Direct FSS Constructions for Branching Programs and More from PRGs with Encoded-Output Homomorphism

Elette Boyle [*] Lisa Kohl [†] Zhe Li [‡] Peter Scholl [§]

Abstract

Function secret sharing (FSS) for a class $\mathcal{F}$ allows to split a secret function $f \in \mathcal{F}$ into (succinct) secret shares f_0, f_1, such that for all $x \in \{0,1\}^n$ it holds $f_0(x) - f_1(x) = f(x)$. FSS has numerous applications, including private database queries, nearest neighbour search, private heavy hitters and secure computation in the preprocessing model, where the supported class $\mathcal{F}$ translates to richness in the application. Unfortunately, concretely efficient FSS constructions are only known for very limited function classes.

In this work we introduce the notion of pseudorandom generators with encoded-output homomorphism (EOH-PRGs), and give direct FSS constructions for branching programs and more based on this primitive. Further, we give constructions of FSS for deterministic finite automatas (DFAs) from a KDM secure variant of EOH-PRGs.

- *New abstractions.* Following the work of Alamati et al. (EUROCRYPT '19), who classify minicrypt primitives with algebraic structure and their applications, we capture the essence of our FSS constructions in the notion of EOH-PRG, paving the road towards future efficiency improvements via new instantiations of this primitive. The abstraction of EOH-PRG and its instantiations may be of independent interest, as it is an approximate substitution of an ideal homomorphic PRG.
- *Better efficiency.* We show that EOH-PRGs can be instantiated from LWE and a small-exponent variant of the DCR assumption. A theoretical analysis of our instantiations suggest efficiency improvements over the state of the art both in terms of key size and evaluation time: We show that our FSS instantiations lead to smaller key sizes, improving over previous constructions by a factor of 3.5 and more. For branching programs our FSS constructions show considerably improved run time by avoiding the expensive generic transformation via universal circuits, shaving off a factor of w and more in the number of abstract operations, where w corresponds to an upper bound on the width of the underlying class of branching programs.
- *New feasibility.* We show that our instantiations of EOH-PRGs additionally support a form of KDM-security, without requiring an additional circular-security assumption. Based on this, we give the first FSS construction for DFAs which supports the evaluation of inputs of a-priori unbounded length without relying on FHE.
- *Applications.* We outline applications of our FSS constructions including pattern matching with wild cards, image matching, nearest neighbor search and regular expression matching.

[*]Reichman University, Herzliya, Israel, and NTT Research, USA eboyle@alum.mit.edu
[†]Cryptology Group, CWI Amsterdam, lisa.kohl@cwi.nl
[‡]Cryptology Group, CWI Amsterdam, lizh0048@e.ntu.edu.sg
[§]Aarhus University, peter.scholl@cs.au.dk

Contents

1 Introduction

Boyle, Gilboa and Ishai [BGI15] introduced the notion of *function secret sharing* in 2015. Function secret sharing for a class of functions $\mathcal{F}$ allows to split up a function $f\colon \{0,1\}^n \to \mathbb{G}$ from $\mathcal{F}$ into secret shares f_0, f_1, such that for all $x \in \{0,1\}^n$ it holds $f_0(x) - f_1(x) = f(x)$. If $f\colon \{0,1\}^n \to \mathbb{G}$ is an arbitrary function, its description size can in general scale with 2^n, and thus there is no hope to get compact secret shares. On the other hand, if f is from a class of functions with *succinct* description, one can hope to split the function up into *succinct* secret shares. As shown in [BGI15], when relaxing the secrecy condition to computational (i.e., requiring that no computationally bounded adversary holding only a subset of the shares can derive information about the function within the function class), this can indeed be achieved.

Function secret sharing schemes have been used in numerous applications, such as multi-server private-information retrieval [GI14, BGI15], oblivious RAM [Ds17], anonymous broadcast messaging [CBM15], private database queries [WYG$^+$17], nearest neighbour search [SLD22], private heavy hitters [BBC$^+$21], private time-series database [DRPS22] and secure computation in the preprocessing model [BCGI18, BGI19, BCG$^+$19, BCG$^+$21], showing significant speed-ups over previous approaches. In many of these settings, the class $\mathcal{F}$ supported by the FSS scheme corresponds to richer applications; for example, more sophisticated private database queries beyond private lookup.

Unfortunately, concretely efficient FSS constructions are only known for very limited function classes. For example, efficient function secret sharing schemes are known to exist for the class of point functions (i.e., functions that take a non-zero value only at a single input) and the class of comparison functions (i.e., functions that take the same non-zero value for all inputs less than a given point) [BGI15, BGI16b, BCG$^+$21]. While these are already sufficient for many powerful applications, they do not allow to support, for instance, complex database queries.

One way to obtain function secret sharing for richer classes of function is via *homomorphic secret sharing* (HSS) [BGI16a], the dual notion of function secret sharing, with the role of function and input reversed. HSS schemes for the class of polynomial-size branching programs (which in particular captures logarithmic-depth circuits) are known from a number of assumptions, such as the decisional Diffie-Hellman assumption [BGI16a], the DCR assumption [FGJS17, OSY21, RS21], and the Learning With Errors assumption [DHRW16, BKS19].

