Key-Homomorphic Constrained Pseudorandom Functions

Abhishek Banerjee[1]*, Georg Fuchsbauer[2]**, Chris Peikert[1] ***, Krzysztof Pietrzak[2] †, and Sophie Stevens[3] ‡

[1] School of Computer Science, College of Computing, Georgia Institute of Technology
[2] Institute of Science and Technology Austria
[3] University of Bristol, UK

Abstract. A pseudorandom function (PRF) is a keyed function $F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ where, for a random key $k \in \mathcal{K}$, the function $F(k, \cdot)$ is indistinguishable from a uniformly random function, given black-box access. A *key-homomorphic* PRF has the additional feature that for any keys k, k' and any input x, we have $F(k + k', x) = F(k, x) \oplus F(k', x)$ for some group operations $+, \oplus$ on $\mathcal{K}$ and $\mathcal{Y}$, respectively. A *constrained* PRF for a family of sets $\mathcal{S} \subseteq \mathcal{P}(\mathcal{X})$ has the property that, given any key k and set $S \in \mathcal{S}$, one can efficiently compute a "constrained" key k_S that enables evaluation of $F(k, x)$ on all inputs $x \in S$, while the values $F(k, x)$ for $x \notin S$ remain pseudorandom even given k_S.

In this paper we construct PRFs that are simultaneously constrained *and* key homomorphic, where the homomorphic property holds even for constrained keys. We first show that the multilinear map-based bit-fixing and circuit-constrained PRFs of Boneh and Waters (Asiacrypt 2013) can be modified to also be *key-homomorphic*. We then show that the LWE-based key-homomorphic PRFs of Banerjee and Peikert (Crypto 2014) are essentially already *prefix-constrained* PRFs, using a (non-obvious) definition of constrained keys and associated group operation. Moreover, the constrained keys themselves are pseudorandom, and the constraining and evaluation functions can all be computed in low depth.

As an application of key-homomorphic constrained PRFs, we construct a proxy re-encryption scheme with fine-grained access control. This scheme allows storing encrypted data on an untrusted server, where each file can be encrypted relative to some attributes, so that only parties whose constrained keys match the attributes can decrypt. Moreover, the server can re-key (arbitrary subsets of) the ciphertexts without learning anything about the plaintexts, thus permitting efficient and fine-grained revocation.

* Supported by the third author's grants.
** Research supported by ERC starting grant (259668-PSPC).
*** This material is based upon work supported by the National Science Foundation under CAREER Award CCF-1054495, by DARPA under agreement number FA8750-11-C-0096, and by the Alfred P. Sloan Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, DARPA or the U.S. Government, or the Sloan Foundation. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.
† Research supported by ERC starting grant (259668-PSPC).
‡ Work done while visiting the Institute of Science and Technology Austria.

1 Introduction

Pseudorandom functions (PRFs), like the AES block cipher, are the workhorses of cryptography. They allow for efficient and elegant solutions to all the basic symmetric-key cryptographic tasks, including authentication and encryption. Not surprisingly, PRFs with additional properties have been intensively investigated, as those properties often allow for useful additional functionalities. We discuss two such properties below.

Key-homomorphic PRFs. A PRF [GGM86] is an efficiently computable keyed function $F\colon \mathcal{K}\times\mathcal{X} \to \mathcal{Y}$. The security property requires that no efficient adversary can distinguish $F(k,\cdot)$ instantiated with a random key $k \leftarrow \mathcal{K}$ from a uniformly random function, given oracle access.

A *key-homomorphic* PRF has the additional feature that for any keys k, k' and any input x, we have $F(k + k', x) = F(k, x) \oplus F(k', x)$ for some group operations $+$ and $\oplus$ on $\mathcal{K}$ and $\mathcal{Y}$, respectively. Naor, Pinkas and Reingold [NPR99] observed that the simple PRF $F(k, x) = H(x)^k$, where $H(\cdot)$ is a random oracle that maps into a group where the DDH problem is assumed to be hard, is a key-homomorphic PRF. The first (almost) key-homomorphic PRFs in the standard model was constructed by Boneh *et al.* [BLMR13] from lattice assumptions, and later generalized and improved by Banerjee and Peikert [BP14].

Applications of key-homomorphic PRFs include an elegant solution to one-round *distributed* PRFs for any threshold [BLMR13]. Here, for some parameters $\ell \leq n$, a user sends an input x to ℓ servers, who each return a short answer from which the user can compute $F(k, x)$. Security requires that to any subset of $\ell - 1$ servers, $F(k,\cdot)$ is pseudorandom. For $\ell = n$, one can simply share the key as $k = k_1 + k_2 + \ldots + k_n$, each server computes $F(k_i, x)$, and these can then be combined to $\sum_{i=1}^{n} F(k_i, x) = F(\sum_{i=1}^{n} k_i, x) = F(k, x)$. Boneh *et al.* [BLMR13] provide a solution for general $\ell \leq n$. Symmetric-key proxy re-encryption is another interesting application, which we will discuss in detail in Section 1.2.

Constrained PRFs. A *constrained* PRF for a family of sets $\mathcal{S} \subseteq \mathcal{P}(\mathcal{X})$ has the property that, given any key k and set $S \in \mathcal{S}$, one can efficiently compute a "constrained" key k_S that enables evaluation of $F(k, x)$ on all inputs $x \in S$, while the values $F(k, x)$ for $x \notin S$ remain pseudorandom even given k_S.

Constrained PRFs were introduced independently in [BW13,KPTZ13,BGI14]. All three papers note that the classical GGM construction [GGM86] already gives a prefix-constrained PRF, where from a key $k \in \{0,1\}^n$, for any $v \in \{0,1\}^{\leq n}$ one can compute a key k_v that enables the computation of $F(k, x)$ for all inputs x that start with v. Boneh and Waters [BW13] construct bit-fixing and circuit-fixing constrained PRFs from multilinear maps. In the bit-fixing construction, for every $v \in \{0, 1, ?\}$ one can compute a key k_v that enables the computation of $F(k, x)$ for any x for which $x_i = v_i$ when $v_i \neq ?$. The more general circuit-constrained construction allows generating constrained keys for any circuit C, where with k_C one can evaluate the PRF on input x if and only if $C(x) = 1$.

Prefix-constrained PRFs (or rather, "punctured" PRFs, which can be constructed from them) are a main tool in almost all the applications of indistinguishability obfuscation [GGH$^+$13b,SW14,PST14]. The papers [BW13,BGI14,KPTZ13] discuss many more applications of constrained PRFs.

1.1 Results and Techniques

Key-Homomorphic Constrained PRFs. In this paper we construct PRFs that are simultaneously key-homomorphic and constrained. The key-homomorphic property holds not only for PRF keys, but also for constrained keys. We first show that the multilinear-map-based bit-fixing and circuit-constrained PRFs due to Boneh and Waters [BW13] can be modified to also be *key-homomorphic*. We then show that the LWE-based key-homomorphic PRFs of Banerjee and Peikert [BP14] are essentially already *prefix-constrained* PRFs, using a (non-obvious) definition of constrained keys and associated group operation. Moreover, the constrained keys themselves are pseudorandom, and the constraining and evaluation functions can all be computed in low depth. The latter feature can be important for applications of obfuscation, e.g., [GGH$^+$13b,SW14], where the use of low-depth constrained/punctured PRFs may avoid the need for costly "bootstrapping" operations and fully homomorphic encryption.

Given the usefulness of the individual key-homomorphic and constraining properties for PRFs, we expect their combination to find even more exciting applications. We discuss one such application, symmetric-key proxy re-encryption, in Section 1.2. We next give a brief overview of our constructions, their salient features, and our proof techniques.

Bit-Fixing PRFs from MDDH. Leveled multilinear maps [GGH13a] are defined over a sequence of groups $(\mathbb{G}_1, \ldots, \mathbb{G}_\kappa)$, where $\mathbb{G}_i$ is generated by an element g_i, as bilinear maps $e_{i,j} \colon \mathbb{G}_i \times \mathbb{G}_j \to \mathbb{G}_{i+j}$; i.e., they satisfy $e_{i,j}\big(g_i^a, g_j^b\big) = (g_{i+j})^{a \cdot b}$ for all a, b. The multilinear decisional Diffie-Hellman assumption states that given random elements $g_1^{c_1}, \ldots, g_1^{c_{\kappa+1}}$, it is hard to distinguish $g_\kappa{}^{\prod_{j=1}^{\kappa+1} c_j}$ from a random group element in $\mathbb{G}_\kappa$.

Using such groups, Boneh and Waters [BW13] define a bit-fixing constrained PRF for bit strings of length n as follows. A key K consists of a sequence of multilinear groups of prime order p and values $(k, \{d_{i,\beta}\}_{i \in [n],\, \beta \in \{0,1\}})$ from $\mathbb{Z}_p$. The PRF is defined as $F(K, x) := g_n{}^{k \cdot \prod_{i \in [n]} d_{i,x_i}}$. While this construction does not appear to be key-homomorphic, in Section 3 we make it so, by observing that we can "outsource" the values $d_{i,\beta}$ as $D_{i,\beta} := g_1^{d_{i,\beta}}$ to public parameters pp, and redefine

$$F(pp, k, x) := e\big(D_{1,x_1}, e(D_{2,x_2}, e(\ldots, e(D_{n-1,x_{n-1}}, D_{n,x_n})))\big)^k = g_n{}^{k \cdot \prod_{i \in [n]} d_{i,x_i}}.$$

We show that the values $D_{i,\beta}$ can be published without compromising security, that is, the function values are pseudorandom under the MDDH assumption. Because the secret key is now just k, the PRF is easily seen to be key-homomorphic.

Low-Depth Prefix-Fixing PRFs from LWE. In Section 4 we construct key-homomorphic prefix-fixing constrained PRFs from the LWE assumption, and hence from the conjectured hardness of worst-case lattice problems [Reg09,Pei09,BLP$^+$13]. In addition, natural instantiations of this construction have polylogarithmic circuit depth. To our knowledge, these are the first sublinear-depth constrained PRFs (whether key-homomorphic or not), and as such they can admit much more efficient obfuscation under existing paradigms. (Recall that the basic GGM construction, which yields a prefix-constrained PRF, is highly sequential.)

Our LWE-based construction is an extension of the recent class of key-homomorphic PRFs of Banerjee and Peikert [BP14], which generalizes and improves a previous construction of Boneh *et al.* [BLMR13]. We show that the BP construction can be made prefix-constrained, and that the constraining algorithm is also key-homomorphic. Notably, the approximation factors for the underlying LWE assumption are essentially the same as in [BP14], e.g., they can be as small as quasi-polynomial $\lambda^{\omega(\lambda)}$ in the security parameter.

To show all this, we start with the observation that the security proof for the BP (and BLMR) construction is very "GGM-like," i.e., it proceeds in a sequence of hybrids, one for each successive bit of the PRF input. However, the functions computed in the hybrids do not quite fit the usual GGM paradigm, because each successive output of the PRG is broken into two pieces: one piece is fed as input into the next PRG step, while the "leftover" piece is retained and then later "folded" back into the final output of all the PRG steps. A natural way to define a constrained key for a *partial* function evaluation, then, is to include all the leftover pieces in the constrained key—and this is indeed the approach we take.

The main technical challenge we face is in defining a suitable *group structure* on the leftover pieces, for key homomorphism. At first sight, this appears easy: since the leftover pieces are eventually combined with the final PRG output via a linear function, it would appear that one could simply add constrained keys by adding their corresponding leftover pieces. While this does indeed work—at least syntactically—it yields a useless construction! The problem is that essentially any application of key-homomorphic (constrained) PRFs (e.g., proxy re-encryption as described below in Section 1.2) will require a statistical "secret sharing"-like property on the (constrained) secret keys. For example, the sum of any fixed key with a uniformly random key must be uniformly random, so that the original key is completely hidden. Formally speaking, for any particular constraint we need the space of constrained keys to be a *finite* additive group (so that it supports a uniform distribution), and for the function to be key-homomorphic under this group structure.

Resolving the difficulty. Going back to the BP construction, the leftover pieces in constrained keys come from a certain finite subset $\mathcal{P} \subset \mathbb{Z}^m$, namely, a fundamental region of a special lattice $\mathcal{L}$. Obviously, the sum of two uniformly random $\mathcal{P}$-elements is *not* uniform in $\mathcal{P}$—indeed, it is typically not in $\mathcal{P}$ at all! So we cannot naïvely use the ambient group $\mathbb{Z}^m$ (which is infinite). Another idea would be to use the *finite* quotient group $\mathbb{Z}^m/\mathcal{L}$, i.e., addition modulo the lattice. This also does not work, because the function is not key-homomorphic under this form of addition.

Our solution to the above problem involves a novel method of adding modulo $\mathcal{L}$ "with carries." That is, the sum of two leftover $\mathcal{P}$-elements is mapped back to $\mathcal{P}$ by reducing modulo the lattice $\mathcal{L}$, i.e., shifting by an appropriate lattice vector $\mathbf{x} \in \mathcal{L}$. The vector $\mathbf{x}$ is then treated as a "carry" term that is "folded into" the sum of the next two $\mathcal{P}$-elements in the key, and so on. (The ultimate effect is analogous to grade-school addition, except that here the "base" in which the "numbers" are written is a high-dimensional lattice.) We show that by appropriately defining the effect of the carry terms, the PRF is indeed key-homomorphic under this form of addition.

All of the above applies to the so-called "noisy" version of the BP construction, an intermediate object that has perfect constraining, homomorphic, and pseudorandomness properties, but high circuit depth and (even worse) *exponentially* large keys. Similar to [BPR12,BP14], we show that by appropriately "rounding" this noisy construction, the keys and depth can be made small while preserving the other desirable properties (at least against computationally bounded attackers). Interestingly, this rounding transformation requires us to work with a "geometrically nice" set $\mathcal{P}$ of representatives modulo the special lattice $\mathcal{L}$ (which fortunately exists), whereas [BP14] works with any set of representatives.

1.2 Applications

Using symmetric encryption, one can store data on an untrusted server simply by first encrypting the files to be stored. Key-homomorphic and/or constrained PRFs enable symmetric encryption schemes with additional properties which are useful in this setting.

