol{\pi}$ gives an invalid result, which we know to be impossible.

6.2 Our Construction

We now give our construction.

Construction 6.1. Let $(\mathsf{SSE.Gen}, \mathsf{SSE.Encode}, \mathsf{SSE.Eval})$ be a secure subset-sum encoding.

- $\mathsf{WPRF.Gen}(\lambda, R)$: On relation $R : \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}$ consisting of g gates, let $R'(x,w) = (x, R(x,w))$. We can implement R' using the g gates of R, as well as k simple pass gates for the x part of the output. We will also add a cheating pass gate on the output of R. Therefore, we can compute $(\mathbf{C}, \mathsf{params}_R) \leftarrow \mathsf{Convert}(R)$ using $\mathsf{Convert}$ from above. Notice that $\mathbf{C} \in \mathbb{Z}^{(k+g+1)\times(2k+g+\ell+2)}$.

  Now, rearrange the columns of $\mathbf{C}$ so that:

  - $\mathbf{C} = (\mathbf{B}, \mathbf{A})$ where $\mathbf{B} \in \mathbb{Z}^{(k+g+1)\times(k+1)}$ and $\mathbf{A} \in \mathbb{Z}^{(m+g+1)\times(m+g+n+1)}$
  - On input $(\mathbf{x}, \mathbf{w})$ and output $b = R(\mathbf{x}, \mathbf{w})$, Prove produces a vector $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1) = ((\mathbf{x}, b), (\mathbf{x}, \mathbf{w}, \boldsymbol{\pi}))$ such that

  $$\mathbf{C} \cdot \mathbf{v} = \mathbf{B} \cdot \mathbf{v}_0 + \mathbf{A} \cdot \mathbf{v}_1 = \mathbf{b}$$

  Now, run and output $(\mathsf{fk}, \mathsf{ek}) = (\mathsf{sk}, \mathsf{ek}) \xleftarrow{R} \mathsf{SSE.Gen}(\lambda, \mathbf{A})$.

- $\mathsf{WPRF.F}(\mathsf{fk}, \mathbf{x})$: Let $\mathbf{t} = \mathbf{b} - \mathbf{B} \cdot (\mathbf{x}, 1)$. Run $\hat{\mathbf{t}} \leftarrow \mathsf{SSE.Encode}(\mathsf{fk}, \mathbf{t})$

- $\mathsf{WPRF.Eval}(\mathsf{ek}, \mathbf{x}, \mathbf{w})$: Let $\mathbf{v}_1 = (\mathbf{x}, \mathbf{w}, \boldsymbol{\pi})$ where $\boldsymbol{\pi}$ is the proof constructed above. Run and output $\hat{\mathbf{t}} \leftarrow \mathsf{SSE.Eval}(\mathsf{ek}, \mathbf{v}_1)$

Note that using $\mathsf{Convert}$ from above, $B \equiv \|\mathbf{A}\|_\infty = 2$.

Remark 6.2. $\mathbf{t} = \mathbf{b} - \mathbf{B}_R \cdot (\mathbf{x}, 1) \in \mathsf{IntRange}(\mathbf{A})$. This is because, on input $\mathbf{x}$ and any witness $\mathbf{w}$, either $R(\mathbf{x}, \mathbf{w}) = 1$, in which case we the proof $\boldsymbol{\pi}$ derived from this instance and witness give a subset sum. If $R(\mathbf{x}, \mathbf{w}) = 0$, we can produce a proof $\boldsymbol{\pi}' \in [0,1]^{g+1}$ by setting all of the gates correctly, except the last "cheating" pass gate, which we set to 1 by setting $\pi = 1/2$ for that gate.

Remark 6.3. For each instance $\mathbf{x}$, the corresponding target $\mathbf{t} = \mathbf{b} - \mathbf{B}_R \cdot (\mathbf{x}, 1)$ is unique. This is due to our construction, where the only valid proof π for input $\mathbf{x}$ must contain $\mathbf{x}$ (since it is part of the output of R'). Since the proof, and hence the subset, must be different for every $\mathbf{x}$, we have that the target must also be different for every $\mathbf{x}$. This is crucial for security, since mapping two instances to the same target would mean F is not a pseudorandom function.