As observed in [BGI16a], there exists a generic transformation from a homomorphic secret sharing scheme to a function sharing scheme by relying on universal circuits. A universal circuit for a function class $\mathcal{F}$, is a circuit $C_\mathcal{F}$ such that $\forall f \in \mathcal{F}, \forall x \in \{0,1\}^n$ it holds $C_\mathcal{F}(f, x) = f(x)$. Given such a universal circuit, one can transform the problem of constructing a function secret sharing scheme for $\mathcal{F}$ to the problem of constructing a homomorphic secret sharing scheme for the class of functions $\mathcal{C}_\mathcal{F} := \{C_\mathcal{F}(\cdot, x) \mid x \in \{0,1\}^n\}$.

For the class of branching programs, there exists a universal circuit that is itself a branching program [BGI16a]. Any homomorphic secret sharing scheme for the class of branching programs thus implies a function secret scheme for the same class. Unfortunately, the transformation introduces a high concrete overhead, especially when the structure of the branching program is wished to be hidden. More precisely, with the techniques given in [BGI16a], if w is an upper bound on the width of a binary branching program, then the resulting universal branching program has a blow-up of w^2 in depth, which leads to large key size and running time. For branching programs over larger fields, this overhead gets even worse. In fact, it is an open problem explicitly posed in a talk by Boyle [Boy22, Page 85] to improve the efficiency of FSS over the universal branching program transformation.

We also consider deterministic finite automata (DFAs) in this work [RS59, Sip97]. A DFA is an automaton with finitely many states that rejects or accepts a given string following a sequence of states, where the next state is determined by the next symbol of the string. As observed, e.g., in [IP07], if f is a function of input length n that is computed by a DFA with s states, it can be computed by a branching program of length n and size $s \cdot n + 1$, an FSS for branching programs thus directly yields an FSS for DFAs with bounded input-length. Note though that FSS for branching programs does not allow to compute general classes of DFAs, since these can support inputs of a-priori unbounded length, while yet having a succinct representation.

1.1 Our Contributions

In this work, we present constructions of function secret sharing schemes for the class of bit-fixing predicates, branching programs and more from an abstract pseudorandom generator with encoded-output

		Assumption	Key Size	Run time(No. of Mul./ Exp.)
LWE	HSS [BKS19]	Ring-LWE	$4\ell w^2 n \log q$	$8\ell w^2 n \log n$
	EOH-PRG(Ours)	Ring-LWE	$2\ell w(n+w)\log p$	$\ell(2+\lceil \frac{2w}{n} \rceil) n \log n$
DCR	HSS [OSY21]	DCR	$7\ell w^2 \log N^2$	$14\ell w^2$
	EOH-PRG(Ours)	DCR	$2\ell w(w+1)\log N^2$	$\ell(3w+2)$

Table 1: Comparison of FSS for branching programs constructed from EOH-PRG and from HSS via universal branching programs. ℓ stands for the length of the branching program and w stands for the width of the branching program. Assume fixed out-degree $d = 2$. For the LWE assumption, n stands for the secret length, q the modulus of the LWE assumption, and p the output modulus of the PRG. The number of multiplications is counted over $\mathbb{Z}_q$. For the DCR assumption, N stands for RSA modulus. For the comparison with [OSY21], we use their most efficient instantiation, for which they have to assume a DCR variant with circular security ([OSY21, Section 4.2]). The number of exponentiations is counted over $\mathbb{Z}_{N^2}$.

homomorphism (EOH-PRG). We further show that if the EOH-PRG additionally satisfies a form of KDM-security, we can construct FSS for deterministic finite automata supporting inputs of a-priori *unbounded* length.

We give instantiations of the EOH-PRG from the standard learning with errors (LWE) assumption or a binary-secret variant of ring-LWE, as well as from a small-exponent variant of the decisional composite residuosity (DCR) assumption. We give an overview of the efficiency comparison of our concretely efficient FSS constructions for branching programs to previous FSS constructions via universal branching program transformations in Table 1. In terms of concrete efficiency, the run time is improved by at least a factor of w, where w is the width of the branching program.

In some sense, our work can be viewed as an extension of the line of work on exploring minicrypt primitives with algebraic structure and their applications, as started by Alamati et al. [AMPR19].

Our main results can be captured in a series of theorems. In the following, we will give a simplified definition of our EOH-PRG, which is yet too demanding for our instantiations, but allows to present the essence of our core theorems. For a full definition and more detailed explanation of our results, we refer to the technical overview section.

1.1.1 EOH-PRG.

We start by introducing the concept of an EOH-PRG. Intuitively, an EOH PRG captures the functionality of a homomorphic pseudorandom generator in the following sense: Given an encryption $c = m + \mathsf{PRG}(s)$, a homomorphic PRG would allow to split the decryption key s into two shares $s_0 - s_1 = s$, s.t.,

$$(c_0 - \mathsf{PRG}(s_0)) - (c_1 - \mathsf{PRG}(s_1)) = m,$$

where $c_0 - c_1 = c$. In other words, a homomorphic PRG would allow the distributed decryption of m, where the size of the decryption keys s_0, s_1 are succinct (i.e., scale with the size of s, rather than $\mathsf{PRG}(s)$).