Assume there are many parties who should get access to the stored data, but that we occasionally need to revoke the access of some party. A simple solution is to re-encrypt all the data with a fresh key, and then give this key to only the parties who should continue to have access. Unfortunately, this requires either that the server knows the secret key k, or that we must download, re-encrypt, and upload the entire database. Boneh *et al.* [BLMR13] show how by using a *key-homomorphic* PRF, one can construct a so-called proxy re-encryption scheme, where the server can locally transform ciphertexts under a key k to ciphertexts under a new key k' without learning the plaintexts. We discuss their construction in Section 5.1.

The second functionality we consider is fine-grained access control, where different parties should get access to different subsets of the stored data. The trivial but inefficient solution is to encrypt each file under a separate key, and then send the appropriate keys to each party. *Constrained* PRFs (CPRF) provide a more elegant solution: every encrypted file is associated with some attribute vector x, and every party gets a constrained key k_p that allows her to evaluate the PRF on only those inputs satisfying an appropriate predicate p. The PRF then allows her to decrypt only those files whose attributes x satisfy her predicate. Of course, the expressive power of the system depends upon the predicates supported by the CPRF. A circuit CPRF allows for any efficiently computable predicate p, whereas prefix CPRFs allow for predicates that are satisfied by inputs starting with a particular prefix. Using *key-homomorphic constrained* PRFs as constructed in this paper, in Section 5 we construct a scheme for outsourced storage that supports proxy re-encryption and fine-grained access control simultaneously. The "obvious" way to outsource storage to an untrusted server using CPRFs is to encrypt a message m for some attributes x as $c = m \oplus F(k, x)$. Now, only a party who has a constrained key k_p where $p(x) = 1$ can decrypt the ciphertext (c, x), by computing $m = c \oplus F(k, x)$. This simple solution has two problems.

First, given two ciphertexts $(c, x), (c', x)$ for the same attributes x, one can compute the XOR of the messages as $c \oplus c' = m \oplus F(k, x) \oplus m' \oplus F(k, x) = m \oplus m'$, breaking the security of the encryption scheme.

Second, a single ciphertext $c = m \oplus F(k, x)$ for a known m reveals $F(k, x) = c \oplus m$. This is a problem because the security game for CPFRs only guarantees that $F(k, x)$ is pseudorandom if the adversary was given constrained keys (for predicates $p(.)$ where $p(x) = 0$), but does not guarantee anything if she is also given outputs $F(k, x')$ for some $x' \neq x$. For the GGM based prefix CPRF there is in fact a simple attack (cf. Footnote 4).

To handle these problems, we show how to "randomize" predicates, in the sense that p^+ is a randomization of p if there exists some encoding $[\cdot, \cdot]$ such that for all (x, r) we have $p^+([x, r]) = 1$ if and only if $p(x) = 1$. Let $\mathcal{P}$ denote the predicates supported by the CPRF considered. We require $p^+ \in \mathcal{P}$ as this will assure that the set of predicates for the encryption scheme is the same as for the CPRF. We will need some other properties from the encoding which we outline below. Although we don't give a generic result showing how to randomize any set of predicates, we show very simple constructions that work for prefix, bit-fixing and circuit CPRFs (that is, all the predicates for which CPRFs have been constructed to date). For bit-fixing and and circuit CPRFs the encoding is simply concatenation $[x, r] = x \| r$. For prefix CPRFs the encoding is a simple prefix-free encoding (cf. the paragraph above Thm. 4).

To solve the problems outlined above, we make encryption probabilistic: we encrypt m as $(r, m \oplus F(k, [x, r]))$ using randomness r. A constrained key for the predicate $p(\cdot)$ for the encryption scheme is now defined as a constrained key for the predicate $p^+(.)$ for the CPRF. Note that with this key we can compute $F(k, [x, r])$ and thus decrypt if $p(x) = 1$ for any r.

Arguing security is more delicate, and will require two extra properties. First, we want $[\cdot, \cdot]$ to be injective, which will ensure that the value $F(k, [x, r])$ used in the challenge ciphertext has never been output before with high probability (i.e., unless we happened to choose the same randomness r for a previous query). Second, we want that for every $[x, r]$, there exists a predicate $p_{[x,r]} \in \mathcal{P}$ such that $p_{[x,r]}([x, r]) = 1$ but $p_{[x,r]}([x', r']) = 0$ for all $(x', r') \neq (x, r)$ (but $p_{[x,r]}(z)$ can be 1 for values z outside the range of $[\cdot, \cdot]$). In the reduction, this latter property allows us to learn the values $F(k, [x, r])$ required to answer encryption queries in the CPA game by querying our oracle (playing the CPRF security game) for the constrained key with predicate $p_{[x,r]}$. The above property ensures that every such query will exclude at most one possible candidate for our challenge ciphertext. Thus, if at some point the adversary asks for a challenge ciphertext using attributes x, we can chose our CPRF challenge as $[x, r]$ (which will be answered either by $F(k, [x, r])$ or uniform), and as we chose r uniformly at random, this query will most likely not be invalid (in the sense that it could be computed using some previously issued constrained key).

Efficient re-encryption. Using proxy re-encryption as outlined above requires the server to re-encrypt the *entire* database to ensure that a revoked party loses access. When using fine-grained access control, a party to be revoked might have access to only a small fraction of the database, so we could re-encrypt only that portion. This would make re-encryption (potentially much) more efficient, but would require some extra key-management, as now different parts of the database are encrypted under different keys.

2 Preliminaries

2.1 Key-Homomorphic Constrained Pseudorandom Functions

We now formally define key-homomorphic constrained pseudorandom functions. We model constrainability as a directed acyclic graph (DAG) on some (typically huge) set of nodes. We restrict our attention to DAGs having a unique node that has no incoming edges, called the *root node*.

Definition 1. *A* constrained function family $\mathcal{C}$ *is given by:*

- *a directed acyclic graph* $D = (V, E)$ *with unique root node* $r \in V$*,*
- *for each node* $u \in V$*, a* key space $\mathcal{K}_u$ *with an efficiently samplable probability distribution* $\mathcal{D}_u$ *over it,*
- *for every edge* $(u, v) \in E$*, a* constraining function $C_{u,v} \colon \mathcal{K}_u \to \mathcal{K}_v$ *that is efficiently computable.*

The functions $C_{u,v}$ *must satisfy the following* consistency *property: for any* $u, v \in V$ *and any two paths* $P = (u = u_0, u_1, \ldots, u_k = v)$ *and* $P' = (u = u'_0, u'_1, \ldots, u'_\ell = v)$ *from* u *to* v*, we have that*

$$C_{u_{k-1},u_k} \circ \cdots \circ C_{u_1,u_2} \circ C_{u_0,u_1} = C_{u'_{\ell-1},u'_\ell} \circ \cdots \circ C_{u'_1,u'_2} \circ C_{u'_0,u'_1} \ .$$

For notational convenience, we let $C_{u,v} \colon \mathcal{K}_u \to \mathcal{K}_v$ *denote the above (composed) functions, and also define* $C_u := C_{r,u}$ *for any node* $u \in V$ *that is reachable from the root node* r*. For consistency with the typical PRF notation, we define* $F_k(u) = C_u(k)$ *(and to also cover constrained PRFs, if* u *represents a subset of inputs then* $\mathsf{Constrain}_k(u) = C_u(k)$*).*

Lastly, a constrained function family may also have a Setup *algorithm, which samples some (public) parameters that are provided as input to all of the other algorithms.*

For the reader who may be familiar with constrained PRFs, we stress that in the above definition, the DAG nodes roughly correspond with (subsets of) *PRF inputs*, while the input k_u and output k_v of constraining function $C_{u,v}$ correspond to (constrained) *secret keys*. Despite these rough correspondences, we stress that in our model there are no distinct notions of PRF "inputs" or "outputs," only DAG nodes. This is without loss of generality: a PRF input can simply be represented as a node w with no outgoing edges, and the corresponding output is the key k_w. In fact, our model is somewhat more general because it allows for defining and proving the pseudorandomness of *constrained keys* themselves (even for nodes having outgoing edges), which can be useful in certain settings.

Definition 2. *Pseudorandomness for a constrained function family* $\mathcal{C} = \big(D = (V, E), \{\mathcal{K}_u\}, \{C_{u,v}\}\big)$ *is defined as follows. It is parameterized by a subset* $R \subseteq V$ *of what we call "challenge" nodes. We consider two closely related experiments ("games"), called "real" and "ideal," which proceed as follows:*

1. Initialize: *For the root node $r \in V$ we choose a value $k = k_r \leftarrow \mathcal{K}_r$ according to the associated distribution $\mathcal{D}_r$. If the family has a* Setup *algorithm, it is run and its output is provided to the adversary.*
2. Query: *The adversary adaptively issues queries $v \in V$, subject to the condition that no query in R and any other query have a common descendant in D. That is, there are no distinct queries $u \in R$, $u' \in V$ and node $w \in V$ such that there exists a (possibly trivial) path from u to w and one from u' to w.*
   - *In the "real game," every query v is answered with $k_v = F_k(v) = C_v(k)$.*
   - *In the "ideal game," if $v \in V \setminus R$ then it is answered as in the real game, otherwise it is answered with an independent value $k_v^* \leftarrow \mathcal{D}_v$. (Repeated queries are answered consistently.)*

The family $\mathcal{C}$ is said to be pseudorandom *if for any polynomial-time adversary, its advantage in distinguishing the real and ideal games is negligible in the security parameter.*

In short, the definition above means that constrained keys *for the set R of challenge nodes* are pseudorandom. The condition on legal queries is necessary to prevent trivial distinguishers that work by observing the inconsistency of the ideal-game answers. In a bit more detail, given answers $k_u, k_{u'}$ for some nodes $u \in R$, $u' \in V$ (respectively) that have a common descendant $w \in V$, the distinguisher could check whether $C_{u,w}(k_u) = C_{u',w}(k_{u'})$. This always holds in the real game, but in the ideal game, where k_u is chosen independently of everything else, it would typically fail to hold.

Definition 3. *A constrained function family is* (key) homomorphic *if all the key spaces $\mathcal{K}_u$ are additive groups and if the constraining functions $C_{u,v}$ are additive homomorphisms, i.e., for every $(u, v) \in E$ and every $k_1, k_2 \in \mathcal{K}_u$, we have*

$$C_{u,v}(k_1) + C_{u,v}(k_2) = C_{u,v}(k_1 + k_2) \ .$$

For key-homomorphic PRFs, all applications we know of implicitly require the key spaces $\mathcal{K}_u$ to be *finite* groups, and the associated distributions $\mathcal{D}_u$ to be *uniform* distributions. In short, this is because the security proofs all rely on statistical "secret sharing"-type properties, e.g., the sum of any group element and a uniformly random one is uniformly random. All our final constructions have finite key spaces with uniform distributions.

3 Bit-Fixing and Circuit-Constrained Constructions from MDDH

Boneh and Waters [BW13] constructed a "bit-fixing" constrained PRF for input space $\mathcal{X} = \{0,1\}^n$, where one can derive constrained keys for any subset of inputs that can be described by arbitrarily fixing the values of any desired input bits. Any such subset can be described by a string $\mathbf{v} \in \{0, 1, ?\}^n$, as the set of all $x \in \{0,1\}^n$ that match $\mathbf{v}$ at all positions where $\mathbf{v}$ is different from '?':

$$S_{\mathbf{v}} := \left\{ x \in \{0,1\}^n \;\middle|\; \forall i \in [n] : x_i = \mathbf{v}_i \vee \mathbf{v}_i = ? \right\} \ . \tag{1}$$

Although not considered in [BW13], their construction can easily be generalized to allow computation of a constrained key for a set $S_{\mathbf{w}}$ not only from the root key, but also from any key for a set $S_{\mathbf{v}}$ for which $S_{\mathbf{w}} \subseteq S_{\mathbf{v}}$. In our DAG-based model, then, the nodes of the DAG consist of the strings $\mathbf{v} \in \{0, 1, ?\}^n$, and there is an edge $(\mathbf{v}, \mathbf{w})$ if and only if $S_{\mathbf{v}} \supseteq S_{\mathbf{w}}$ (equivalently, $\mathbf{w}_i = \mathbf{v}_i$ whenever $\mathbf{v}_i \neq ?$).

The original BW construction does not appear to be key homomorphic. However, we show how to make it so by defining public parameters for the function (which consist of elements previously contained in the secret key), and only keeping one $\mathbb{Z}_p$ element as the original secret key.

After these two modifications, we show that the PRF remains a bit-fixing constrained pseudorandom function family as defined in Definition 2. The set of challenge nodes is $R = \{0, 1\}^n$, corresponding to all "fully constrained" keys. That is, constrained keys for terminal nodes in the DAG are pseudorandom, but for nodes with outgoing edges they are not.

3.1 Preliminaries

Multilinear groups. Candidates for sequences of groups with leveled multilinear maps were first proposed by Garg, Gentry and Halevi [GGH13a]. These constructions implement *graded encodings*, which could be viewed as approximate multilinear groups. We present our results in the language of multilinear groups.