6.3 Security of Our Construction

We now state the security of our construction. For an NP relation R, let $\mathbf{A}$ be the matrix defined above.

Theorem 6.4. *If* SSE *is an adaptive target interactively secure subset-sum encoding, then* WPRF *in Construction 6.1 is adaptive instance interactively secure.*

This theorem is an immediate consequence of our construction, and the fact that each instance maps to a unique target. We can also consider the fine-grained security notions. For an R-instance sampler $\mathcal{D}_{\mathsf{WPRF}}$ for WPRF, let $\mathcal{D}_{\mathsf{SSE}}$ be the following $\mathbf{A}$-target sampler for SSE: On input ek, simulate $\mathcal{D}_{\mathsf{WPRF}}$ on input ek. Whenever $\mathcal{D}_{\mathsf{WPRF}}$ makes a query to F on input $\mathbf{x}$, compute $\mathbf{t} = \mathbf{b} - \mathbf{B} \cdot (\mathbf{x}, 1)$, and make an encode query on $\mathbf{t}$, obtaining $\hat{\mathbf{t}}$. Return $\hat{\mathbf{t}}$ to $\mathcal{D}_{\mathsf{WPRF}}$. When $\mathcal{D}_{\mathsf{WPRF}}$ produces a witness $\mathbf{x}^*$, compute and output the target $\mathbf{t}^* = \mathbf{b} - \mathbf{B} \cdot (\mathbf{x}, 1)$. Security immediately follows:

Theorem 6.5. *If* SSE *is interactively (resp. non-interactively) secure for* $\mathbf{A}$ *and* $\mathcal{D}_{\mathsf{SSE}}$*, then* WPRF *is interactively (resp. non-interactively) secure for* R *and* $\mathcal{D}_{\mathsf{WPRF}}$*. Moreover, if* SSE *is extractable, then so is* WPRF*.*

6.4 Reducing Key Sizes Using Verifiable Computation

Our witness PRF requires a subset-sum encoding a matrix $\mathbf{A}$ of width proportional to the size of the circuit evaluating R. Our subset-sum encodings are in turn constructed from multilinear maps, and the total multilinearity will be equal to the width of the matrix $\mathbf{A}$. Since multilinearity is very expensive, it is important to reduce the size of the circuit $\mathbf{A}$.

The role of $\mathbf{A}$ is basically to prove that R has the output claimed, and operates by checking every step of the computation. In order to shrink $\mathbf{A}$, we could have Eval first compute $R(x, w)$, and simultaneously compute a proof π that the computation was correct. Then $\mathbf{A}$ does not check the evaluation of $R(x, w)$ directly, but instead checks the evaluation of the program that verifies π. If π and the verification algorithm can be made much smaller than R itself, this will decrease the size of $\mathbf{A}$.

One candidate approach would be to use PCPs. However, this will not work directly, as the randomness used by the verification algorithm would need to be hard-coded into the matrix $\mathbf{A}$, but the proof is generated in Eval, which gets (the ecoding of) $\mathbf{A}$ as input. Therefore, the prover could craft the proof to fool the verifier for the specific random coins used.

Instead, we can use verifiable computation, defined as follows:

Given a circuit R, a verifiable computation scheme consists of:

- $\mathsf{Gen}(\lambda, R)$ which outputs a verification key vk and evaluation key ek.
- $\mathsf{Compute}(\mathsf{ek}, x)$ computes $y = R(x)$ as well as a proof π.
- $\mathsf{Ver}(\mathsf{vk}, x, y, \pi)$ is a deterministic algorithm that outputs 0 or 1.

Given a verifiable computation scheme, we can build the following witness PRF

Construction 6.6. Let $(\mathsf{WPRF.Gen}, \mathsf{F}, \mathsf{Eval})$ be a witness PRF and $(\mathsf{VC.Gen}, \mathsf{Compute}, \mathsf{Ver})$ be a verifiable computation scheme. Build the following:

- $\mathsf{WPRF.Gen}'(\lambda, R')$: run $(\mathsf{vk}, \mathsf{ek}_0) \xleftarrow{R} \mathsf{VC.Gen}(\lambda, R')$. Then define the relation

$$R((\mathsf{vk}, \mathbf{x}), (\mathbf{w}, \pi)) = \mathsf{Ver}(\mathsf{vk}, (\mathbf{x}, \mathbf{w}), 1, \pi)$$