Unfortunately, perfectly homomorphic PRGs with both the domain and image being additive groups in the typical sense are not known to exist; one barrier is that any homomorphic PRG with an output space that supports efficient linear algebra can be broken by Gaussian elimination.

In this paper, we observe that if we relax the above to require the equation only relative to "encoded" messages m, it can be instantiated from standard assumptions.[1] We formalize this requirement in the following definition of pseudorandom generators with encoded output homomorphism. While this definition might look somewhat complex at first glance, we would like to stress that the intuition behind it is very simple: We leverage the observation that if we have some control over the message and PRG seed, one can recover the functionality of a homomorphic PRG while being able to give instantiations from standard assumptions.

Definition 1.1 (EOH-PRG, simplified)**.** *Let $\mathbb{S}, \mathbb{H}, \widetilde{\mathbb{H}}$ be finite abelian groups. A function $\mathsf{PRG}\colon \mathbb{S} \to \widetilde{\mathbb{H}}$ is a PRG with encoded output homomorphism (EOH-PRG) relative to $\mathbb{H}$ if it is a pseudorandom generator and there exists a deterministic polynomial-time encoding function $\mathsf{Encode} : \mathbb{H} \to \widetilde{\mathbb{H}}$ and conversion (or*

[1]Actually, for our instantiations we additionally have to restrict the seed s to be from a special subset $S \subset \mathbb{S}$, and our message from a special subset $H \subset \mathbb{H}$, but for simplicity we start by presenting our results with the slightly simpler definition.

"decoding") function $\mathsf{Conv} : \widetilde{\mathbb{H}} \to \mathbb{H}$ *such that for all* $m \in \mathbb{H}$*, for* $s \in \mathbb{S}$ *it holds*

$$\mathsf{Conv}(c_0 - \mathsf{PRG}(s_0)) - \mathsf{Conv}(c_1 - \mathsf{PRG}(s_1)) = m,$$

where $c_0 - c_1 = \mathsf{PRG}(s) + \mathsf{Encode}(m)$ *and* $s_0, s_1 \in \mathbb{S}$ *with* $s_0 - s_1 = s$ *(except with negligible probability over the random choice of the shares).*

Note that given a truly homomorphic PRG, one could indeed instantiate the above definition of EOH-PRG by setting $\mathbb{H} := \widetilde{\mathbb{H}}$ and choosing Encode and Conv as identity functions.

We will show that the EOH-PRG can be instantiated with different paradigms: It can be instantiated by an *almost* homomorphic PRG, in which Conv corrects introduced errors and transforms shares in $\widetilde{\mathbb{H}}$ back to shares in $\mathbb{H}$ based on learning with errors (similar to the rounding and lifting in [BKS19]), as well as with a homomorphic PRG mapping additive shares to multiplicative shares, in which Conv converts multiplicative shares back to additive shares based on a variant of the DCR assumption (similar to the conversion procedure in [BGI16a, OSY21, RS21]), thereby presenting a way to unify these two approaches to distributed decryption.

For our constructions, we further need the PRG to support a "tag-space" $\mathbb{T}$. We will defer a formal definition to later, but we observe that for our constructions one can simply set $\mathbb{T} = \mathbb{Z}_\tau$, where τ is the order of $\mathbb{H}$ (which will also be satisfied by our instantiations).

1.1.2 Tensor product theorem.

With this EOH-PRG, we can state our main results. We start by giving our tensor product theorem, which can be viewed as lifting the tensor product theorem of [BGI16b] for point predicates (i.e., the family of predicates taking 1 exactly at one point) to arbitrary predicates. Below we present it for the family of bit-predicates, we note though that it readily extends to any predicates with logarithmic-size input space.[2] For more detailed results we refer to the technical overview section and Section 5.

Theorem 1.2 (Tensor product FSS (simplified))**.** *Let* $\ell = \ell(\lambda)$ *be a polynomial. Let* $\mathcal{P}$ *be a family of predicates* $\{0,1\} \to \{0,1\}$*. Let* $\mathbb{S}, \widetilde{\mathbb{H}}, \mathbb{T}$ *be finite abelian groups. Then, if there exists an EOH-PRG* $\mathsf{PRG}\colon \mathbb{S} \to \widetilde{\mathbb{H}}$ *relative to* $\mathbb{H} := (\mathbb{S} \times \mathbb{T})^2$ *with tag space* $\mathbb{T}$*, there exists an FSS for the function class*

$$\mathcal{P}^{\otimes} := \left\{ g_{P_1,\ldots,P_\ell}\colon \{0,1\}^\ell \to \{0,1\}, x \mapsto \bigwedge_{i=1}^{\ell} P_i(x_i) \;\middle|\; \forall i \in [\ell]\colon P_i \in \mathcal{P} \right\}$$

with polynomial key size.

By instantiating the above with the family of bit-fixing predicates, we obtain a FSS construction for bit-fixing predicate. We capture this result in the following corollary.