Leveled multilinear groups are generated by a group generator $\mathcal{G}$, which takes as input the security parameter 1^λ and $\kappa \in \mathbb{N}$, which determines the number of levels. $\mathcal{G}(1^\lambda, \kappa)$ outputs a sequence of groups $\mathbb{G} = (\mathbb{G}_1, \ldots, \mathbb{G}_\kappa)$ of prime order $p > 2^\lambda$. We assume that the description of each group contains a canonical generator g_i. For all $i, j \geq 1$ with $i + j \leq \kappa$, there exists a bilinear map $e_{i,j} \colon \mathbb{G}_i \times \mathbb{G}_j \to \mathbb{G}_{i+j}$, which satisfies:

$$\forall a, b \in \mathbb{Z}_p : e_{i,j}\big(g_i^a, g_j^b\big) = (g_{i+j})^{a \cdot b} \ .$$

We will omit the indices of e and write $e(h_1, h_2, \ldots, h_n)$ or $e(\{h_i\}_{i=1}^n)$ as a shorthand for $e(h_1, e(h_2, e(\ldots, e(h_{n-1}, h_n))))$. We make the following hardness assumption:

Assumption 1 *The* κ-Multilinear Decisional Diffie-Hellman *(κ-MDDH) assumption states that given $(\mathbb{G}_1, \ldots, \mathbb{G}_\kappa) \leftarrow \mathcal{G}(1^\lambda, \kappa)$ and $g = g_1, g^{c_1}, \ldots, g^{c_{\kappa+1}}$ for (uniformly) random $c_1, \ldots, c_{\kappa+1} \leftarrow \mathbb{Z}_p$, it is hard to distinguish $g_\kappa^{\prod_{j \in [\kappa+1]} c_j} \in \mathbb{G}_\kappa$ from a random group element in $\mathbb{G}_\kappa$.*

3.2 Key-Homomorphic Bit-Fixing PRF

$\mathsf{Setup}(1^\lambda, 1^n)$: On input the security parameter λ and the input length n, run $\mathcal{G}(1^\lambda, n)$ to compute a sequence of groups $\mathbb{G} = (\mathbb{G}_1, \ldots, \mathbb{G}_n)$ of prime order p, with generators $g := g_1, \ldots, g_n$. Choose $(d_{1,0}, d_{1,1}), \ldots, (d_{n,0}, d_{n,1}) \leftarrow \mathbb{Z}_p^2$ uniformly at random and set $D_{i,\beta} := g^{d_{i,\beta}}$ for $i \in [n]$ and $\beta \in \{0, 1\}$. Output the parameters of the scheme as

$$pp \ := \ \big(\mathbb{G} = (\mathbb{G}_1, \ldots, \mathbb{G}_n), \{D_{i,\beta}\}_{i \in [n],\, \beta \in \{0,1\}}\big) \ .$$

They define the domain as $\mathcal{X} = \{0,1\}^n$ and the range of the PRF as $\mathcal{Y} = \mathbb{G}_n$. For a key $k \in \mathbb{Z}_p$, the PRF value on input $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ is defined as

$$F(pp, k, x) := e\big(\{D_{i,x_i}\}_{i\in[n]}\big)^k \;=\; g_n{}^{k\cdot\prod_{i\in[n]} d_{i,x_i}} \quad .$$

$\mathsf{Constrain}(pp, k, \mathbf{w})$: On input pp, a key $k \in \mathbb{Z}_p \cup \bigcup_{i=1}^{n-1} \mathbb{G}_i$ and a vector $\mathbf{w} \in \{0,1,?\}^n$, which describes the constrained set as $S_\mathbf{w} := \{x \in \{0,1\}^n \mid \forall i \in [n] : x_i = \mathbf{w}_i \vee \mathbf{w}_i = ?\}$, let $W := \{i \in [n] : \mathbf{w}_i \neq ?\}$ be the set of indices that $\mathbf{w}$ fixes.

– If $k \in \mathbb{Z}_p$ (that is, k is a master key) then return

$$k_\mathbf{w} := e\big(\{D_{i,v_i}\}_{i\in W}\big)^k \;=\; (g_{|W|})^{k\cdot\prod_{i\in W} d_{i,\mathbf{w}_i}} \quad .$$

– Otherwise, we have $k = k_\mathbf{v}$ for some set $\mathbf{v} \in \{0,1,?\}^n$, for which we let V be the set of fixed indices. If $V \not\subseteq W$ or $\mathbf{v}_i \neq \mathbf{w}_i$ for some $i \in V$ then return $\perp$ (since $S_\mathbf{v} \not\supseteq S_\mathbf{w}$); else return

$$\begin{aligned} k_\mathbf{w} &:= e\big(k_\mathbf{v}, e(\{D_{i,v_i}\}_{i\in W\setminus V})\big) \\ &= e\big((g_{|V|})^{k\cdot\prod_{i\in V} d_{i,\mathbf{v}_i}}, (g_{|W\setminus V|})^{\prod_{i\in W\setminus V} d_{i,\mathbf{w}_i}}\big) = (g_{|W|})^{k\cdot\prod_{i\in W} d_{i,\mathbf{w}_i}} \quad . \end{aligned}$$

$\mathsf{Eval}(pp, k, x)$: – If $k \in \mathbb{Z}_p$, return $F(pp,k,x) = e(\{D_{i,x_i}\}_{i\in[n]})^k = g_n{}^{k\cdot\prod_{i\in[n]} d_{i,x_i}}$.

– Otherwise, $k = k_\mathbf{v}$ for some $\mathbf{v} \in \{0,1,?\}^n$. Let $V := \{i \in [n] : \mathbf{v}_i \neq ?\}$ and $\overline{V} := \{i \in [n] : \mathbf{v}_i = ?\}$ be its complement. If $x_i \neq \mathbf{v}_i$ for some $i \in V$ then return $\perp$ (since $x \notin S_\mathbf{v}$); else return

$$\begin{aligned} e\big(k_\mathbf{v}, e\big(\{D_{i,x_i}\}_{i\in\overline{V}}\big)\big) &= e\big((g_{|V|})^{k\cdot\prod_{i\in V} d_{i,\mathbf{v}_i}}, (g_{|\overline{V}|})^{\prod_{i\in\overline{V}} d_{i,x_i}}\big) \\ &= (g_n)^{k\cdot\prod_{i\in[n]} d_{i,x_i}} = F(pp,k,x) \quad . \end{aligned}$$

3.3 Properties

Key homomorphism. The construction is key-homomorphic in the sense of [BLMR13], but it also satisfies Definition 2, which requires that Constrain is homomorphic as well. The PRF can be described in the language of Definition 1 as follows. (Note that we identify the set $S_\mathbf{v}$, defined in (1), with the vector $\mathbf{v}$ defining it.)

- The set of vertices of the graph D is defined as $V := \{\mathbf{v} : \mathbf{v} \in \{0,1,?\}^n\}$ and the root node is $\mathbf{r} := (?, \ldots, ?)$, representing the set $\mathcal{X} = \{0,1\}^n$. There is an edge from $\mathbf{v}$ to $\mathbf{w}$ if all bits fixed by $\mathbf{v}$ are fixed by $\mathbf{w}$ to the same value, i.e., for all $i \in [n]$: if $\mathbf{v}_i \in \{0,1\}$ then $\mathbf{w}_i = \mathbf{v}_i$.
- The additive group associated to $\mathbf{r}$ is $\mathcal{K}_\mathbf{r} := (\mathbb{Z}_p, +)$; for all other vertices $\mathbf{v}$ it is $\mathcal{K}_\mathbf{v} := (\mathbb{G}_{|V|}, \cdot)$ with $V := \{i \in [n] : \mathbf{v}_i \neq ?\}$, i.e., the positions of 0's and 1's in $\mathbf{v}$. For all $\mathbf{v}$, the distribution $\mathcal{D}_\mathbf{v}$ is the uniform distribution over $\mathcal{K}_\mathbf{v}$.
- $C_{\mathbf{v},\mathbf{w}} : \mathcal{K}_\mathbf{v} \to \mathcal{K}_\mathbf{w}$, for all $\mathbf{v}, \mathbf{w}$ for which $(\mathbf{v}, \mathbf{w})$ is an edge in D, is defined as

$$C_{\mathbf{v},\mathbf{w}}(k) := \mathsf{Constrain}(pp, k, \mathbf{w}) \quad .$$

(Note that for $\mathbf{w} \in \{0,1\}$ we have $\mathsf{Constrain}(pp, k, \mathbf{w}) = \mathsf{Eval}(pp, k, \mathbf{w})$.)

By construction, running $\mathsf{Constrain}(pp, k_{\mathbf{v}}, \mathbf{w})$ on any key $k_{\mathbf{v}} \in \mathcal{K}_{\mathbf{v}}$ derived for some $\mathbf{v} \in \{0,1,?\}^n$ from some master key $k \in \mathbb{Z}_p$ always yields $(g_{|W|})^{k \cdot \prod_{i \in W} d_{i,\mathbf{w}_i}}$ if $(\mathbf{v}, \mathbf{w})$ is an edge in D. This shows consistency, which requires that for any nodes $\mathbf{v}, \mathbf{w} \in \{0,1,?\}^n$ and any two paths $P = (\mathbf{v} = \mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_k = \mathbf{w})$ and $P' = (\mathbf{v} = \mathbf{v}'_0, \mathbf{v}'_1, \ldots, \mathbf{v}'_\ell = \mathbf{w})$ from $\mathbf{v}$ to $\mathbf{w}$ in D, we have

$$C_{\mathbf{v}_{k-1},\mathbf{v}_k} \circ \cdots \circ C_{\mathbf{v}_1,\mathbf{v}_2} \circ C_{\mathbf{v}_0,\mathbf{v}_1} = C_{\mathbf{v}'_{\ell-1},\mathbf{v}'_\ell} \circ \cdots \circ C_{\mathbf{v}'_1,\mathbf{v}'_2} \circ C_{\mathbf{v}'_0,\mathbf{v}'_1} \ .$$

Finally, our construction is *homomorphic*, that is, for every edge $(\mathbf{v}, \mathbf{w})$ in D:

$$C_{\mathbf{v},\mathbf{w}}(k_1) \cdot C_{\mathbf{v},\mathbf{w}}(k_2) = C_{\mathbf{v},\mathbf{w}}(k_1 + k_2) \ . \qquad (2)$$

To show this, let $pp = (\mathbb{G}, \{D_{i,\beta}\}_{i \in [n],\, \beta \in \{0,1\}}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$. For all $k_1, k_2 \in \mathbb{Z}_p$ we then have the following:

1. $F(pp, k_1 + k_2, x) = {g_n}^{(k_1+k_2) \cdot \prod_{i \in [n]} d_{i,x_i}} = F(pp, k_1, x) \cdot F(pp, k_2, x)$.
2. For any $\mathbf{v} \in \{0,1,?\}^n$ we have:

$$\begin{aligned}\mathsf{Constrain}(pp, k_1 + k_2, \mathbf{v}) &= (g_{|V|})^{(k_1+k_2) \cdot \prod_{i \in V} d_{i,\mathbf{v}_i}} \\ &= \mathsf{Constrain}(pp, k_1, \mathbf{v}) \cdot \mathsf{Constrain}(pp, k_2, \mathbf{v}) \ .\end{aligned}$$

3. For any $\mathbf{v}, \mathbf{w} \in \{0,1,?\}^n$ for which $\mathbf{v}_i = ?$ or $\mathbf{v}_i = \mathbf{w}_i$ for all $i \in [n]$: if $k_{\mathbf{v}} = \mathsf{Constrain}(pp, k, \mathbf{v})$ and $k'_{\mathbf{v}} = \mathsf{Constrain}(pp, k', \mathbf{v})$ then

$$\begin{aligned}\mathsf{Constrain}(pp, k_{\mathbf{v}} \cdot k'_{\mathbf{v}}, \mathbf{w}) = e(k_{\mathbf{v}} \cdot k'_{\mathbf{v}}, D_{W \setminus V}(\mathbf{w})) &= (g_{|W|})^{(k+k') \cdot \prod_{i \in W} d_{i,\mathbf{w}_i}} \\ &= \mathsf{Constrain}(pp, k_{\mathbf{v}}, \mathbf{w}) \cdot \mathsf{Constrain}(pp, k'_{\mathbf{v}}, \mathbf{w}) \ .\end{aligned}$$

By 1. we have $C_{\mathbf{r},x}(k_1 + k_2) = C_{\mathbf{r},x}(k_1) \cdot C_{\mathbf{r},x}(k_2)$ for all $x \in \mathcal{X}$; by 2. we have $C_{\mathbf{r},\mathbf{v}}(k_1 + k_2) = C_{\mathbf{r},\mathbf{v}}(k_1) \cdot C_{\mathbf{r},\mathbf{v}}(k_2)$ for all $\mathbf{v}$; and by 3. we have $C_{\mathbf{v},\mathbf{w}}(k_1 + k_2) = C_{\mathbf{v},\mathbf{w}}(k_1) \cdot C_{\mathbf{v},\mathbf{w}}(k_2)$ for $\mathbf{v} \neq \mathbf{r}$; together this shows Equation (2).

Security. We show that publishing part of the secret key as parameters does not make the construction insecure. In particular, we show that our construction satisfies Definition 2, when the challenge set R is $\{0,1\}^n \subseteq V = \{0,1,?\}^n$, that is, the set of leaves of the DAG, which corresponds to the PRF domain $\mathcal{X} = \{0,1\}^n$.

We need to show that when $pp \leftarrow \mathsf{Setup}$ and $k \leftarrow \mathbb{Z}_p$ then an adversary that is given an oracle, which when queried on $\mathbf{v} \in \{0,1,?\}^n \setminus \{0,1\}^n$ returns $\mathsf{Constrain}(pp, k, \mathbf{v})$ and when queried on $x \in \{0,1\}^n$ returns either $F(pp, k, x)$ or a random element, cannot distinguish these two cases—provided it does not query a descendant $x \in R$ of some other query $u \in V$.

Theorem 1. *If there exists a PPT adversary breaking security of the above key-homomorphic bit-fixing PRF for n-bit-inputs, with challenge set $R = \{0,1\}^n$, with advantage $\varepsilon(\lambda)$ and making $q(\lambda)$ queries for challenge elements, then there exists a PPT algorithm that breaks the n-Multilinear Decisional Diffie-Hellman assumption with advantage $2^{-n} \cdot q(\lambda)^{-1} \cdot \varepsilon(\lambda)$.*

The proof first reduces the original game to a game where the adversary can only ask for one challenge query, which loses a factor $q(\lambda)$, by a standard hybrid argument. That game is then reduced to MDDH, following the proof from [BLMR13]; in particular, since in the simulation the reduction knows the values $\{D_{i,\beta}\}$, it can output them as public parameters (which do not exist in the original proof). See the full version for a detailed proof.

3.4 Circuit-Constrained PRF with Key-Homomorphic Evaluation

Boneh and Waters [BW13] give a second construction based on multilinear maps, which allows for constraining keys to more expressive sets, namely, all sets that can be decided by a circuit of some fixed depth. By defining public parameters, we construct a variant that is key-homomorphic as defined by Boneh *et al.* [BLMR13]. That is, we have that for all pp, k_1, k_2, x,

$$F(pp, k_1 + k_2, x) = F(pp, k_1, x) \cdot F(pp, k_2, x) \ . \tag{3}$$

However, our construction is not key-homomorphic in the sense of Definition 3, as the key-constraining function is not homomorphic.