  Run $(\mathsf{fk}, \mathsf{ek}_1) \xleftarrow{R} \mathsf{WPRF.Gen}(\lambda, R)$. Output the secret key fk and public parameters $\mathsf{ek} = (\mathsf{vk}, \mathsf{ek}_0, \mathsf{ek}_1)$.
- $\mathsf{F}'(\mathsf{fk}, \mathbf{x}) = \mathsf{F}(\mathsf{fk}, (\mathsf{vk}, \mathbf{x}))$
- $\mathsf{Eval}'(\mathsf{ek}, \mathbf{x}, \mathbf{w})$: run $\mathsf{Compute}(\mathsf{ek}_0, (\mathbf{x}, \mathbf{w}))$ to obtain $b = R'(x, w)$ and a proof π. If $b = 0$, abort and output $\perp$. Otherwise, run $\mathsf{Eval}(\mathsf{ek}_1, (\mathsf{vk}, \mathbf{x}), (\mathbf{w}, \pi))$.

Correctness is immediate. Now the parameter size ek is the length of the evaluation key ek_0 plus the length of the parameter ek_1 for WPRF. However, the size of R is independent of the size of R', and only depends on the running time of Ver. Using the verifiable computation system of, say, Parno, Howell, Gentry and Raykova [PHGR13], this is linear in $|x|$ and $|w|$. The size of ek_0 however does depend linearly on the size of R. Thus, the total parameter size is depends only linearly on the size of R', rather than polynomially.

Security. Unfortunately, we need to rely on the strong form of extractable security. This is because false proofs do exist, and thus the languages defined by R and R' will not coincide. Since extracting security does not seem to imply non-extracting security, it is unlikely that we can prove F' non-extracting secure.

Given an R'-instance sampler $\mathcal{D}'$ for WPRF', we construct the following R-instance sampler $\mathcal{D}$ for WPRF. On input ek_1, run run $(\mathsf{ek}_0, \mathsf{vk}) \xleftarrow{R} \mathsf{VC.Gen}(\lambda, R')$ and give $\mathsf{ek} = (\mathsf{ek}_0, \mathsf{ek}_1, \mathsf{vk})$ to $\mathcal{D}'$. When $\mathcal{D}'$ makes a F' query on instance $\mathbf{x}$, make a F query on $(\mathsf{vk}, \mathbf{x})$. When $\mathcal{D}'$ outputs an instance $\mathbf{x}^*$, output $(\mathsf{vk}, \mathbf{x}^*)$, along with auxiliary information, namely Aux outputted by $\mathcal{D}'$ and ek_0.

Theorem 6.7. *If* WPRF *is extractable interactively (resp. non-interactively) secure for relation* R *and instance sampler* $\mathcal{D}$*, then* WPRF' *is extractable interactively (resp. non-interactively) secure for instance sampler* $\mathcal{D}'$*.*

Proof. We prove the interactive case, the other case begin similar. Let $\mathcal{A}'$ be an extractable adversary for WPRF' relative to instance sampler $\mathcal{D}'$. We construct the following adversary $\mathcal{A}$ for WPRF relative to $\mathcal{D}$. $\mathcal{A}$, on input $\mathsf{ek}_1, (\mathsf{vk}, \mathbf{x}^*), k, \mathsf{Aux}, \mathsf{ek}_0$, runs $\mathcal{A}'$ on $\mathsf{ek} = (\mathsf{ek}_0, \mathsf{ek}_1, \mathsf{vk}), \mathbf{x}^*, k, \mathsf{Aux}$. When $\mathcal{A}'$ makes a F' query on instance $\mathbf{x}$, answer by making a F query on $(\mathsf{vk}, \mathbf{x})$. When $\mathcal{A}'$ outputs a guess b', $\mathcal{A}$ outputs the same guess.