Corollary 1.3 (FSS for bit-fixing predicates)**.** *Assume all parameters are as in Theorem 1.2 and* $\mathsf{PRG}\colon \mathbb{S} \to \widetilde{\mathbb{H}}$ *is a EOH-PRG relative to* $\mathbb{H} := (\mathbb{S} \times \mathbb{T})^2$ *with tag space* $\mathbb{T}$*. Then, there exists an FSS for* ℓ*-bit bit-fixing predicates with key size* $\log|\mathbb{H}| + (\ell - 1)\log\left|\widetilde{\mathbb{H}}\right|$.

1.1.3 FSS for branching programs.

Next, we state our main theorem for branching programs. We remark that the FSS for branching programs only hides the transition function whereas the topology of the branching program, i.e., the number of nodes of each level, is revealed. It is easy to extend each level to w nodes via adding dummy nodes and then construct an FSS for the extended branching program (note that the same has to be done in order to apply the generic transformation from HSS to FSS, if the topology is wished to be hidden). For more details on the FSS for branching programs, we refer to the technical overview section.

Theorem 1.4 (FSS for branching programs, simplified)**.** *Let* P *be an oblivious, layered branching program with* ℓ *levels, width* w *and out-degree* d*. Let* $\mathbb{S}, \widetilde{\mathbb{H}}, \mathbb{T}$ *be finite abelian groups. Then, if there exists an EOH-PRG* $\mathsf{PRG} : \mathbb{S} \to \widetilde{\mathbb{H}}$ *relative to* $\mathbb{H} := (\mathbb{S} \times \mathbb{T}^w)^d$ *with tag space* $\mathbb{T}$*, there exists an FSS for* P *with key size* $\log|\mathbb{H}| + (\ell - 1) \cdot w \cdot \log\left|\widetilde{\mathbb{H}}\right|$.

With FSS for branching programs, we present an FSS for the class of approximate matching functions Section E.1. We further give an FSS for multivariate polynomials over polynomial size rings in Section E.2.

[2] Note though that this assumes an EOH-PRG with an accordingly larger output space and thus results in larger key sizes.

1.1.4 FSS for DFAs.

Finally, we give our construction of FSS for definite finite automata. Note that the construction of FSS for branching programs would directly imply an FSS for DFA, but requires the input size to be a-priori bounded as the FSS keys scale with the size of the input. Instead, we give a direct construction of a DFA, which can accept inputs of a-priori unbounded size (and for which the key sizes are independent of the size of the input). To that end, we introduce the notion of EOH-PRG with KDM-security. We stress that the kind of KDM-security we require for our FSS construction comes "for free" in our instantiations from LWE and DCR, without needing to assume a circular-security type assumption.

Definition 1.5 (KDM-secure EOH-PRG (simplified)). *Let Ψ be a family of embeddings $\psi\colon \mathbb{S} \to \mathbb{H}$. Let $\mathsf{PRG}\colon \mathbb{S} \to \widetilde{\mathbb{H}}$ be an EOH-PRG relative to $\mathbb{H}$. We say that PRG satisfies KDM-security relative to Ψ, if for each $\psi \in \Psi$, $\mathsf{PRG}^{\psi}(s) := \mathsf{PRG}(s) + \mathsf{Encode}(\psi(s))$ is a secure PRG.*

With this we obtain the following theorem.

Theorem 1.6 (FSS for DFAs (simplified)). *Let M be a DFA with state set Q and alphabet Σ. Let $\mu := |Q \cup \{A, R\}| = |Q| + 2$, where A and R stand for the merged accept state and rejection state, respectively. Let $\mathbb{S}, \widetilde{\mathbb{H}}, \mathbb{T}$ be finite abelian groups. Then, if there exists a EOH-PRG $\mathsf{PRG} : \mathbb{S} \to \widetilde{\mathbb{H}}$ relative to $\mathbb{H} := (\mathbb{S} \times \mathbb{T}^{\mu})^{|\Sigma|+1}$ with tag space $\mathbb{T}$ which satisfies KDM-security relative to a suitable function family Ψ, there exists an FSS for M with key size $|\mathbb{H}| + |Q| \cdot |\widetilde{\mathbb{H}}|$.*

It is worth to mention that the FSS for DFA is the first that allows key size independent of the length of the input(except for the generic constructions from FHE).

1.1.5 Towards Instantiating the EOH-PRG.

In order to instantiate our constructions, we have to allow for a slightly more permissive notion of EOH-PRG, for which it is rather straightforward to adapt the above theorems. Namely, we additionally have to restrict the seed s to be from a special subset $S \subset \mathbb{S}$ (and require that the PRG restricted to S is still a PRG), and the message m from a special subset $H \subset \mathbb{H}$. With this relaxation, we show that it is possible to instantiate the EOH-PRG from LWE and binary-secret ring-LWE building on the techniques of [BKS19], and from a short exponent variant of the DCR assumption inspired by the techniques of [OSY21, RS21]. More precisely, we obtain the following results.