The PRF is set up for input length n and circuit depth ℓ. The parameters are a sequence $(\mathbb{G}_1, \ldots, \mathbb{G}_\kappa)$ of groups $\mathbb{G}_i$ of prime order p, generated by g_i, where $\kappa = n+\ell$; as well as elements $D_{i,\beta}$, uniformly chosen from $\mathbb{G}_1$, for $i \in [n]$ and $\beta \in \{0,1\}$. The secret-key space is $\mathbb{Z}_p$ and the PRF on input $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ is defined as

$$F(pp, k, x) := e\big(e(\{D_{i,x_i}\}_{i\in[n]})^k, g_\ell\big) \ = \ g_\kappa{}^{k \cdot \prod_{i\in[n]} d_{i,x_i}} \tag{4}$$

(with $d_{i,\beta}$ such that $D_{i,\beta} = g^{d_{i,\beta}}$). It is thus defined exactly as for the bit-fixing construction, except that there are more groups in the sequence, and satisfies (3).

Removing the values $d_{i,\beta}$ from the secret key of the construction in [BW13] entails another syntactical change. Above, we defined the PRF value F in terms of k and the parameter values $D_{i,\beta}$, whereas in [BW13] they are defined directly as the last term of Equation (4). In [BW13], components of constrained keys (those corresponding to input wires) are defined as $K_w := g^{r_w \cdot d_{w,1}}$, which we replace by $K_w := (D_{w,1})^{r_w}$.

The values $d_{i,\beta}$ are not used anywhere else. Our construction still satisfies pseudorandomness, since, as for the bit-fixing PRF, in the security proof the simulator knows the values $D_{i,\beta}$.

4 Prefix-Fixing Construction from LWE

In this section we prove that variants of the LWE-based key-homomorphic PRF of Banerjee and Peikert [BP14] also support prefix constraints, and that the constraining functions are key-homomorphic as well. After recalling some standard background and notation in Section 4.1, the contents of this section have the following high-level structure:

- In Section 4.2 we define a key-homomorphic, prefix-constrained pseudorandom function family called $\mathsf{Constrain}$, which we refer to as the "noisy" family. However, the functions in this family are highly sequential, with circuit depth proportional to the input length. More significantly, they have huge keys, of size *exponential* in the input length, so they cannot actually be used in reality. The purpose of defining this family is to give us a baseline object that has "perfect" consistency, homomorphic, and pseudorandomness properties (but terrible space and depth complexity), which we rely upon in the later subsections.
- In Section 4.3 we specialize the noisy $\mathsf{Constrain}$ family to be "errorless," i.e., all error terms are set to zero. We call the resulting family $\mathsf{PConstrain}$. As a specialization of $\mathsf{Constrain}$, it inherits that latter's perfect consistency and homomorphic properties. We show that the $\mathsf{PConstrain}$ functions (1) have small keys, (2) can be computed in low depth (e.g., logarithmic in the input length) by a slight modification to the $\mathsf{Constrain}$ algorithms, and (3) have outputs that are "close" to those of the noisy $\mathsf{Constrain}$ functions (under a mild condition on the input). However, we are still not quite done yet, because the errorless $\mathsf{PConstrain}$ functions are not pseudorandom.
- In Section 4.4 we combine the previous two families to obtain a family $\overline{\mathsf{PConstrain}}$ that has essentially all the desired properties: small keys, low depth, pseudorandomness, consistency, and homomorphism. (The latter two of these properties do not hold perfectly, but only *computationally*: no efficient adversary can make any queries that reveal a violation of either property.) The $\overline{\mathsf{PConstrain}}$ functions are defined simply as appropriately *rounded* functions from *errorless* family $\mathsf{PConstrain}$. As such, they inherit the latter's small keys and low depth. In addition, they are pseudorandom because they coincide with the rounded *noisy* pseudorandom $\mathsf{Constrain}$ functions; this follows from the fact that the (unrounded) errorless $\mathsf{PConstrain}$ and noisy $\mathsf{Constrain}$ functions have "close" outputs, and the rounding precision is taken to be sufficiently coarse to conceal this difference. Finally, consistency and homomorphism hold for $\overline{\mathsf{PConstrain}}$ essentially because rounding can be seen as adding a particular kind of (deterministic) error, so $\overline{\mathsf{PConstrain}}$ may be seen as an instantiation of $\mathsf{Constrain}$.

4.1 Preliminaries

We first recall some standard background from [MP12,BP14]. For an integer modulus $q \geq 1$, let $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ denote the quotient ring of integers modulo q, where for convenience we always let $q = 2^\ell$ be a power of two. For $\ell = \log q \geq 2$, define the "gadget" (row) vector

$$\mathbf{g} = (1, 2, 4, \ldots, 2^{\ell-1}) \in \mathbb{Z}_q^\ell \ ,$$

and the (deterministic) "binary decomposition" function $\mathbf{g}^{-1} \colon \mathbb{Z}_q \to \{0,1\}^\ell$ as follows: identifying each $a \in \mathbb{Z}_q$ with its integer representative in $\{0, \ldots, q-1\}$, let $\mathbf{g}^{-1}(a) = (x_0, x_1, \ldots, x_{\ell-1}) \in \{0,1\}^\ell$ where $a = \sum_{i=0}^{\ell-1} x_i 2^i$ is the binary representation of a. Note that by definition, $\langle \mathbf{g}, \mathbf{g}^{-1}(a) \rangle = a$ for all $a \in \mathbb{Z}_q$, which explains our choice of notation.

Similarly, for vectors and matrices over $\mathbb{Z}_q$ we define the function $\mathbf{G}^{-1}\colon \mathbb{Z}_q^{n\times m} \to \{0,1\}^{n\ell\times m}$ by applying $\mathbf{g}^{-1}$ entry-wise. Notice that for all $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ we have

$$\mathbf{G}\cdot\mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}, \quad \text{where} \quad \mathbf{G} = \mathbf{I}_n \otimes \mathbf{g} = \operatorname{diag}(\mathbf{g},\ldots,\mathbf{g}) \in \mathbb{Z}_q^{n\times n\ell}$$

is the block-diagonal matrix having n copies of $\mathbf{g}$ as diagonal blocks, and zeros elsewhere. We let $\mathcal{P} \subseteq \mathbb{Z}^{n\ell}$ denote a certain set of canonical representatives of the additive quotient group $\mathbb{Z}_q^{n\ell}/(\mathbb{Z}_q^n \cdot \mathbf{G})$. Specifically, as shown in [MP12], we can use[4]

$$\mathcal{P} := \{-\tfrac{q}{4},\ldots,\tfrac{q}{4}-1\}^{n\ell} \ .$$

We define a bijection $\mathsf{Decode}\colon \mathbb{Z}_q^{n\ell} \to \mathcal{P}\times\mathbb{Z}_q^n$ as $\mathsf{Decode}(\mathbf{u}) = (\mathbf{v},\mathbf{s})$, where

$$\mathbf{u} = \mathbf{v} + \mathbf{s}\cdot\mathbf{G} \ .$$

As shown in [MP12], there is an efficient algorithm for computing Decode in depth proportional to $\ell = \log q$, and clearly $\mathsf{Decode}^{-1}(\mathbf{v},\mathbf{s}) = \mathbf{v} + \mathbf{s}\cdot\mathbf{G}$.

We recall the following easy lemma about the spectral norm, denoted $s_1(\cdot)$, of binary matrices. (See, e.g, [BP14, Lemma 3.1] for a proof.) Recall that $s_1(\mathbf{M}) = \max_{\|\mathbf{u}\|=1}\|\mathbf{u}\mathbf{M}\|$, where the maximum is taken over real unit vectors $\mathbf{u}$.

Lemma 1. *If $\mathbf{S}$ is a binary (i.e., 0-1) m-by-m matrix, then $s_1(\mathbf{S}) \le m$.*

Binary trees. A *full* binary tree T is one in which each node is either a leaf, or has two (nonempty) children. We let $|T|$ denote the number of leaves in T, and index the leaves from 0 to $|T|-1$ by the inorder traversal of T. If $|T| \ge 1$, we let $T.l$ and $T.r$ respectively denote its left and right subtrees, both of which are nonempty.

Given matrices $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n\times n\ell}$, we define the function $\mathbf{A}_T(x)\colon \{0,1\}^{|T|} \to \mathbb{Z}_q^{n\times n\ell}$ as follows:

$$\mathbf{A}_T(x) := \begin{cases} \mathbf{A}_x & \text{if } |T| = 1, \\ \mathbf{A}_{T.l}(x_l)\cdot\mathbf{G}^{-1}(\mathbf{A}_{T.r}(x_r)) & \text{otherwise,} \end{cases}$$

where in the second case we parse the input $x = x_l x_r$ where $|x_l| = |T.l|$ and $|x_r| = |T.r|$.

Rounding. For a positive integer e, we define the integer rounding function $\lfloor\cdot\rceil_e\colon \mathbb{Z} \to e\mathbb{Z}$ as $\lfloor x\rceil_e := \lfloor x/e\rceil \cdot e$, and extend it component-wise to vectors and matrices. In words, $\lfloor x\rceil$ simply rounds x to the nearest integer multiple of e.[5]

[4] This choice of $\mathcal{P}$ is possible because we have taken q to be a power of two. It may be possible to generalize our results to other values of q using the alternative lattice bases given in [MP12], but it seems to substantially complicate the proofs.

[5] We point out that this function differs slightly from the “modular” rounding function considered in prior works, which mapped $\mathbb{Z}_q$ to $\mathbb{Z}_p$ as $\lfloor x\rceil_p = \lfloor x\cdot p/q\rceil \bmod p$. Here e corresponds with q/p, but the rounding input and output have the same “scale.”

4.2 "Noisy" Function Family $\mathcal{C}$

As in previous work [BPR12,BP14], we first define and analyze a certain family $\mathcal{C}$ of "noisy" constraining functions, which have *huge* (exponential-size) keys, because each key contains many error terms. To avoid technical complications related to efficient computation on exponential-size inputs, throughout this section the error terms are always sampled "lazily," i.e., not until they are needed. In Section 4.2 we show that the constraining functions are consistent, in Section 4.2 we show that they are homomorphisms under an appropriate group operation on the key spaces, and in Section 4.2 we show that the family is pseudorandom.

The public parameters of the noisy family are two matrices $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n \times n\ell}$, chosen uniformly at random. Following Definitions 1 and 3, to describe our family we need to give a DAG with a unique root node, a key space with an additive group structure for each node in the DAG, and a constraining function for each edge in the DAG.

DAG. For a full binary tree T, our DAG corresponds to prefix-fixing constraints on $\{0,1\}^{|T|}$, i.e., the nodes are identified with the strings in $\{0,1\}^{\leq |T|}$, and there is an edge (w, wx) for every w and $x \neq \varepsilon$ such that $|wx| \leq |T|$. This DAG clearly has a unique root node, namely, the empty string ε.

Key spaces. For any full binary tree T and $0 \leq j < |T|$, we define

$$\mathcal{R}_{T,j} := \begin{cases} \mathbb{Z}_q^n & \text{if } |T| = 1, \\ \mathcal{R}_{T.l,j} & \text{if } j \leq |T.l|, \\ \mathcal{P} \times \mathcal{R}_{T.r,j-|T.l|} & \text{otherwise.} \end{cases}$$

For convenience in our recursive algorithms, we also define $\mathcal{R}_{T,|T|} = \mathcal{P} \times \mathbb{Z}_q^n$. In words, $\mathcal{R}_{T,j}$ has one $\mathcal{P}$-component for each *left* subtree "hanging off" the path from the root to the jth leaf. (Recall that we number the leaves starting from zero.) We also define, for $0 \leq j \leq |T|$,

$$\mathcal{E}_{T,j} := \prod_{y \in \{0,1\}^{\leq |T|-j}} \mathbb{Z}^{n\ell} = (\mathbb{Z}^{n\ell})^{2^{|T|-j+1}-1} \ .$$

In $\mathcal{E}_{T,j}$, the several $\mathbb{Z}^{n\ell}$ components (which represent the error vectors) are indexed by the binary strings of length at most $|T| - j$, which is why there are $2^{|T|-j+1} - 1$ components.

For each $w \in \{0,1\}^{\leq |T|}$, the key space $\mathcal{K}_{T,w}$ and associated distribution for w are defined as:

$$\begin{aligned} \mathcal{K}_{T,w} &:= \mathcal{R}_{T,|w|} \times \mathcal{E}_{T,|w|} \ , \\ \mathcal{D}_{T,w} &:= U(\mathcal{R}_{T,|w|}) \times (\chi^{nl})^{2^{|T|-|w|+1}-1} \ , \end{aligned}$$

where χ is some error distribution over $\mathbb{Z}$ that will be used in the security proof. Note that $\mathcal{K}_{T,w}$ does not depend on the actual bits in w, only on its length $|w|$.

To make $\mathcal{K}_{T,w}$ an additive group (for $|w| > 0$), we stress that we do *not* simply treat it as a product group of its components—indeed, $\mathcal{P} \subset \mathbb{Z}^{n\ell}$ is not even closed under addition, so it is not a group. Instead, in Section 4.2 below we define a special addition operation on $\mathcal{R}_{T,|w|}$ to make it a group. Then $\mathcal{K}_{T,w}$ is simply the product group of this group with $\mathcal{E}_{T,|w|}$, with the usual addition operation on the latter.

Constraining functions. It now remains to define (consistent) constraining functions $\mathsf{Constrain}_{T,w,x} \colon \mathcal{K}_{T,w} \to \mathcal{K}_{T,wx}$ for all strings w, x such that $x \neq \varepsilon$ and $|wx| \leq |T|$; for convenience, we also define $\mathsf{Constrain}_{T,w,x}$ to be the identity function for $x = \varepsilon$. Functional pseudocode for the constraining functions is given in Algorithm 4.1. We remark that it would have been sufficient to define functions $\mathsf{Constrain}_{T,x,w}$ for $|x| = 1$ alone. Indeed, by Lemma 2 below it follows that our pseudocode is actually *equivalent* to the sequential composition of such functions, and hence has circuit depth proportional to $|x|$. We choose to present the constraining functions for general x here because in Section 4.4 we show that a slight modification yields highly parallel functions.