Suppose $\mathcal{A}'$ has non-negligible advantage $\epsilon = 1/q_{\mathcal{A}'}$. Observe that $\mathcal{A}$ perfectly simulates the view of $\mathcal{A}'$, so $\mathcal{A}$ also has advantage $1/q_{\mathcal{A}'}$. This implies an extractor $\mathcal{E}$ and polynomial $q_{\mathcal{E}}$ such that $\mathcal{E}$ has advantage at least $1/q_{\mathcal{E}}$. We construct the following extractor $\mathcal{E}'$ for WPRF'. On input $(\mathsf{ek}, x^*, \mathsf{Aux}, y^*, \{x_i, y_i\}, r)$, $\mathcal{E}'$ runs $\mathcal{E}$ on input $(\mathsf{ek}_1, (\mathsf{vk}, x^*), (\mathsf{Aux}, \mathsf{ek}_0), y^*, \{(\mathsf{vk}, x_i), y_i\}, r)$. $\mathcal{E}'$ perfectly simulates the view of $\mathcal{E}$, so the output (w, π) will satisfy $R'((\mathsf{vk}, \mathbf{x}^*), (w, \pi)) = 1$ with probability at least $1/q_{\mathcal{E}}$. Output w as a witness for $\mathbf{x}^*$.

Suppose $\mathcal{E}'$ has negligible advantage. This implies that $R(\mathbf{x}^*, w) = 1$ with negligible probability. But then π is an invalid proof, meaning $\mathcal{E}'$ can be used to break the security of VC, a contradiction. Therefore, $\mathcal{E}'$ has non-negligible advantage, say $1/2q_{\mathcal{E}}$.

□

References

[ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. http://eprint.iacr.org/.

[AGIS14] Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding barrington's theorem. Cryptology ePrint Archive, Report 2014/222, 2014. http://eprint.iacr.org/.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *Theoretical Cryptography Conference (TCC)*, 2014.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGI13] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. Cryptology ePrint Archive, Report 2013/401, 2013.

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *Proc. of EuroCrypt*, 2014.

[BH13] Mihir Bellare and Viet Tung Hoang. Adaptive witness encryption and asymmetric password-based cryptography. Cryptology ePrint Archive, Report 2013/704, 2013. http://eprint.iacr.org/.

[Bla79] George R. Blakley. Safeguarding cryptographic keys. In *Proc. of the AFIPS National Computer Conference*, 1979.

[BR13] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. Cryptology ePrint Archive, Report 2013/563, 2013. http://eprint.iacr.org/.

[BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90, 2003.

[BST13] Mihir Bellare, Igors Stepanovs, and Stefano Tessaro. Poly-many hardcore bits for any one-way function. Cryptology ePrint Archive, Report 2013/873, 2013. http://eprint.iacr.org/.

[BW13] Dan Boneh and Brent Waters. Constrained Pseudorandom Functions and Their Applications. *Advances in Cryptology (AsiaCrypt)*, pages 1–23, 2013.

[BWZ14] Dan Boneh, Brent Waters, and Mark Zhandry. Low overhead broadcast encryption from multilinear maps. Cryptology ePrint Archive, Report 2014/195, 2014. http://eprint.iacr.org/.

[BZ13] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2013/642, 2013. http://eprint.iacr.org/.

[CGHG01] Dario Catalano, Rosario Gennaro, and Nick Howgrave-Graham. The bit security of paillierâĂŹs encryption scheme and its applications. In Birgit Pfitzmann, editor, *Advances in Cryptology âĂŤ EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 229–243. Springer Berlin Heidelberg, 2001.

[CLT13] Jéan-Sebastien Coron, Tancède Lepoint, and Mehdi Tibouchi. Practical Multilinear Maps over the Integers. *Advances in Cryptology — CRYPTO 2013*, pages 1–22, 2013.

[CRV10] Ran Canetti, Guy N. Rothblum, and Mayank Varia. Obfuscation of hyperplane membership. In *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, volume 5978 of *Lecture Notes in Computer Science*, pages 72–89. Springer, 2010.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. *Advances in Cryptology (EuroCrypt)*, pages 45–64, 2002.

[GGH13a] S Garg, Craig Gentry, and S Halevi. Candidate multilinear maps from ideal lattices. *Advances in Cryptology (EuroCrypt)*, 2013.

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

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure mpc from indistinguishability obfuscation. In *Theoretical Cryptography Conference (TCC)*, 2014.

[GGHW13] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. Cryptology ePrint Archive, Report 2013/860, 2013. http://eprint.iacr.org/.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *Proc. of STOC*, 2013.