Theorem 1.7 (EOH-PRG from LWE (simplified)). *Let $n, p, q, r, \ell, w \in \mathbb{N}$ such that $r|p, p|q, 1 \ll r \ll p$,*[3] *and $n \log q < m \log p$, where $m := \ell(n + w)$. Further, let $q > 2pB$ and let χ be a B-bounded error distribution.*[4]

Then, assuming learning with errors $\mathsf{LWE}_{n,m,q,\chi}$ is hard, there exists an EOH-PRG $\mathsf{PRG}\colon \mathbb{S} \to \widetilde{\mathbb{H}}$ relative to $(S, H, \mathbb{H})$ with tag space $\mathbb{T}$, where $S = \{0,1\}^n, \mathbb{S} = \mathbb{Z}_p^n, \mathbb{T} = \mathbb{Z}_p, H = (S \times \{0,1\}^w)^{\ell} = \{0,1\}^m$ and $\widetilde{\mathbb{H}} = \mathbb{H} = (\mathbb{S} \times \mathbb{T}^w)^{\ell} = \mathbb{Z}_p^m$.

Recall that the DCR assumption states an N-th residue over $\mathbb{Z}_{N^2}^*$ is computationally indistinguishable from a random element over $\mathbb{Z}_{N^2}^*$. Based on the DCR assumption, Brakerski and Goldwasser [BG10] showed that $(g_1 \dots g_d, g_1^s \dots g_d^s)$ is pseudorandom, where $d \in \mathbb{N}$, each g_i is a N-th residue over $\mathbb{Z}_{N^2}^*$, and s is random element in $\mathbb{Z}_{\phi(N)}$. (Note that this can also be viewed as the DDH assumption over $\mathbb{Z}_{N^2}^*$.)

We have to rely on a variant of this assumption, where the secret is chosen from a (sufficiently large) bounded subspace $[-B/2, B/2] \subset \mathbb{Z}_{\phi(N)}$. Note that similar flavors of small-exponent assumptions have been used in [KK04, ADOS22, BCG+17]. With this, we obtain the following theorem.

Theorem 1.8 (EOH-PRG from DCR (simplified)). *Let B be an integer such that $B \cdot 2^{\lambda} \le N$ and $B > 2^{\lambda}$. Further, let $\ell, w \in \mathbb{N}$ be arbitrary.*

Then, assuming a small exponent variant of DCR holds relative to B, there exists an EOH-PRG $\mathsf{PRG}\colon \mathbb{S} \to \widetilde{\mathbb{H}}$ relative to $(S, H, \mathbb{H})$ with tag space $\mathbb{T}$, where $S = [-B/2, B/2], \mathbb{S} = \mathbb{Z}_{\phi(N^2)}, \mathbb{T} = \mathbb{Z}_{\phi(N^2)}, H = (S \times \{0,1\}^w)^{\ell}, \mathbb{H} = (\mathbb{S} \times \mathbb{T}^w)^{\ell} = (\mathbb{Z}_{\phi(N^2)})^{\ell(1+w)}$ and $\widetilde{\mathbb{H}} = (\mathbb{Z}_{N^2}^*)^{\ell(1+w)}$.

Note that in order for the DCR assumption to hold, the parties cannot know $\phi(N^2)$. In our construction, this will not be an issue. The computation mod $\phi(N^2)$ or $\phi(N)$ in the exponent is automatic because of the structure of the Paillier group, and to sample from $\mathbb{Z}_{\phi(N^2)}$, we can sample from $\mathbb{Z}_{N^2}$

[3] Here, by $\ll$ we denote a super-polynomial gap between parameters.

[4] Note that this requirement on the error distribution is to ensure that LWE implies LWR [BGM+16].

instead, as the two distributions are statistically close. As we will explain in the technical overview, we are able to generate secret shares of elements $x \bmod \phi(N^2)$ whenever $|x|$ is sufficiently small (following the techniques of [OSY21]), and can otherwise perform operations simply over $\mathbb{Z}$.

1.1.6 Comparisons.

We give the concrete comparisons between our FSS for branching programs from EOH-PRGs and the previous FSS constructions via homomorphic secret sharing (HSS) in Table 1. Building on EOH-PRG yields more efficient constructions in terms of key size and runtime. Most notably, the new FSS schemes for branching programs provide significant improvements in run time over FSS from HSS for universal branching programs, by avoiding the overhead of the generic transformation. For example, consider the Multiply-Then-Truncate (MTT) operation [BCG+21], which is central for multiplying numbers in fixed-point arithmetic. With FSS for NC1, the MTT operation can be implemented in one round. The width for an oblivious BP for MTT is lower bounded by $w = N/logN$ [WW05] with N the input number length. For inputs of size N=64 bits as in [BGI19], we thus obtain a lower bound $w = 10$ for the width of the BP. For the DCR-based instantiation we achieve an improvement of roughly a factor > 3.5 in the key size and factor > 40 in the run time and for the Ring-LWE based instantiation we obtain a factor around 20 improvement in the key size and a factor > 250 improvement in the run time. We want to highlight that the run time improvement both for the DCR and the LWE based instantiations scales with w (where w is the width of the BP), and thus is even more significant for wider branching programs. For details on the efficiency comparison we refer to Section H.