In summary, the constraining functions are defined recursively on the tree structure. In the base case $|T| = 1$, for key $(\mathbf{v}, (\mathbf{e}_x)_{x \in \{0,1\}^{\leq 1}}) \in \mathcal{K}_{T,w} = \mathbb{Z}_q^n \times (\mathbb{Z}^{n\ell})^3$, we simply compute and decode the "noisy" value $\mathbf{v}\mathbf{A}_x + \mathbf{e}_x \in \mathbb{Z}_q^{n\ell}$. There are three recursive cases, depending on whether we are constraining entirely within the left subtree, within the right subtree, or across the two subtrees. In the first two cases, we simply recurse on the appropriate subtree. In the third case, we recursively constrain over the remainder of the left subtree, then over the desired portion of the right subtree. Lastly, whenever we finish constraining over an *entire* (sub)tree we need to appropriately "fold" the results, which consist of some leftover value in $\mathcal{P} \subset \mathbb{Z}^{n\ell}$ from the left subtree and some value in $\mathcal{P} \times \mathbb{Z}_q^n$ from the completed right subtree, into a value in $\mathcal{P} \times \mathbb{Z}_q^n$ for the entire tree.

We remark that although our presentation is (necessarily) quite different, our constraining functions correspond to the *partial* evaluations of the noisy function family from [BP14], which the simulator computes internally when answering queries in the security proof.

Consistency. We first show consistency.

Lemma 2 (Consistency). *For any full binary tree T, parameters $\mathbf{A}_0, \mathbf{A}_1$, and strings w, x, z where $|wxz| \leq |T|$, we have that*

$$\mathsf{Constrain}_{T,wx,z} \circ \mathsf{Constrain}_{T,w,x} = \mathsf{Constrain}_{T,w,xz} \quad .$$

Proof. We proceed by induction on $|T|$. The base case of $|T| = 1$ is trivial, because $\mathsf{Constrain}_{T,w,\varepsilon}$ is the identity function.

We have three inductive cases. In the first two cases, where $|wxz| \leq |T.l|$ or $|w| \geq |T.l|$, the claim follows immediately by the inductive hypothesis on $T.l$ or $T.r$, respectively. The last inductive case is $|w| < |T.l|$ and $|wxz| > |T.l|$. We analyze this in two subcases.

The first subcase is $|wx| > |T.l|$. Here we parse $x = x_l x_r$ with $|wx_l| = |T.l|$. By definition, we have

$$\mathsf{Constrain}_{T,w,x} = \mathsf{Constrain}_{T,wx_l,x_r} \circ \mathsf{Constrain}_{T,w,x_l}$$
$$\mathsf{Constrain}_{T,w,xz} = \mathsf{Constrain}_{T,wx_l,x_rz} \circ \mathsf{Constrain}_{T,w,x_l} \quad .$$

Algorithm 4.1 $\mathsf{Constrain}_{T,w,x} \colon \mathcal{K}_{T,w} \to \mathcal{K}_{T,wx}$ for $|wx| \le |T|$, $x \neq \varepsilon$

```
Input: $(\mathbf{v}, (\mathbf{e}_y)_{|y| \le |T|-|w|}) \in \mathcal{K}_{T,w} = \mathcal{R}_{T,|w|} \times \mathcal{E}_{T,|w|}$
 1: if $|T| = 1$ then                                          ▷ base case, so $\mathbf{v} \in \mathbb{Z}_q^n$
 2:     return $\mathsf{Decode}(\mathbf{v} \cdot \mathbf{A}_x + \mathbf{e}_x)$
 3: else if $|wx| \le |T.l|$ then                ▷ constrains entirely in left subtree...
 4:     return $\big(\mathsf{Constrain}_{T.l,w,x}\big(\mathbf{v}, (\mathbf{e}_y)_{|y| \le |T.l|-|w|}\big), (\mathbf{e}_{xy})_{|y| \le |T|-|wx|}\big)$   ▷ ...so just
        recurse.
 5: else if $|w| < |T.l|$ then                               ▷ incomplete left subtree...
 6:     parse $x = x_l x_r$ where $|wx_l| = |T.l|$
 7:     let $(\mathbf{v}_l, \star) = \mathsf{Constrain}_{T,w,x_l}\big(\mathbf{v}, (\mathbf{e}_y)_{|y| \le |T.l|-|w|}\big)$     ▷ ...complete left subtree
        (self-recurse)...
 8:     return $\mathsf{Constrain}_{T,wx_l,x_r}\big(\mathbf{v}_l, (\mathbf{e}_{x_l y})_{|y| \le |T.r|}\big)$     ▷ ...and self-recurse to finish.
 9: else                                              ▷ constrains entirely in right subtree...
10:     parse $w = w_l w_r$ where $|w_l| = |T.l|$ and $\mathbf{v} = (\mathbf{v}_l, \mathbf{v}_r) \in \mathcal{P} \times \mathcal{R}_{T.r,|w_r|}$
11:     let $(\mathbf{k}_r, \star) = \mathsf{Constrain}_{T.r,w_r,x}\big(\mathbf{v}_r, (\mathbf{e}_y)_{|y| \le |T|-|w|}\big)$                ▷ ...so recurse.
12:     if $|wx| = |T|$ then             ▷ constrains over entire tree (so $\mathbf{k}_r \in \mathcal{P} \times \mathbb{Z}_q^n$)...
13:         return $\mathsf{Decode}(\mathbf{v}_l \cdot \mathbf{G}^{-1}(\mathbf{A}_{T.r}(w_r x)) + \mathsf{Decode}^{-1}(\mathbf{k}_r))$ ▷ ...so fold results.
14:     else                                                  ▷ doesn't complete the tree...
15:         return $(\mathbf{v}_l, \mathbf{k}_r)$                                        ▷ ...so append results.
```

The claim then follows by the inductive hypothesis on $T.r$, by composing $\mathsf{Constrain}_{T,wx,z}$ on the left of the first equation above.

The second subcase is $|wx| \leq |T.l|$. This proceeds essentially identically to the first subcase, where we instead parse $z = z_l z_r$ with $|wxz_l| = |T.l|$.

Homomorphism. Before we can prove that the constraining functions are homomorphisms, we must make $\mathcal{K}_{T,w} = \mathcal{R}_{T,|w|} \times \mathcal{E}_{T,|w|}$ an additive group for $|w| > 0$. (Recall that $\mathcal{R}_{T,0} = \mathbb{Z}_q^n$, which is already a group.) We do so by defining a special group operation $\mathsf{Add}_{T,w}$ on the set $\mathcal{R}_{T,|w|}$—note that the operation depends on w itself, not just its length. We then let $\mathcal{K}_{T,w}$ be the product group with $\mathcal{E}_{T,|w|}$, under its usual addition operation. For convenience, we overload $\mathsf{Add}_{T,w}$ to also refer to the group operation for this product group, where the intended domain should be clear by context.

For convenience in the recursive definitions, we let $\mathsf{Add}_{T,w}$ take an auxiliary input $\mathbf{t} \in \mathbb{Z}_q^n$, which should be thought of as a kind of "carry" term that comes from reducing the sum of two $\mathcal{P}$-elements (in $\mathbb{Z}^{n\ell}$) back to $\mathcal{P}$. Initializing this carry input to zero yields the group operation. Formally, we define

$$\mathsf{Add}_{T,w}\big(\mathbf{t} \in \mathbb{Z}_q^n, \mathbf{v}, \mathbf{v}'\big) := \begin{cases} \mathbf{t} + \mathbf{v} + \mathbf{v}' & \text{if } |T| = 1, |w| = 0, \\ \mathsf{Add}_{T.l,w}(\mathbf{t}, \mathbf{v}, \mathbf{v}') & \text{if } |w| \leq |T.l|, \\ \Big(\bar{\mathbf{v}}_l, \mathsf{Add}_{T.r,w_r}\big(\bar{\mathbf{t}}, \mathbf{v}_r, \mathbf{v}'_r\big)\Big) & \text{if } |T.l| < |w| < |T|, \\ \mathsf{Decode}\big(\mathbf{t} \cdot \mathbf{A}_T(w) + \mathsf{Decode}^{-1}(\mathbf{v}) + \mathsf{Decode}^{-1}(\mathbf{v}')\big) & \text{if } |w| = |T|, \end{cases}$$

where in the third case we parse $w = w_l w_r$ for $|w_l| = |T.l|$ and $\mathbf{v} = (\mathbf{v}_l, \mathbf{v}_r), \mathbf{v}' = (\mathbf{v}'_l, \mathbf{v}'_r) \in \mathcal{P} \times \mathcal{R}_{T.r,|w_r|}$, and let $(\bar{\mathbf{v}}_l, \bar{\mathbf{t}}) = \mathsf{Decode}(\mathbf{t} \cdot \mathbf{A}_{T.l}(w_l) + \mathbf{v}_l + \mathbf{v}'_l)$.

We now prove that the $\mathsf{Constrain}$ functions are homomorphisms.

Lemma 3 (Homomorphism). *For any parameters $\mathbf{A}_0, \mathbf{A}_1$ and any full binary tree T, any bit strings w, x such that $|wx| \leq |T|$, and any $\mathbf{t} \in \mathbb{Z}_q^n$ and $\mathbf{k}, \mathbf{k}' \in \mathcal{K}_{T,w}$, we have*

$$\begin{aligned} &\mathsf{Constrain}_{T,w,x}(\mathsf{Add}_{T,w}(\mathbf{t}, \mathbf{k}, \mathbf{k}')) \\ &\qquad = \mathsf{Add}_{T,wx}(\mathbf{t}, \mathsf{Constrain}_{T,w,x}(\mathbf{k}), \mathsf{Constrain}_{T,w,x}(\mathbf{k}')) \ . \end{aligned} \tag{5}$$

In particular, by setting $\mathbf{t} = \mathbf{0}$ we have that $\mathsf{Constrain}_{T,w,x}$ is an additive homomorphism.

Proof. The claim is trivial for $x = \varepsilon$, so from now on we assume that $x \neq \varepsilon$. Let $i = |w|$ and $j = |wx|$, so $0 \leq i < j \leq |T|$. Parse $\mathbf{k} = (\mathbf{v}, (\mathbf{e}_y))$, $\mathbf{k}' = (\mathbf{v}', (\mathbf{e}'_y))$, and let $\bar{\mathbf{k}} = (\bar{\mathbf{v}}, (\bar{\mathbf{e}}_y)) = \mathsf{Add}_{T,w}(\mathbf{t}, \mathbf{k}, \mathbf{k}')$.

As usual, we proceed by induction over $|T|$. In the base case, where $w = \varepsilon$ and $|x| = |T| = 1$, we have

$$\begin{aligned}\mathsf{Constrain}_{T,\varepsilon,x}(\bar{\mathbf{v}}, (\bar{\mathbf{e}}_y)) &= \mathsf{Decode}(\bar{\mathbf{v}} \cdot \mathbf{A}_x + \bar{\mathbf{e}}_x) \\ &= \mathsf{Decode}((\mathbf{t} + \mathbf{v} + \mathbf{v}') \cdot \mathbf{A}_x + (\mathbf{e}_x + \mathbf{e}'_x)) \\ &= \mathsf{Decode}(\mathbf{t} \cdot \mathbf{A}_x + (\mathbf{v} \cdot \mathbf{A}_x + \mathbf{e}_x) + (\mathbf{v}' \cdot \mathbf{A}_x + \mathbf{e}'_x)) \\ &= \mathsf{Add}_{T,x}(\mathbf{t}, \mathsf{Constrain}(\mathbf{v}, (\mathbf{e}_y)), \mathsf{Constrain}(\mathbf{v}', (\mathbf{e}'_y))) \ .\end{aligned}$$

We now consider the inductive cases. Because $\mathsf{Constrain}$ simply passes an appropriate subset of the input error terms (the $\mathbb{Z}^{n\ell}$ vectors in the key) to the output, for simplicity of exposition we suppress the error terms in the remainder of the proof. The final claim then follows by the product group structure of $\mathcal{K}_{T,w}$.

The first inductive case, where $i < j \le |T.l|$, follows immediately from the inductive hypothesis on $\mathsf{Constrain}_{T.l,w,x}$. For the second inductive case, where $i < |T.l| < j$, we defer to the final paragrap of the proof. For the third inductive case, where $|T.l| \le i$, parse $w = w_l w_r$ and $\mathbf{v} = (\mathbf{v}_l, \mathbf{v}_r), \mathbf{v}' = (\mathbf{v}'_l, \mathbf{v}'_r), \bar{\mathbf{v}} = (\bar{\mathbf{v}}_l, \bar{\mathbf{v}}_r)$, and note that by definition,

$$\bar{\mathbf{v}}_r = \mathsf{Add}_{T.r,w_r x}(\bar{\mathbf{t}}, \mathbf{v}_r, \mathbf{v}'_r) \ , \tag{6}$$

$$\bar{\mathbf{v}}_l + \bar{\mathbf{t}} \cdot \mathbf{G} = \mathbf{t} \cdot \mathbf{A}_{T.l}(w_l) + \mathbf{v}_l + \mathbf{v}'_l \tag{7}$$

for some $\bar{\mathbf{t}} \in \mathbb{Z}_q^n$. As in the code for $\mathsf{Constrain}$, let $\mathbf{k}_r = \mathsf{Constrain}_{T.r,w_r,x}(\mathbf{v}_r)$ and similarly for $\mathbf{k}'_r, \bar{\mathbf{k}}_r$. Then by the inductive hypothesis on $\mathsf{Constrain}_{T.r,w_r,x}$ and Equation (6), we have

$$\bar{\mathbf{k}}_r = \mathsf{Constrain}_{T.r,w_r,x}(\bar{\mathbf{v}}_r) = \mathsf{Add}_{T.r,w_r x}(\bar{\mathbf{t}}, \mathbf{k}_r, \mathbf{k}'_r) \ . \tag{8}$$

Now if $j = |wx| < |T|$, then by definition of $\mathsf{Constrain}_{T,w,x}$ and $\mathsf{Add}_{T,wx}$ (respectively), the left- and right-hand sides of Equation (5) are respectively just $\bar{\mathbf{v}}_l$ prepended to the left- and right-hand sides of Equation (8), so they are equal. But if $j = |wx| = |T|$, then the output of $\mathsf{Constrain}_{T,w,x}(\bar{\mathbf{v}})$ is “folded,” (i.e., in $\mathcal{P} \times \mathbb{Z}_q^n$), as are $\mathbf{k}_r, \mathbf{k}'_r, \bar{\mathbf{k}}_r$ as defined above. We proceed by applying the folding operation to both sides of Equation (8), namely, apply Decode^{-1}, add $\bar{\mathbf{v}}_l \cdot \mathbf{G}^{-1}(\mathbf{A}_{T.r}(w_r x))$, and apply Decode. For the left-hand side we get exactly $\mathsf{Constrain}_{T,w,x}(\bar{\mathbf{v}})$, which is the left-hand side of Equation (5). On the right-hand side, by definition of $\mathsf{Add}_{T.r,w_r x}$, by Equation (7), and by definition of $\mathbf{A}_T(\cdot)$, we get Decode applied to

$$\begin{aligned}&\bar{\mathbf{v}}_l \cdot \mathbf{G}^{-1}(\mathbf{A}_{T.r}(w_r x)) + \mathsf{Decode}^{-1}(\mathsf{Add}_{T.r,w_r x}(\bar{\mathbf{t}}, \mathbf{k}_r, \mathbf{k}'_r)) \\ &\quad= (\bar{\mathbf{v}}_l + \bar{\mathbf{t}} \cdot \mathbf{G}) \cdot \mathbf{G}^{-1}(\mathbf{A}_{T.r}(w_r x)) + \mathsf{Decode}^{-1}(\mathbf{k}_r) + \mathsf{Decode}^{-1}(\mathbf{k}'_r) \\ &\quad= (\mathbf{t} \cdot \mathbf{A}_{T.l}(w_l) + \mathbf{v}_l + \mathbf{v}'_l) \cdot \mathbf{G}^{-1}(\cdots) + \mathsf{Decode}^{-1}(\mathbf{k}_r) + \mathsf{Decode}^{-1}(\mathbf{k}'_r) \\ &\quad= \mathbf{t} \cdot \mathbf{A}_T(wx) + (\mathbf{v}_l \cdot \mathbf{G}^{-1}(\cdots) + \mathsf{Decode}^{-1}(\mathbf{k}_r)) \\ &\qquad\qquad\qquad\qquad + (\mathbf{v}'_l \cdot \mathbf{G}^{-1}(\cdots) + \mathsf{Decode}^{-1}(\mathbf{k}'_r)) \ .\end{aligned}$$

As desired, this is the right-hand side of Equation (5), by definition of $\mathsf{Constrain}_{T,w,x}$ and $\mathsf{Add}_{T,wx}$.