[GKP+13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, , and Nickolai Zeldovich. How to run turing machines on encrypted data. In *Proc. of Crypto*, 2013.

[GL89] O. Goldreich and L. A. Levin. A hard-cire predicate for all one-way functions. In *Proc. of STOC*, pages 25–32, 1989.

[HSS93] J. HÃěstad, A. W. Schrift, and A. Shamir. The discrete logarithm modulo a composite hides o(n) bits. *Journal of Computer and System Sciences*, 47:376–404, 1993.

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In *Proc. of EuroCrypt*, 2014.

[Jou04] Antoine Joux. A One Round Protocol for Tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, June 2004.

[KNY14] Ilan Komargodski, Moni Naor, and Eylon Yogev. Secret-sharing for np from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2014/213, 2014. http://eprint.iacr.org/.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *Proceedings ACM CCS*, 2013.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 238–252, Washington, DC, USA, 2013. IEEE Computer Society.

[PPS13] Omkant Pandey, Manoj Prabhakaran, and Amit Sahai. Obfuscation-based non-black-box simulation and four message concurrent zero knowledge for np. Cryptology ePrint Archive, Report 2013/754, 2013. http://eprint.iacr.org/.

[PST13] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. Cryptology ePrint Archive, Report 2013/781, 2013. http://eprint.iacr.org/.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[SW14] Amit Sahai and Brent Waters. How to Use Indistinguishability Obfuscation: Deniable Encryption, and More. In *Proc. of STOC*, 2014.

A Generic Hardness

In this section, we prove the generic hardness of our multilinear subset-sum Diffie-Hellman problem. We prove the hardness of the strongest variant, the extracting adaptive witness interactive assumption.

Generic Multilienar Group Model. We prove security in the so-called generic multilinear group model. In this model, rather than having direct access to group elements, the adversary only has access to âĂćphlabels of group elements. It also has access to an oracle that allows it to multiply elements of the same group, as well as apply the multilinear operation. We allow the adversary to successively pair elements together, rather than only providing the full multilinear map. This reflects the structure of current map candidates.

More precisely, we have a family of groups $\mathbb{G}_{\mathbf{u}}$ where $\mathbf{u} \in \{0,1\}^n$. The target group is $\mathbb{G}_T = \mathbb{G}_{1^n}$, and $\mathbb{G}_i = \mathbb{G}_{\mathbf{e}_i}$, where $\mathbf{e}_i$ is the ith unit vector. We represent the groups using a function $\xi : \mathbb{Z}_p \times \{0,1\}^n \to \{0,1\}^m$, which maps elements of the ring $\mathbb{Z}_p$ (along with a group index $\mathbf{u} \in \{0,1\}^n$) into bit strings of length m. We provide the adversary with oracles Mult and Pair to compute the induced group and pairing operations:

- $\mathsf{Encode}(x, \mathbf{u})$ returns $\xi(x, \mathbf{u})$. Note that to compute a generator for a group $\mathbb{G}_{\mathbf{u}}$ as $\mathsf{Encode}(1, \mathbf{u})$
- $\mathsf{Mult}(\xi_1, \xi_2, b)$ If $\xi_1 = \xi(x_1, \mathbf{u})$ and $\xi_2 = \xi(x_2, \mathbf{u})$ for the same index $\mathbf{v}$, then return $\xi(x_1 + (-1)^b x_2, \mathbf{u})$. Otherwise output $\perp$.
- $\mathsf{Pair}(\xi_1, \xi_2)$ if $\xi_1 = \xi(x_1, \mathbf{u}_1)$ and $\xi_2 = \xi(x_2, \mathbf{u}_2)$ where $\mathbf{u}_1 + \mathbf{u}_2 \leq 1^n$ (that is, the component-wise sum over the integers has all entries less than or equal to 1), then output $\xi(x_1 x_2, \mathbf{u}_1 + \mathbf{u}_2)$. Otherwise output $\perp$.