1.1.7 Applications.

The central application of FSS schemes are forms of two-server private information retrieval [BGI16a]. Here, it is assumed that two (non-colluding) servers each hold a replication of a database DB with D items, and a client wants to launch a query to the database while keeping the query hidden from both servers individually. Given an FSS scheme supporting the query class, this can be achieved with succinct communication, by having the client split its query into succinct shares, which can then be evaluated by the server. By secrecy of the FSS, the servers do not learn anything about the query, as long as they are non-colluding. In the following, we outline a number of applications, and how our construction can be used towards boosting the applications in terms of expressiveness and/or efficiency.

First of all, our improved FSS constructions for bit-fixing and branching programs yield direct applications to applications such as 2-server private counting queries and private payload computations, as considered, e.g., in [BKS19], with better efficiency. In the following, we further outline applications of our FSS constructions for approximate matching functions, DFAs and more.

Note that only in the private nearest neighbour search(Section G.3), the database privacy is required. We do not pay much attention to server privacy, as we mainly use the applications to show the high compatibability of our FSS schemes.

Private image matching. In private image matching protocols, a client wishes to perform an image matching based on some similarity metric without the revealing the query image. Private image matching protocols have many crucial applications, including patient CT image retrieval, logo patent search and face detection. Most existing secure privacy-preserving image matching protocols [JB22] rely on searchable encryption [SWP00] or homomorphic encryption [Pai99] to perform computation on encrypted images, which incur large computation cost. Recently a private approximate membership computation protocol with perceptual hash matching was proposed in [KM21], which can also be used to perform image matching. However, the approximate membership computation protocol heavily relies on fully homomorphic encryption. Our FSS allows to support approximate matching queries in the Hamming metric, without the need for expensive ciphertext multiplication. Previous to our work, the only way to achieve FSS for approximate matching required the expensive transformation of HSS for branching programs via universal circuits, we thus expect significant efficiency improvements using our FSS constructions for 2-server private image matching.

Private nearest neighbour search. In private nearest neighbour search, a client wants to find the nearest neighbour to the query feature in a database of feature vectors based on some metric, e.g., the Euclidean, Hamming, or ℓ_1 metric, without revealing information on the query feature vector. In this setting, typically also database privacy is required, i.e., that the client learn nothing on the database beyond the query answer.

Most of the existing protocols for private nearest neighbor search rely on 2-party secure computation or fully homomorphic encryptions. For instance, the protocol SANNS [CCD+20] uses oblivious RAM, garbled circuits and homomorphic encryptions. In the recent protocol [SLD22], distributed point functions and locality sensitive hashing are used to achieve a 2-server private nearest neighbor search protocol. Here, the local sensitive hashing leads to some accuracy loss. To achieve a good accuracy, say $> 95\%$, $O(\sqrt{D})$ queries to the database are necessary, which leads to an extra $O(\sqrt{D})$ factor to the communication cost.

With our FSS construction, we can obtain direct construction of 2-server private nearest neighbor search without relying on locality sensitive hashing, removing this extra factor of $O(\sqrt{D})$, albeit at the cost of an increased run time.

Private partial text matching. In private partial text matching protocol, a client wants to run a fuzzy pattern matching without leaking the query pattern. The like operator in SQL is a typical case of such a partial text matching. This can be modeled as a DFA, and can thus be instantiated with our FSS for DFA (although this potentially requires the addition of some dummy states, in order to hide the number of states of the underlying DFA). Our FSS for DFA is the first that allows communication complexity independent of the length of the inputted text (except for generic instantiations relying on fully homomorphic encryption). Other interesting applications of FSS for DFA include private searching on DNA sequences or pharmaceutical databases and malware detection.

Extension of Splinter via more expressive FSS for interval functions. The FSS construction supporting constant dimension intervals presented in [BCG+21] has many practical applications such as to Splinter [WYG+17], which provides a platform for private searching queries, e.g., for restaurant reviews on Yelp. However, the key size of the construction of [BCG+21] scales with n^d, where n is the length of the PRG seed and d the number of dimensions, and is thus inherently bounded to constant. Our tensor operation technique for FSS from EOH-PRGs, on the other hand, works for an arbitrary polynomial number of dimensions, with key size scaling only linearly with the number of intervals, and has thus direct applications to more expressive queries in Splinter. We leave it as an interesting open direction to explore other applications of EOH-PRG, such as efficient 1-round secure evaluation of multiply-then-truncate [BCG+21] .

1.2 Discussion and Related Work

1.2.1 Beyond the two-party case.

Note that the FSS constructions from one-way functions [GI14, BGI15, BGI16b] cannot be easily extended to more than two parties. Our FSS construction approach from EOH-PRGs, on the other hand, naturally extends beyond the two-party setting. However, it is not known how to instantiate the EOH-PRG from concrete assumptions for more than two parties. Our two-party instantiations from LWE and the DCR variant heavily rely on the distributed rounding [BKS19] and distributed discrete logarithm [OSY21], respectively, which were developed for two-party homomorphic secret sharing. To date, it is unclear how to generalize the distributed rounding or distributed discrete logarithm to more than two parties. In fact, [BDIR18] proved that there exists a barrier to directly generalize the share conversion from two-party to multi-party. Any such progress may lead to significant improvements for efficient multi-party FSS/HSS constructions.