Going back to the second inductive case, where $i < |T.l| < j$, it follows by writing $\mathsf{Constrain}_{T,w,x} = \mathsf{Constrain}_{T,wx_l,x_r} \circ \mathsf{Constrain}_{T,w,x_l}$ where $x = x_l x_r$ for

$|wx_l| = |T.l|$, then applying the inductive hypothesis on $T.l$ and $T.r$. This completes the proof of Lemma 3.

Pseudorandomness. We now show that the function family $\mathcal{C}$ defined above is pseudorandom according to Definition 2, with *all nodes* $R = \{0,1\}^{\leq |T|}$ as challenge nodes. This follows immediately from the PRG-like property demonstrated in Lemma 4 below, together with the fact (shown in prior works [BGI14,KPTZ13,BW13]) that the GGM construction [GGM86] instantiated with such a PRG family yields a prefix-constrained PRF.[6] In a bit more detail: in the following lemma we show that for any string $w \in \{0,1\}^{<|T|}$, the function $G_{T,w} \colon \mathcal{K}_{T,w} \to \mathcal{K}_{T,w0} \times \mathcal{K}_{T,w1}$ defined as $G(\mathbf{k}) = (\mathsf{Constrain}_{T,w,0}(\mathbf{k}), \mathsf{Constrain}_{T,w,1}(\mathbf{k}))$ is a pseudorandom generator, under the LWE assumption. Instantiating the GGM construction with these PRGs yields our constraining functions $\mathsf{Constrain}_{T,w,x}$, therefore they are pseudorandom.

Lemma 4. *Let T be any full binary tree and $w \in \{0,1\}^{<|T|}$ be any string. Then assuming the hardness of decision-$\mathsf{LWE}_{n,q,\chi}$, for $\mathbf{k} \leftarrow \mathcal{D}_{T,w}$ we have*

$$(\mathsf{Constrain}_{T,w,0}(\mathbf{k}), \mathsf{Constrain}_{T,w,1}(\mathbf{k})) \overset{c}{\approx} \mathcal{D}_{T,w0} \times \mathcal{D}_{T,w1} \ .$$

The proof of this lemma involves a simulator embedding the appropriate LWE challenge in the base case in the computation of $\mathsf{Constrain}_{T,w,b}$. The rest of the proof consists of showing the outputs corresponding to each distribution (LWE vs uniform) are distributed accordingly. We defer the details to the full version.

4.3 Parallel Errorless Constrain

In this subsection we consider the "errorless" variants of our Constrain functions, which we call PConstrain, and show that they can be computed in low depth. We also show that the output of PConstrain is typically close to that of Constrain, when the errors used in the latter are small.

Parallel Evaluation. The PConstrain functions simply correspond to the Constrain functions when all the error vectors are set to zero, that is, $\mathsf{PConstrain}_{T,w,x}(\mathbf{v}, \mathbf{s}) = \mathsf{Constrain}_{T,w,x}(\mathbf{v}, \mathbf{s}, \mathbf{0})$. In particular, this implies that the PConstrain functions are both consistent and homomorphisms, because the Constrain functions are. In addition, the errorless setting allows PConstrain to be computed with good parallelism (i.e., in low depth) by an alternative algorithm that "short circuits" the computation via a base case that constrains over an *entire (sub)tree* in just one step. More specifically, we modify the base case (Lines 1 and 2) of Algorithm 4.1 as shown in Algorithm 4.2 below. The rest of the algorithm remains unchanged, apart from the fact that PConstrain does not take or output any error terms.

In Lemma 5 we prove that the alternative algorithm is correct. Then in Section 4.3 we describe how PConstrain can be evaluated in low depth.

[6] It is easy to verify that this remains true even for our slightly stronger definition, where the adversary can query the function at inputs corresponding to internal nodes of the GGM tree.

Algorithm 4.2 $\mathsf{PConstrain}_{T,w,x} \colon \mathcal{R}_{T,|w|} \to \mathcal{R}_{T,|wx|}$ for $|wx| \leq |T|$, $x \neq \varepsilon$

```
Input: $\mathbf{v} \in \mathcal{R}_{T,|w|}$
1: if $w = \varepsilon$ and $|x| = |T|$ then                                    ▷ base case
2:     return $\mathsf{Decode}(\mathbf{v} \cdot \mathbf{A}_T(x))$
3: The remaining code is the same as in Algorithm 4.1, but without any error terms
   $(\mathbf{e}_y)$.
```

For convenience, we define the function $\mathsf{Project}_{T,w} \colon \mathcal{K}_{T,w} \to \mathcal{R}_{T,|w|}$, which just outputs the $\mathcal{R}_{T,|w|}$-component of its input (dropping the $\mathcal{E}_{T,|w|}$-component, i.e., the errors), and $\mathsf{PrjConstrain}_{T,w,x} = \mathsf{Project}_{T,wx} \circ \mathsf{Constrain}_{T,w,x}$.

The following lemma states that what the algorithm above does is indeed correct. It is proved by a simple induction for *complete inputs* only, that is, for inputs $x = \varepsilon$ and $w \in \{0,1\}^{|T|}$. The complete proof appears in the full version.

Lemma 5. *For any fully binary tree T, any bit strings w, x with $|wx| \leq |T|$, and any $\mathbf{v} \in \mathcal{R}_{T,|w|}$,*

$$\mathsf{PConstrain}_{T,w,x}(\mathbf{v}) = \mathsf{PrjConstrain}_{T,w,x}(\mathbf{v}, \mathbf{0}) \ .$$

Parallel Evaluation of PConstrain. We now analyze the parallel complexity of the PConstrain functions according to Algorithm 4.2 (and Algorithm 4.1) above. Our main goal is to bound what we call the "nonlinear depth" of $\mathsf{PConstrain}_{T,w,x}$ in terms of the topology of T and the strings w, x. Nonlinear depth only takes into account the nonlinear Decode and $\mathbf{G}^{-1}$ operations; the remaining operations are all linear over $\mathbb{Z}_q$. For an implementation of PConstrain by an arithmetic or boolean circuit, the depth will depend on the precise circuit model used and the implementation of the linear and nonlinear operations, but in any case the final depth will be proportional to the nonlinear depth.

To state our claim we recall from [BP14] the notions of "left depth" and "right depth" of the jth leaf in a binary tree T, and of T itself. The left depth $l_T(j)$ (respectively, right depth $r_T(j)$) of the jth leaf is the number of edges from a parent to its left (resp., right) child on the path from the root to that leaf. The left and right depths $l(T), r(T)$ are respectively the maximum left and right depths over all leaves in T.

Lemma 6. *The function $\mathsf{PConstrain}_{T,w,x}(\mathbf{v})$ can be computed via (1) a preprocessing phase (independent of $\mathbf{v}$) of nonlinear depth at most $r(T)$, and (2) an online phase (dependent on $\mathbf{v}$) of nonlinear depth at most $l_T(|w|) + r_T(|x|) \leq l(T) + r(T)$.*

We remark that in [BP14], the nonlinear depth of computing the (non-constrained) PRF is just $r(T)$, so one can obtain an extremely parallel PRF using a "left spine" tree with $r(T) = 1$ and $l(T) = |T| - 1$ (this corresponds to the function from [BLMR13]). But here, evaluating the PRF from a constrained key can require nonlinear depth proportional to the sum of T's left and right depths. Therefore, to get good parallelism for all w, x we must use a shallow tree T, e.g., one with depth $O(\log|T|)$. We defer the proof of this Lemma to the full version.

Closeness. We next analyze the size of the $\mathcal{P}$-components of $\mathbf{d}$ discussed above, as they relate to the errors in the original key $\mathbf{k}_\varepsilon = (\mathbf{s}, (\mathbf{e}_y))$. Recalling that each $\mathcal{P}$-component of $\mathbf{d}$ corresponds to some left-child subtree in T, it is therefore sufficient to analyze the accumulated error in fully constrained keys over arbitrary trees. For this purpose we define a "growth factor" Φ_T associated with an arbitrary full binary tree, defined recursively as follows:

$$\Phi_T := \begin{cases} 1 & \text{if } |T| = 1, \\ \sqrt{(\Phi_{T.l} \cdot n\ell)^2 + (\Phi_{T.r})^2} & \text{otherwise.} \end{cases} \tag{9}$$

We next state a lemma that is essentially a restatement of [BP14, Lemma 3.7].

Lemma 7 (Error Bound). *For any w such that $|w| = |T|$, let*

$$(\mathbf{k}, \star) = \mathsf{Constrain}_{T,\varepsilon,w}(\mathbf{0}, (\mathbf{e}_y))$$

for $(\mathbf{e}_y) \leftarrow \mathcal{E}_{T,0}$, where the error distribution χ is subgaussian with parameter r. Then $\mathsf{Decode}^{-1}(\mathbf{k}) = \mathbf{e} \pmod q$ for some $\mathbf{e} \in \mathbb{Z}^{n\ell}$ that is subgaussian with parameter $r \cdot \Phi_T$.

More generally, let $\mathbf{d} = \mathsf{PrjConstrain}_{T,\varepsilon,w}(\mathbf{0}, (\mathbf{e}_y)) \in \mathcal{R}_{T,|w|}$ for nonempty $w \in \{0,1\}^{\le |T|}$. Then if $q \ge 4r \cdot \Phi_T \cdot \omega(\sqrt{\log \lambda})$, the following are true with $1 - \mathrm{negl}(\lambda)$ probability over the choice of $(\mathbf{e}_y) \leftarrow \mathcal{E}_{T,0}$: (1) the $\mathbb{Z}_q^n$-component of $\mathbf{d}$ is $\mathbf{0}$, and (2) each $\mathcal{P}$-component of $\mathbf{d}$ for subtree T' is subgaussian with parameter $r \cdot \Phi_{T'}$.

4.4 "Rounded" Function Family $\overline{\mathcal{C}}$

We now define our final "rounded" family of constraining functions, denoted $\overline{\mathcal{C}}$, which we prove to be pseudorandom, as well as (computationally) key-homomorphic and consistent. In $\overline{\mathcal{C}}$ we use the same DAG on $\{0,1\}^{\le |T|}$ as in the noisy function family, but we define somewhat different "rounded" (and errorless) key spaces, and thereby different constraining functions and group operations.

We note that in this scenario, we would only be able to achieve a *computational* version of consistency and homomorphism. That is to say that it is computationally infeasible to find inputs on which our family is not consistent (respectively, homomorphic). We discuss about these properties in more detail in the full version.

Rounding and key spaces. The family $\overline{\mathcal{C}}$ is parameterized by a "rounding factor" $e_{T'}$ for each subtree T' of T. For convenience of analysis, we choose these factors to all divide q, hence they are also powers of two. The factors are defined recursively to satisfy the inequalities

$$e_{T'} \ge \begin{cases} r \cdot \lambda^{\omega(1)} & \text{if } |T'| = 1, \\ (e_{T'.l} \cdot (n\ell) + e_{T'.r}) \cdot \lambda^{\omega(1)} & \text{otherwise.} \end{cases} \tag{10}$$

Note that by inspection of Equations (9) and (10), for all subtrees T' we have

$$e_{T'} \ge r \cdot \Phi_{T'} \cdot \lambda^{\omega(1)} \ .$$

Next, mirroring the definitions from Section 4.2, we define the "rounded" domain $\overline{\mathcal{K}}_{T,j}$ for $0 \leq j < |T|$ as follows

$$\overline{\mathcal{R}}_{T,j} := \begin{cases} \mathbb{Z}_q^n & \text{if } |T| = 1, \\ \overline{\mathcal{R}}_{T.l,j} & \text{if } j \leq |T.l|, \\ \lfloor \mathcal{P} \rceil_{e_{T.l}} \times \overline{\mathcal{R}}_{T.r,j-|T.l|} & \text{otherwise.} \end{cases}$$

As with $\mathcal{R}$, we also define $\overline{\mathcal{R}}_{T,|T|} = \lfloor \mathcal{P} \rceil_{e_T} \times \mathbb{Z}_q^n$. Note that for every subtree T' of T, we have that $\lfloor \mathcal{P} \rceil_{e_{T'}} \subseteq \mathcal{P}$ (because every $e_{T'}$ divides q), we have $\overline{\mathcal{R}}_{T,j} \subseteq \mathcal{R}_{T,j}$. The key space for $w \in \{0,1\}^{\leq |T|}$ and its associated distribution are then defined to be

$$\overline{\mathcal{K}}_{T,w} := \overline{\mathcal{R}}_{T,|w|} \ ,$$
$$\overline{\mathcal{D}}_{T,w} := U(\overline{\mathcal{K}}_{T,w}) \ .$$

Constraining functions. We first define $\mathsf{Round}_{T,j} \colon \mathcal{R}_{T,j} \to \overline{\mathcal{R}}_{T,j}$ for $0 \leq j < |T|$ as follows:

$$\mathsf{Round}_{T,j}(\mathbf{v}) := \begin{cases} \mathbf{v} & \text{if } |T| = 1 \\ \mathsf{Round}_{T.l,j}(\mathbf{v}) & \text{if } 0 < j \leq |T.l| \\ (\lfloor \mathbf{v}_l \rceil_{e_{T.l}}, \mathsf{Round}_{T.r,j-|T.l|}(\mathbf{v}_r)) & \text{otherwise,} \end{cases}$$

where we parse $(\mathbf{v}, \mathbf{s}) = (\mathbf{v}_l, \mathbf{v}_r) \in \mathcal{P} \times \mathcal{R}_{T.r,j-|T.l|}$ in the last case above. For $(\mathbf{v}, \mathbf{s}) \in \mathcal{R}_{T,|T|}$, we simply define $\mathsf{Round}_{T,|T|}(\mathbf{v}, \mathbf{s}) := (\lfloor \mathbf{v} \rceil_{e_T}, \mathbf{s})$.