The following theorem shows the hardness of the general multilinear subset-sum Diffie-Hellman problem:

Theorem A.1. *Let $\mathbf{A} \in \mathbb{Z}^{m \times n}$ be a matrix with entries bounded by B. Assume B/p is negligible, where p is the group order. Then for any adaptive instance interactive adversary $\mathcal{A}$ for the multilinear subset-sum Diffie-Hellman problem, $\mathcal{A}$ has negligible advantage.*

Now we prove Theorem A.1:

Proof. Fix a matrix $\mathbf{A} \in \mathbb{Z}^{m \times n}$ with entries bounded by B, and adversary $\mathcal{A}$. We will set m arbitrarily high so that with overwhelming probability, ξ is injective and moreover the adversary cannot guess any representations, but must instead make queries to $\mathsf{Encode}, \mathsf{Mult}, \mathsf{Pair}$

Consider the execution of the experiment on $\mathcal{A}$. A random vector $\boldsymbol{\alpha} \xleftarrow{R} (\mathbb{Z}_p^*)^m$ is chosen at random, and the labels $V_i = \xi(\boldsymbol{\alpha}^{\mathbf{v}_i}, \mathbf{e}_i)$ for $i \in [n]$ are given to $\mathcal{A}$. $\mathcal{A}$ is allowed to make queries on vectors $\mathbf{t} \in \mathsf{IntRange}(\mathbf{A})$, to which we respond with $E_{\mathbf{t}} = \xi(\boldsymbol{\alpha}^{\mathbf{t}}, 1^n)$. $\mathcal{A}$ can also make a polynomial number of queries to $\mathsf{Encode}, \mathsf{Mult}, \mathsf{Pair}$ to perform group pairing operations. At some point, $\mathcal{A}$ produces a target $\mathbf{t}^*$ that was not among the queries made so far. Set $y_0 = E_{\mathbf{t}^*} = \xi(\boldsymbol{\alpha}^{\mathbf{t}^*}, 1^n)$. We also choose a random β and set $y_1 = \xi(\beta, 1^n)$. $\mathcal{A}$ is given y_b in response. $\mathcal{A}$ is allowed to make additional queries on $\mathbf{t} \neq \mathbf{t}^*$ to get values $E_{\mathbf{t}}$. Finally, $\mathcal{A}$ produces a guess b' for b. $\mathcal{A}$ has advantage $|\Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0]| = \epsilon$. Our goal is to show that ϵ is negligible.

Now let $\mathcal{B}$ be an (inefficient) algorithm that plays the above game with $\mathcal{A}$. Rather than choose $\boldsymbol{\alpha}, y_0, y_1$, $\mathcal{B}$ treats them as formal variables. $\mathcal{B}$ maintains a list $L = \{(p_j, \mathbf{u}_j, \xi_j)\}$ where p_j is a polynomial in $\boldsymbol{\alpha}, y_0, y_1$, $\mathbf{u}_j \in \{0,1\}^m$ indexes the groups, and ξ_j is a string in $\{0,1\}^m$. Note that the exponent of α_i in any polynomial may be negative. The list is initialized with the tuples $(\boldsymbol{\alpha}^{\mathbf{v}_i}, \mathbf{e}_i, \xi_{i,1})$ and $(1, \mathbf{e}_i, \xi_{i,0})$ for $i \in [n]$, where $\xi_{i,b}$ are randomly generated strings in $\{0,1\}^m$.

The game starts with $\mathcal{B}$ giving $\mathcal{A}$ the tuple of strings $\{\xi_{i,b}\}_{i\in[n],b\in\{0,1\}}$. Now $\mathcal{A}$ is allowed to make the following queries:

$\mathsf{Encode}(r, \mathbf{u})$: If $r \in \mathbb{Z}_p$ and $\mathbf{u} \in \{0,1\}^n$, then $\mathcal{B}$ looks for a tuple $(p, \mathbf{u}, \xi) \in L$, where p is the constant polynomial equal to r. If such a tuple exists, then $\mathcal{B}$ responds with ξ. Otherwise, $\mathcal{B}$ generates a random string $\xi \in \{0,1\}^m$, adds the tuple $(\pi, \mathbf{u}, \xi)$ to L, and responds with ξ.