1.2.2 On FSS from weaker assumptions.

While constructing FSS for function classes such as branching programs solely based on the assumption of one-way functions would be a major breakthrough [BGI15], it seems a more tractable open question if such FSS can be constructed for subclasses of $\mathtt{AC0}$ such as bit-fixing predicates or t-CNF. In the technical overview, we give some intuition why it seems unlikely that the techniques of the line of work on FSS from one-way functions [GI14, BGI15, BGI16b] allow for this without relying on additional structure (such as EOH-PRGs), due to an inherent exponential blow-up. An alternative route could be taken following [DKN+20], who give constructions of privately constrained PRFs for t-CNFs from one-way functions. Here, however, the problem is that de-randomizing the constrained points to fixed values would again introduce an exponential blow-up. We leave it as an interesting open questions to either give such candidates, or give barriers towards their construction.

1.2.3 Prior work.

As mentioned above, our constructions are inspired by the tensor production construction for point functions of [BGI16b] based on one-way functions, which generalizes the previous constructions of DPFs in [GI99] and in [BGI15]. Despite recent advances such as [BCG$^+$21], this line of work yielding very efficient FSS from one-way functions is restricted to very simple function classes, such as FSS for point or comparison functions.

Our LWE-based instantiations of EOH-PRGs build on a series of works of PRGs and HSS from lattices. Namely, the underlying PRG builds on the LWR which was introduced in [BPR12] and shown to imply almost homomorphic PRGs and PRFs in [BLMR13]. The "distributed rounding operation" we rely on towards instantiating our EOH-PRG was first used in [DHRW16] to support spooky relationships for encryptions and used again in [BKS19] to construct homomorphic secret sharing. We further also build on the "lifting operation", which was proposed in [BKS19] to avoid a hierarchy of decreasing moduli.

Our DCR-based instantiations of EOH-PRGs follow the line of work on group-based HSS. Boyle et al. [BGI16a] were the first to propose the framework for HSS in group model based on DDH. Following the framework of [BGI16a], [FGJS17] constructed a HSS based on the DCR assumption. However, they only achieved polynomial correctness error and polynomial size plaintext space as in [BGI16a]. This was later improved in [OSY21, RS21], both of which achieved negligible correctness error and super-polynomial size plaintext space. Our work follows the techniques introduced by [OSY21].

1.2.4 Relation to secure branching program evaluation protocols.

There is a line of work on secure branching program evaluation (BPE) [BPSW07, BFK$^+$09, BFL$^+$11, BPTG15, WFNL16, KNL$^+$19, TKK19] relying on garbled circuits or homormorphic encryption. The setting considered in their work is somewhat orthogonal to ours: They consider a branching program (held by a sender) to be evaluated on a single input (held by a receiver), such that the result is learned by the receiver, and such that both the branching program provided by the sender and receiver input remain hidden. We, on the other hand, consider a branching program (held by a client) to be evaluated on a database (held by two servers), such that a linear combination of the outputs is learned by the client, and such that the branching program (i.e., database query) provided by the client remains hidden, as long as the two servers are not colluding. With our approach, the communication cost scales with $\log N$ for a database of size N, since the same branching program can be evaluated on *all* inputs. Except for the FHE-based approach [BPTG15], the communication cost of all other protocols in the BPE line of work instead scales with N to achieve the same functionality. This is even true for the protocols [WFNL16] relying on additively homomorphic encryption, since they still require communication between the receiver and sender *per input* to be evaluated. It is worth to point out that sublinear communication complexity in the line of work on BPE (as achieved in [TKK19]) refers to sublinear in the *size of the branching program*, whereas we consider settings where the size of the database N is the dominating cost.

1.3 Organization

Only the main results and techniques are presented in the body part. Section 2 presents an overview of the central techniques, followed by preliminaries in Section 3. The EOH-PRG is formally defined in Section 4. We show the constructions for tensor product, branching programs and DFAs in Section 5,6,7, respectively. Finally, in Section 8 we present instantiations of the EOH-PRG.

2 Technical Overview

In the following we give an overview of the central techniques. We start by explaining the *tensor product FSS for point functions* of Boyle, Gilboa and Ishai [BGI16b] (in the following refered to as BGI16), and show how to extend their construction to a more general tensor product using an encoded-output PRG. Then, we show how this yields FSS for the classes of bit-fixing predicates. Next, we explain how the construction can be extended towards FSS for branching programs and for DFAs.

2.1 Existing Constructions and Limitations

2.1.1 Background [BGI15, BGI16b]

Before giving the construction, we recall some required preliminaries. Firstly, recall that a *point function* is simply a function that takes a non-zero value only at one dedicated point. More precisely, the point function f_α^β with input space $\{0,1\}^n$ and output space $\mathcal{R}$ (for some group $\mathcal{R}$) is defined as

$$f_\alpha^\beta(x) := \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{else} \end{cases}.$$

A *function secret sharing scheme* for a family of function $\mathcal{F}$ consists of tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{Eval})$, such that Gen takes as input the description $\hat{f}$ of f and returns a tuple of keys (k_0, k_1) and Eval takes as input a party index b, a key k_b and an input value x and outputs an output value y_b, such that the following holds:

Correctness: For all $x \in \{0,1\}^n$. it holds $\mathsf{Eval}(0, k_0, x) - \mathsf{Eval}(1, k_1, x) = f(x)$.