With this definition in mind, the "rounded" constraining functions $\overline{\mathsf{PConstrain}}_{T,w,x} \colon \mathcal{R}_{T,|w|} \to \overline{\mathcal{K}}_{T,w}$ are simply defined as

$$\overline{\mathsf{PConstrain}}_{T,w,x} := \mathsf{Round}_{T,|wx|} \circ \mathsf{PConstrain}_{T,w,x} \ .$$

Pseudorandomness. We now show that the construction of the family $\overline{\mathcal{C}}$ from Section 4.4 is a constrained PRF, according to Definition 2. Here, we prove selctive security of the function as defined in Definition 2, and use the Security of the Constrain family of functions, as defined in Section 4.2 above. We note that this theorem is very similar to analogous ones proved in prior work [BPR12,BP14], and thus we defer the proof to the full version.

Theorem 2. *The family $\overline{\mathcal{C}}$ described above is pseudorandom for the set of challenge nodes $\{0,1\}^{\leq |T|}$, assuming that the family $\mathcal{C}$ is also pseudorandom over the same set of challenge nodes, where the χ distribution of $\mathcal{C}$ is a subgaussian distribution over $\mathbb{Z}$ with parameter r, where r is the number used to define the rounding factors in Equation* (10).

5 Proxy Re-Encryption with Fine-Grained Access Control

Below we explain the symmetric proxy re-encryption scheme as defined by Boneh *et al.* [BLMR13]. Using this scheme as a starting point, we then construct our scheme which additionally allows for fine-grained access control.

5.1 Symmetric-key Proxy Re-Encryption from Key Homomorphic PRFs

As an application of key homomorphic PRFs Boneh *et al.* [BLMR13] construct a *symmetric-key proxy re-encryption* scheme, a symmetric-key analogue of public-key proxy re-encryption [BBS98,CH07,ABH09,LV08]. A symmetric proxy re-encryption scheme is a symmetric encryption scheme, where given a ciphertext $c = \mathsf{Enc}(k, m)$ of some message m under key k, a proxy can translate this ciphertext to a new ciphertext $c' = \mathsf{Enc}(k', m)$ under a new key given only some re-encryption token Δ. The security definition requires roughly that the token *only* allows to translate ciphertexts in this way, but does not reveal anything about the encrypted message or the involved keys. Given a key-homomorphic PRF $F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, where $(\mathcal{K}, \circ), (\mathcal{Y}, \otimes)$ are groups such that

$$F(k \circ k', x) = F(k, x) \otimes F(k', x)$$

and any symmetric encryption scheme $(\mathsf{enc}\colon \mathcal{Y} \times \mathcal{M} \to \mathcal{C}, \mathsf{dec}\colon \mathcal{Y} \times \mathcal{C} \to \mathcal{M})$ we construct $\Pi_{proxy} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReKeyGen}, \mathsf{ReEnc})$ as follows. $\mathsf{Setup}(1^\lambda)$ outputs public parameters *pp* to be used by F. All algorithms will have *pp* as input, which we will not denote explicitly. The key generation algorithm KeyGen simply outputs a random key $k \leftarrow \mathcal{K}$ for F. Encryption of the proxy re-encryption scheme is defined as $\mathsf{Enc}(k, m) = (r, c_1, c_2)$ where

$$c_1 = \kappa \otimes F(k, r)\ ,\ c_2 = \mathsf{enc}(\kappa, m) \text{ for random } (r, \kappa) \leftarrow \mathcal{X} \times \mathcal{Y} \tag{11}$$

Decryption is $\mathsf{Dec}(r, c_1, c_2) = \mathsf{dec}(c_1 \otimes F(k, r)^{-1}, c_2) = m$. The re-encryption-key generation $\mathsf{ReKeyGen}$ takes two keys k, k' and outputs a re-encryption token

$$\mathsf{ReKeyGen}(k, k') = k^{-1} \circ k'\ .$$

The re-encryption procedure ReEnc takes a re-encryption token $\Delta = \mathsf{ReKeyGen}(k, k')$ and a ciphertext under key k and outputs a ciphertext of the same plaintext under the key k' as

$$\mathsf{ReEnc}(\Delta, (r, c_1, c_2)) = (r, c_1 \otimes F(\Delta, r), c_2)\ .$$

Note $(r, c_1 \otimes F(\Delta, r), c_2) = (r, \kappa \otimes F(k, r) \otimes F(\Delta, r)), c_2) = (r, \kappa \otimes F(k', r), c_2)$ is indeed an encryption of m under key k' as required. We refer the reader to [BLMR13] for a formal definition of symmetric-key proxy re-encryption and the proof of the following

Theorem 3 ([BLMR13]). *If F is a secure key-homomorphic PRF where the input space $\mathcal{X}$ is of superpolynomial size, then Π_{proxy} is a secure proxy re-encryption scheme.*

The superpolynomial domain is required in order for the probability that any two of the randomly chosen $r \in \mathcal{X}$ collide to be negligible.

5.2 Fine-Grained Access Control from Constrained PRFs

Assume the PRF F from the previous section is not only key-homomorphic, but also a constrained PRF. That is, there is a function $\mathsf{Constrain}\colon \mathcal{K} \times \mathcal{P} \to \mathcal{K}_P$ which given a key k and some predicate p outputs a constrained key k_p that allows to evaluate $F(k, \cdot)$ on all inputs x where $p(x) = 1$.

Consider the proxy re-encryption scheme outlined above, but where we slightly change the encryption procedure from Eq. (11), and now instead of choosing r at random during encryption, it is given as part of the input. We call this input x the *attributes* of the ciphertext. I.e., we let $\mathsf{Enc}(k, m, x) = (x, c_1, c_2)$ with

$$c_1 = \kappa \otimes F(k, x)\ ,\ c_2 = \mathsf{enc}(\kappa, m) \text{ for random } \kappa \leftarrow \mathcal{Y}\ .$$

This little change now gives us an extra property: given a constrained key k_p for a predicate p, one can decrypt ciphertexts with attribute x iff $p(x) = 1$. The correctness property of Enc as a proxy re-encryption scheme is not affected by this change.

Informally, the security notion (which we will define formally later) requires that ciphertexts encrypted for some attributes x under key k hide the plaintext as long as it cannot be trivially computed from the outputs of the queries of the adversary (where we allow adversaries to make re-encryption queries and ask for constrained keys).

The security notion of constrained PRFs implies that given keys $k_{p_1}, \ldots, k_{p_\ell}$ for predicates where $p_i(x) = 0$ for all $i = 1, \ldots, \ell$, the output $F(k, x)$ is pseudorandom. It might therefore seem that the key κ is pseudorandom given the encapsulated key $c_1 = \kappa \otimes F(k, x)$. Unfortunately, as discussed in the introduction, this is not true, because in a CPA attack the adversary can not only ask for constrained keys, but also for ciphertexts which reveal function values. We therefore will use a carefully defined probabilistic encoding of attributes such that the functionality of the scheme is preserved, while solving the problems discussed in the introduction.

Randomizable Predicates. How to appropriately define the required encoding is best explained by an example: Consider a bit-fixing CPRF F with inputs from $\{0,1\}^n$. Recall that given a constrained key $k_p \leftarrow \mathsf{Constrain}(k, p)$ for a predicate $p \in \{0, 1, ?\}^n$, we can compute $F(k, x)$ for any attribute x where for every $i \in [n]$ we have $(x[i] = p[i] \vee p[i] = ?)$. For any such predicate p, we denote with p^+ the predicate on $n + \lambda$ bits (where λ is a statistical security parameter) as $p^+(x \| \alpha) = p(x)$, so p^+ simply evaluates p on the first n bits.

In the encryption scheme, the predicate space is still $\{0,1\}^n$, but F is evaluated on inputs of length $n + \lambda$ and a constrained key for $p \in \{0, 1, ?\}^n$ is computed as $k_{p^+} \leftarrow \mathsf{Constrain}(k, p \| ?^\lambda)$. During encryption we now compute the encapsulated key as $c_1 = \kappa \otimes F(k, x \| \alpha)$ for some random α (α is also output as part of the ciphertext). Note that this preserves the proxy re-encryption property: given k_{p^+} one can compute $F(k, x \| \alpha)$ on any (x, α) where $p(x) = 1$. On the other hand, we'll show that the $c_1 = \kappa \otimes F(k, x \| \alpha)$ part of the challenge ciphertext hides the encapsulated key κ because $F(k, x \| \alpha)$ is pseudorandom.

Definition 4. *A randomization of a set of predicates $\mathcal{P}$ is given by an efficient injective encoding $[\cdot, \cdot] \colon \mathcal{P}_{in} \times \{0,1\}^\lambda \to \mathcal{P}_{out}$ ($\mathcal{P}_{in}, \mathcal{P}_{out} \subseteq \mathcal{P}$ and λ being a statistical security parameter) and a mapping $\phi : \mathcal{P}_{in} \to \mathcal{P}_{out}$ (we'll use p^+ as shortcut for $\phi(p)$) such that $p^+([x_1, x_2]) = 1 \iff p(x_1) = 1$. For every $[x, r]$ we require that there exists a $p_{[x,r]} \in \mathcal{P}$ s.t. $p_{[x,r]}([x, r]) = 1$ but $p_{[x,r]}([x', r']) = 0$ for all $(x', r') \neq (x, r) \in \mathcal{P}_{in} \times \{0,1\}^\lambda$.*

For a CPRF for predicates $\mathcal{P}$ that can be randomized, we define $\mathsf{Constrain}^+(k, p) \equiv \mathsf{Constrain}(k, p^+)$. *Note that a key $k_{[x',r']} \leftarrow \mathsf{Constrain}(k, p_{[x,r]})$ allows to evaluate*

$F(k, \cdot)$ only on the value $[x, r]$ in the range of $[\cdot, \cdot]$ (but might allow to evaluate it on other points not in the range of the encoding).[7]

With this definition, the encoding for bit-fixing CPRFs we discussed above can be cast as a randomized predicate with $[x_1, x_2] = x_1 \| x_2$ simply being concatenation and $p^+ = p \| ?^\lambda$ for any $p \in \{0, 1, ?\}^n$.

For prefix CPRF, we let $\tau \colon \{0, 1, 2\} \to \{0, 1\}^2$ be an encoding of a ternary to a binary alphabet (say, $0, 1, 2$ maps to $00, 01, 10$). Then we can use the encoding $[x_1, x_2] = \tau(x_1 \| 2 \| x_2)$ and set $\phi(x) = \tau(x)$ (so $\mathsf{Constrain}^+(k, x) = \mathsf{Constrain}(k, \tau(x))$). [8]

We will prove the following theorem.

Theorem 4. *If F is a secure key-homomorphic constrained PRF, the scheme $\Pi_{\text{fg-proxy}}$ defined in Section 5.4 is a secure proxy re-encryption scheme with fine-grained access control (as defined in Section 5.3).*

5.3 Definition of Proxy Re-Encryption with Fine-Grained Access Control

A *proxy re-encryption scheme with fine-grained access control* for predicates $\mathcal{P}$ over $\mathcal{X}$, where for $p \in \mathcal{P}, x \in \mathcal{X}$ we denote by $p(x) = 1$ that p holds on x, is given by algorithms

$$\Pi_{\text{fg-proxy}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Constrain}, \mathsf{ReKeyGen}, \mathsf{ReEnc}) \ .$$

$\mathsf{Setup}(1^\lambda)$. Setup outputs a set of public parameters pp, which are an implicit input to all other algorithms.

$\mathsf{KeyGen}(1^\lambda)$. Key generation outputs a key $k \in \mathcal{K}$.

$\mathsf{Enc} \colon \mathcal{K} \times \mathcal{X} \times \mathcal{M} \to \mathcal{X} \times \mathcal{C}$. Encryption takes a key k, attributes x and a message m and outputs a ciphertext $(x, c) \leftarrow \mathsf{Enc}(k, x, m)$.

$\mathsf{Constrain}_{\mathsf{ENC}} \colon \mathcal{K} \times \mathcal{P} \to \mathcal{K}_\mathcal{P}$. Constraining takes a key k and a predicate p and outputs a constrained key $k_p \leftarrow \mathsf{Constrain}_{\mathsf{ENC}}(k, p)$ (we use the subscript ENC to avoid confusion with the $\mathsf{Constrain}$ algorithm of the CPRF).

$\mathsf{Dec} \colon (\mathcal{K}_\mathcal{P} \cup \mathcal{K}) \times \mathcal{X} \times \mathcal{C} \to \mathcal{M}$. Decryption takes k and a ciphertext (x, c) and outputs $m \leftarrow \mathsf{Dec}(k, x, c)$; except when $k = k_p \in \mathcal{K}_\mathcal{P}$ and $p(x) = 0$, then it outputs $\perp$.

$\mathsf{ReKeyGen} \colon \mathcal{K} \times \mathcal{K} \to \mathcal{K}$. Re-encryption key-generation takes two keys and outpus a re-encryption key $k_\Delta \leftarrow \mathsf{ReKeyGen}(k, k')$.