$\mathsf{Mult}(\xi_k, \xi_\ell, b)$: $\mathcal{B}$ looks for tuples $(p_k, \mathbf{u}_k, \xi_k), (p_\ell, \mathbf{u}_\ell, \xi_e ll) \in L$. If one of the tuples is not found, $\mathcal{B}$ responds with $\perp$. If both are found, but $\mathbf{u}_k \neq \mathbf{u}_\ell$, then $\mathcal{B}$ responds with $\perp$. Otherwise, $\mathcal{B}$ lets $\mathbf{u} \equiv \mathbf{u}_k = \mathbf{u}_\ell$, and computes the polynomial $p = p_k + (-1)^b p_\ell$. Then $\mathcal{B}$ looks for the tuples $(p, \mathbf{u}, \xi) \in L$, and if the tuple is found $\mathcal{B}$ responds with ξ. Otherwise, $\mathcal{B}$ generates a random string ξ, adds $(p, \mathbf{u}, \xi)$ to L, and responds with ξ.

$\mathsf{Pair}(\xi_k, \xi_\ell)$: $\mathcal{B}$ looks for tuples $(p_k, \mathbf{u}_k, \xi_k), (p_\ell, \mathbf{u}_\ell, \xi_\ell) \in L$. If one of the tuples is not found, $\mathcal{B}$ responds with $\perp$. If both are found, but $\mathbf{u}_k \wedge \mathbf{u}_\ell \neq 0^n$ (in other words, $\mathbf{u}_k$ and $\mathbf{u}_\ell$ have a 1 in the same locaiton), then $\mathcal{B}$ also responds with $\perp$. Otherwise, let $\mathbf{u} = \mathbf{u}_k + \mathbf{u}_\ell$ (addition over $\mathbb{Z}$), and let $p = p_k \cdot p_\ell$. $\mathcal{B}$ looks for $(p, \mathbf{u}, \xi) \in L$, and if found, responds with ξ. Otherwise, $\mathcal{B}$ generates a random $\xi \in \{0,1\}^m$, adds $(p, \mathbf{u}, \xi)$ to L, and responds with ξ.

$O(\mathbf{t})$: $\mathcal{B}$ looks for a tuple $(\boldsymbol{\alpha}^{\mathbf{t}}, 1^n, \xi) \in L$, and responds with ξ if the tuple is found. Otherwise, $\mathcal{B}$ generates a random $\xi \in \{0,1\}^m$, adds $(\boldsymbol{\alpha}^{\mathbf{t}}, 1^n, \xi)$ to L, and responds with ξ. Let Q be the set of queries to O.

Challenge$(\mathbf{t}^*)$: $\mathcal{B}$ (inefficiently) tests if $\mathbf{t}^* \in \mathsf{SubSums}(\mathbf{A})$. If so, $\mathcal{B}$ aborts the simulation. Otherwise, $\mathcal{B}$ creates a new formal variable y, and adds the following tuple to L: $(y, 1^n, \xi)$. Then $\mathcal{B}$ gives to $\mathcal{A}$ the value ξ.

$\mathcal{A}$ ultimately produces a guess b'. Now, $\mathcal{B}$ chooses a random $b \in \{0,1\}$, as well as values for $\boldsymbol{\alpha} \in (\mathbb{Z}_p^*)^m$. If $b = 0$, $\mathcal{B}$ sets $y = \alpha^{\mathbf{t}^*}$, and otherwise, $\mathcal{B}$ chooses a random $y \in \mathbb{Z}_p$.

The simulation provided by $\mathcal{B}$ is perfect unless the choice for the variables $\boldsymbol{\alpha}, y$ results in an equality in the values of two polynomials p_k, p_ℓ in L that is not an equality for polynomials. More precisely, the simulation is perfect unless for some k, ℓ the following hold:

- $\mathbf{u}_k = \mathbf{u}_\ell$
- $p_k(\boldsymbol{\alpha}, y_0, y_1) = p_\ell(\boldsymbol{\alpha}, y_0, y_1)$, yet the polynomials p_k, p_ℓ are not equal.