[7] Looking forward, this condition will allow us to replace (in the reduction) an output value $F(k, [x, r])$ with a constrained key, while only excluding one possible challenge ciphertext. We observe that without this condition simple concatenation $[x_1, x_2] = x_1 \| x_2$ would already give a randomized predicate for prefix predicates, but this would lead to a trivially insecure encryption scheme $\mathsf{Enc}(k, x, m) = (r, m \otimes F(k, [x, r]))$ if using a GGM based prefix CPRF. In such CPRFs, given some $F(k, x \| r)$ (that an adversary can learn via an encryption query) one can compute $F(k, x \| r \| r')$ for any r'. Using this fact we can break security of the encryption scheme by asking for a challenge for attribute $x' = x \| r$ which we'll be able to decrypt.

[8] The extra symbol 2 in-between the prefix x_1 and the randomness part x_2 is there so the condition from Def. 4 is satisfied. In particular, note that for any $z = [x_1, x_2] = \tau(x_1 \| 2 \| x_2)$, the constrained key $k_z = \mathsf{Constrain}(k, z)$ allows to evaluate $F(k, \cdot)$ only on inputs of the form $z \| w$, but this is in the range of the encoding $[\cdot, \cdot]$ only if w is empty (i.e., only on the unique input z in the range of $[\cdot, \cdot]$). Note that with this encoding the attack from Footnote 7 does no longer work.

$\mathsf{ReEnc}\colon \mathcal{K} \times \mathcal{C} \to \mathcal{C}$. Re-encryption takes a re-encryption key (from k to k') and a ciphertext under k, and outputs a ciphertext of the same plaintext under k'.

Correctness. For any *pp* output by Setup (which is an implicit input to all algorithms) and all k, x, m and p with $p(x) = 1$, let $c \leftarrow \mathsf{Enc}(k, x, m)$. Then we require the following: $\mathsf{Dec}(k, x, c) = m$. For all $k_p \leftarrow \mathsf{Constrain}_{\mathsf{ENC}}(k, p)$: $\mathsf{Dec}(k_p, x, c) = m$. For any k', $k_\Delta \leftarrow \mathsf{ReKeyGen}(k, k')$, $c' \leftarrow \mathsf{ReEnc}(k_\Delta, c)$ we have $\mathsf{Dec}(k', c', x) = m$.

Security. The notion of security for proxy re-encryption with fine-grained access control below is a generalization of the security notion for proxy re-encryption of [BLMR13].

We consider a game between an adversary $\mathcal{A}$ and a challenger. The challenger runs $pp \leftarrow \mathsf{Setup}(1^\lambda)$ (and *pp* is given to $\mathcal{A}$ and to all other algorithms as input), initializes a counter $\mathsf{ctr} := 1$ and samples a random bit $b \in \{0, 1\}$. Then $\mathcal{A}$ can make the following queries.

Uncorrupted Key-Generation: Challenger samples $k^{\mathsf{ctr}} \leftarrow \mathsf{KeyGen}(1^\lambda)$ and increases ctr (the key is not given at $\mathcal{A}$).
Corrupted Key-Generation: Challenger samples $k^{\mathsf{ctr}} \leftarrow \mathsf{KeyGen}(1^\lambda)$ and increases ctr. The key is given to $\mathcal{A}$.
Re-encryption Key-Generation: On input $(i, j), i, j \leq \mathsf{ctr}$ return $\mathsf{ReKeyGen}(k^i, k^j)$ to $\mathcal{A}$. We require that both keys k^i, k^j are uncorrupted.
Constrained Key Request: On input (i, p) return $\mathsf{Constrain}_{\mathsf{ENC}}(k^i, p)$ to $\mathcal{A}$.
Encryption: On input (i, x, m) return $\mathsf{Enc}(k^i, x, m)$ to $\mathcal{A}$.
Re-Encryption: On input (i, j, c) return $\mathsf{ReEnc}(\mathsf{ReKeyGen}(k^i, k^j), c)$ to $\mathcal{A}$. We require that k^j was generated using uncorrupted key generation.
Challenge: This oracle is queried only once in an input (i^*, x^*, m_0^*, m_1^*), we require that k^i was generated using uncorrupted key generation, and for every "Constrained Key Request" query (i, p) where k^i was generated using uncorrupted key generation, we have $p(m_0^*) = p(m_1^*) = 0$ (this also holds for queries to be made after this challenge query).
The challenger returns $\mathsf{Enc}(k^i, x^*, m_b^*)$ to $\mathcal{A}$.
Guess: $\mathcal{A}$ outputs a guess bit b' (the experiment stops at this point).

Definition 5. *$\Pi_{\text{fg-proxy}}$ is a secure proxy re-encryption scheme with fine-grained access control if for all polynomial-time adversaries $\mathcal{A}$, the advantage $|\Pr[b = b'] - 1/2|$ in the above game is negligible in the security parameter λ.*

A Remark on Selective Security. Note that the above notion considers *selective* security in the sense that the adversary must commit whether a key is corrupted or not during its generation (the challenge is chosen adaptively, and for this we'll have to assume adaptive security of the underlying constrained PRF). This will be useful in the security proof, where the reduction will sample corrupted keys itself, and implicitly uses the key of the challenger in the constrained PRF security game to generate uncorrupted keys. We can get selective security via "complexity leveraging", but this loses a huge exponential (in the number m of generated keys) factor in the security reduction[9] as we have to

[9] That is, an attacker with advantage ε against the scheme is turned into an adversary with advantage $\varepsilon/2^m$ against the constrained PRF.

guess initially which keys will be corrupted. When the encryption scheme is actually used to outsource data to an untrusted server, we can assume that re-encryption-key generation queries are not arbitrary, but only applied to consecutive keys, i.e., we only can ask for re-encryption keys $\mathsf{ReKeyGen}(k^i, k^{i+1})$. In this case, adaptive security can be proven losing only a quadratic factor (as for the reduction it will be sufficient to only guess which key will be the first corrupted key before and after the key chosen for the challenge.)

5.4 Construction of Proxy Re-Encryption with Fine-Grained Access Control from Key-Homomorphic Constrained PRFs

We now describe how to construct a scheme $\Pi_{\mathit{fg\text{-}proxy}}$ from a key-homomorphic constrained PRF F for predicates $\mathcal{P}$ that can be randomized (cf. Def. 4) and any symmetric encryption scheme $(\mathsf{enc}, \mathsf{dec})$.

$\mathsf{Setup}(1^\lambda)$ samples and outputs public parameters pp as used by F.
$\mathsf{KeyGen}(1^\lambda)$ outputs a random key $k \in \mathcal{K}$ for F.
$\mathsf{Enc}(k, x, m)$ picks a random $\alpha \in \{0,1\}^\lambda$, a random key κ for enc and sets (with $[\cdot,\cdot]$ as in Def 4)

$$\mathsf{Enc}(k, m, x) = ([x,\alpha], c_1, c_2)\,, \text{ where } c_1 = \kappa \otimes F(k, [x,\alpha]) \text{ and } c_2 = \mathsf{enc}(\kappa, m)$$

$\mathsf{Dec}(k_p, x, c = ([x,\alpha], c_1, c_2))$ checks if $p(x) = 1$. If so, it computes $\kappa = c_1 \otimes F(k_p, [x,\alpha])^{-1}$ and returns $\mathsf{dec}(\kappa, c_2)$.
$\mathsf{Constrain}_{\mathsf{ENC}}(k, p)$ returns $k_p \leftarrow \mathsf{Constrain}^+(k, p)$ (cf. Def. 4)
$\mathsf{ReKeyGen}(k, k')$ returns $k_\Delta = k' \circ k^{-1}$.
$\mathsf{ReEnc}(k_\Delta, c = ([x,\alpha], c_1, c_2)$ returns $c' = ([x,\alpha], F(k_\Delta, [x,\alpha]) \otimes c_1, c_2)$.

Proof of Theorem 4. We now show that the scheme constructed in Section 5.4 satisfies the security notion from Definition 5. We construct an adversary $\mathcal{B}$, who given an adversary $\mathcal{A}$ that breaks the security of the scheme, breaks the security of the underlying constrained PRF with almost the same advantage (we lose an exponentially small additive term due to the possibility of collisions in the randomness used for encryption).

At setup, adversary $\mathcal{B}$ gets the public parameters pp for F, and forwards them to $\mathcal{A}$. Now, $\mathcal{B}$ has access to an oracle $\mathsf{Constrain}(k, \cdot)$ (below $\mathsf{Constrain}^+(k,\cdot)$ is as in Def. 4). $\mathcal{B}$ will answer $\mathcal{A}$'s queries as follows.

Corrupted Key-Generation: $\mathcal{B}$ samples a key $k^{\mathsf{ctr}} \leftarrow \mathsf{KeyGen}(1^\lambda)$, increases ctr and gives the key to $\mathcal{A}$.

Uncorrupted Key-Generation: $\mathcal{B}$ samples a key k_Δ^{ctr} and implicitly sets $k^{\mathsf{ctr}} = k \circ k_\Delta^{\mathsf{ctr}}$ where k is the key used in the $\mathsf{Constrain}(k,\cdot)$ oracle of the security game against the CPRF. Note that k^{ctr} is uniform.

Re-encryption Key-Generation: On input (i,j) where $i, j \leq \mathsf{ctr}$ are uncorrupted keys, $\mathcal{B}$ must return $\mathsf{ReKeyGen}(k^i, k^j)$ to $\mathcal{A}$. It can compute these without knowing k as

$$\mathsf{ReKeyGen}(k^i, k^j) = k \circ k_\Delta^j \circ (k \circ k^i)^{-1} = k_\Delta^j \circ (k_\Delta^i)^{-1}$$

Constrained Key Request: On input (i,p) $\mathcal{B}$ queries its oracle for the key $k_{p+} \leftarrow \mathsf{Constrain}^{+}(k,p)$, then computes $k^{i}_{p+} = k_{p+} \circ \mathsf{Constrain}^{+}(k^{i}_{\Delta},p)$ and returns this key to $\mathcal{A}$.

Encryption: On input (i,m,x) compute $([x,\alpha],c_1,c_2) \leftarrow \mathsf{Enc}(k,m,x)$ as in Sect. 5.4, note that for this we have to learn $F(k,[x,\alpha])$. For this $\mathcal{B}$ queries for the constrained key $k_{[x,\alpha]} \leftarrow \mathsf{Constrain}(k,p_{[x,\alpha]})$ (cf. Def. 4), and then computes $F(k,[x,\alpha])$ using this key.
Return $c = ([x,\alpha],c'_1,c_2)$ to $\mathcal{A}$ where $c'_1 = c_1 \otimes F(k^{i}_{\Delta},[x,\alpha])$ (this step re-encrypts from k to k^i).

Re-Encryption: On input (i,j,c) return $\mathsf{ReEnc}(\mathsf{ReKeyGen}(k^i,k^j),c)$ to $\mathcal{A}$ (note that we already explained how to compute $\mathsf{ReEnc}(\mathsf{ReKeyGen}(k^i,k^j),c)$).

Challenge and Guess: If $\mathcal{A}$ outputs the challenge (i^*,x^*,m^*_0,m^*_1) (where for any predicate p where there was a constrain key-request (i,p) we have $p(x^*)=0$).
$\mathcal{B}$ samples a random α and forwards the challenge $[x^*,\alpha]$ to its CPRF challenger (note that as α is random, with overwhelming probability $\mathcal{B}$ hasn't made the query $\mathsf{Constrain}(k,p_{[x,\alpha]})$ in a previous encryption query, and thus this is a legal challenge. $\mathcal{B}$ gets from his challenger a value γ which is either $F(k,[x^*,\alpha])$ or a uniformly random, depending on whether the challenger's bit b was 0 or 1.
$\mathcal{B}$ samples a random bit β, a random key κ and computes $c = ([x,\alpha],\kappa \otimes \gamma, \mathsf{enc}(\kappa,m^*_{\hat{b}}))$. $\mathcal{B}$ sends c to $\mathcal{A}$, who answers with β'.
If $\beta = \beta'$ $\mathcal{B}$ outputs the guess bit $b' = 0$ (guessing γ is pseudorandom), otherwise $b' = 1$ (guessing γ is uniform).

We analyze the probability that $b = b'$. Conditioned on $b = 0$, c is correctly generated and thus $\mathcal{A}$ has some non-negligible advantage δ in guessing correctly. If $b = 1$, the $c_1 = \kappa \otimes \gamma$ part of the ciphertext is independent of κ, and thus $\mathcal{A}$'s advantage is some negligible $\varepsilon_{\mathsf{enc}}$ (by the security of enc).

$$\Pr[b=b'] - 1/2$$
$$1 \geq \frac{\Pr[b=b'|\beta=0]-1/2}{2} + \frac{\Pr[b=b'|\beta=1]-1/2}{2} \geq \frac{\delta}{2} - \frac{\varepsilon_{\mathsf{enc}}}{2} \ ,$$

which is non-negligible assuming $\varepsilon_{\mathsf{enc}}$ is negligible but δ is not.

References

ABH09. Giuseppe Ateniese, Karyn Benson, and Susan Hohenberger. Key-private proxy re-encryption. In Marc Fischlin, editor, *CT-RSA 2009*, volume 5473 of *LNCS*, pages 279–294. Springer, April 2009.

BBS98. Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 127–144. Springer, May / June 1998.

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

BLMR13. Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, August 2013.

BLP$^+$13. Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, pages 575–584, 2013.

BP14. Abhishek Banerjee and Chris Peikert. New and improved key-homomorphic pseudorandom functions. In *CRYPTO*, pages 353–370, 2014.

BPR12. Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *EUROCRYPT*, pages 719–737, 2012.

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

CH07. Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07*, pages 185–194. ACM Press, October 2007.

GGH13a. Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, May 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 *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

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

KPTZ13. Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 669–684. ACM Press, November 2013.

LV08. Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In Ronald Cramer, editor, *PKC 2008*, volume 4939 of *LNCS*, pages 360–379. Springer, March 2008.

MP12. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, pages 700–718, 2012.

NPR99. Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 327–346. Springer, May 1999.

Pei09. Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, pages 333–342, 2009.

PST14. Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 500–517. Springer, August 2014.

Reg09. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):1–40, 2009. Preliminary version in STOC 2005.

SW14. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.