Let Fail be the event that these conditions hold for some k, ℓ. Our goal is to bound the probability Fail occurs. First, in the $b = 0$ case, consider setting $y = \boldsymbol{\alpha}^{\mathbf{t}^*}$ as polynomials *before* assigning values to $\boldsymbol{\alpha}$. We claim that this does not create any new polynomial equalities. Suppose towards contradiction that this setting causes $p_k = p_\ell$ for some k, ℓ where equality did not hold previously. Then $p_k - p_\ell = 0$. Consider expanding $p_k - p_\ell$ out into monomials prior to the substitution. First, this expansion must contain a y term, and this term cannot have been multiplied by other variables (since polynomials involving y can only exist in the group $\mathbb{G}_{1^n}$). The expansion may also contain $\boldsymbol{\alpha}^{\mathbf{t}}$ terms for all $\mathbf{t} \in Q$, also not multiplied by any other variable (since $\mathbf{t} \in Q$ were only provided in the group $\mathbb{G}_{1^n}$). All other terms came from multiplying and pairing the V_i and g_i together. In particular, each remaining term must come from pairing a subset of the V_i together with the complementing subset of the g_i. Therefore, we can write $p_k - p_\ell$ as

$$p_k - p_\ell = C^* y + \sum_{\mathbf{t} \in Q} C_{\mathbf{t}} \boldsymbol{\alpha}^{\mathbf{t}} + \sum_{\mathbf{t} \in \mathsf{SubSums}(\mathbf{A})} D_{\mathbf{t}} \boldsymbol{\alpha}^{\mathbf{t}}$$

Recall that $\alpha^(p-1) = 1 \mod p$ for all $\alpha \in \mathbb{Z}_p \setminus \{0\}$. This means we should only consider monomials with exponents reduced mod $p-1$ to the range $(-(p-1)/2, (p-1)/2]$. Since all $\mathbf{t} \in Q$ are required to satisfy $\mathbf{t} \in \mathsf{IntRange}(\mathbf{A})$, meaning $\|\mathbf{t}\|_\infty \leq n\|\mathbf{A}\|_\infty$, and $p > 2n\|\mathbf{A}\|_\infty$, all of the exponents in $\boldsymbol{\alpha}^{\mathbf{t}}$ are already reduced. The same applies to $\boldsymbol{\alpha}^{\mathbf{t}^*}$. Since $\mathbf{t}^* \notin Q$ and $\mathbf{t}^* \notin \mathsf{SubSums}(\mathbf{A})$, the monomial $\boldsymbol{\alpha}^{\mathbf{t}^*}$ did not exist prior to substitution. Therefore substituting y with $\boldsymbol{\alpha}^{\mathbf{t}^*}$ cannot make the polynomial zero.

Now, notice that all of the V_i are monomials where each α_i has exponent at most B and at least $-B$. This means the exponent of α_i lies in the range $[-nB, nB]$ for any monomial in the expansion of $p_k - p_\ell$. Consider $p = \boldsymbol{\alpha}^{nB}(p_k - p_\ell)$ (where the exponentiation applies to each of the components of $\boldsymbol{\alpha}$). p is then a proper polynomial where all coefficients are non-negative, and the total degree is at most $2mnB$. Moreover, $p = 0$ as a polynomial if and only if $p_k = p_\ell$ as polynomials. Lastly, the only zeros of p are when $\alpha_i = 0$ for some i, or $p_k - p_\ell = 0$. Since $\boldsymbol{\alpha}$ is chosen to have only non-zero components, this leaves only the zeros of $p_k - p_\ell$. The Swartz-Zippel lemma then shows that if $p \neq 0$, the probability that the polynomial evaluates to zero is at most $2mnB/(p-1)$. This means that p_k and p_ℓ evaluate to the same value with probability at most $2mnB/(p-1)$.

Let q_e, q_m, q_p, q_o be the number of encode, multiply, pair, and O queries made by $\mathcal{A}$. Then the total length of L is at most $q_e + q_m + q_p + q_o + n + 1$. Therefore, the number of pairs is at most $(q_e + q_m + q_p + q_o + n + 1)^2/2$, and so Fail happens with probability at most $mnB(q_e + q_m + q_p + q_o + n + 1)^2/2(p-1)$. If Fail does not occur, $\mathcal{B}$'s simulation is perfect, and in this case b is independent from $\mathcal{A}$'s view since b was chosen after the simulation. It is straightforward to show that $\mathcal{A}$'s advantage is then at most $mnB(q_e + q_m + q_p + q_o + n + 2)^2/2(p-1)$. For any polynomial number of queries, this is negligible provided B/p is negligible, as desired.

$\square$