Secrecy: For $b \in \{0,1\}$, k_b computationally hides $\hat{f}$ within $\mathcal{F}$.

Note that an FSS for the class of point functions, is also refered to as *distributed point function (DPF)*.

2.1.2 Tensor product FSS for point functions [BGI16 [BGI16b]]

Given a function secret sharing scheme for the class $\mathcal{F}^\circ$ of *point functions*, and a function secret sharing scheme for a function class $\mathcal{F}$ of *arbitrary functions*, BGI16 gives a construction for the *tensor product* $\mathcal{F}^\circ \otimes \mathcal{F}$, i.e., the class of functions

$$F_{\alpha,f}(x_1, x_2) := \begin{cases} f(x_2) & \text{if } x_1 = \alpha \\ 0 & \text{else} \end{cases}$$

for $\alpha \in \{0,1\}^n, f \in \mathcal{F}$, where the key size scales polynomially in the key sizes of the underlying FSS schemes.[5]

In the following, we describe the construction of BGI16 in a number of steps, adding layers of secrecy one-by-one. For the construction we assume that the FSS scheme $(\mathsf{Gen}, \mathsf{Eval})$ for $\mathcal{F}$ satisfies a *symmetry* property, i.e., $\mathsf{Eval}(0, k, x) = \mathsf{Eval}(1, k, x)$ for all inputs x and keys k. Further, we assume that keys $(k_0, k_1) \leftarrow \mathsf{Gen}(f)$ are individually pseudorandom over the same key space $\mathcal{K}$.

First attempt: a construction with very limited secrecy: To get a construction where α is hidden from P_1, one can proceed as follows: To share $F_{\alpha,f}$, one can generate keys $(k_0, k_1) \leftarrow \mathsf{Gen}(\hat{f})$ and set $K_0 := (\alpha, k_0, k_1)$ and $K_1 := k_1$. To evaluate on point (x_1, x_2), party P_0 outputs $y_0 := \mathsf{Eval}(0, k_0, x_2)$ if $x_1 = \alpha$ and $y_0 := \mathsf{Eval}(0, k_1, x_2)$ otherwise. Party P_1 simply outputs $y_1 := \mathsf{Eval}(1, k_1, x_2)$.

Correctness and very limited secrecy: Here, for $x_1 = \alpha$ we have $y_0 - y_1 = \mathsf{Eval}(0, k_0, x_2) - \mathsf{Eval}(1, k_1, x_2) = f(x_2)$ by the correctness of $(\mathsf{Gen}, \mathsf{Eval})$. For $x_1 \neq \alpha$, on the other hand, it holds $y_0 - y_1 = \mathsf{Eval}(0, k_1, x_2) - \mathsf{Eval}(1, k_1, x_2) = 0$ by symmetry, as required.[6] This construction does obviously hide α from P_1, but otherwise does not provide any secrecy guarantees.

Second attempt: a construction with secret α. Towards hiding α also from P_0, the trick is to additionally use the FSS scheme $(\mathsf{Gen}^\circ, \mathsf{Eval}^\circ)$ for $\mathcal{F}^\circ$, and flipping the order of k_0 and k_1 with probability $1/2$, thereby hide from the parties when they use the same keys. More precisely, assume to be given an FSS for point functions with output space $\{0,1\}$ (i.e., the point function maps to 1 at the unique non-zero point α, and otherwise to 0). Now, the idea is to generate keys $(k_0^\circ, k_1^\circ) \leftarrow \mathsf{Gen}^\circ(\hat{f}_\alpha^1)$, and compute "tag" values $\tau_b \leftarrow \mathsf{Eval}^\circ(b, k_b^\circ, \alpha)$ (i.e., $\tau_0 \oplus \tau_1 = 1$ by construction) relative to these keys. The tag values are used to hide if the parties use the same key k_b, by defining $\mathsf{cw}_{\tau_b} := k_b$ (where $(k_0, k_1) \leftarrow \mathsf{Gen}(\hat{f})$ as before) and setting $K_b := (k_b^\circ, \mathsf{cw}_0, \mathsf{cw}_1)$. To evaluate at a point (x_1, x_2), party P_b first computes $t_b \leftarrow \mathsf{Eval}^\circ(b, k_b^\circ, x_1)$ and then outputs $y_b \leftarrow \mathsf{Eval}(b, \mathsf{cw}_{t_b}, x_2)$.

[5]The resulting scheme actually satisfies a stronger notion of key compactness, namely the non-public part of the key does not grow, allowing to apply the tensor product operation recursively a polynomial number of times.

[6]This construction would not actually require symmetry of the underlying FSS since P_0 knows when $x_1 \neq \alpha$ and could evaluate $\mathsf{Eval}(1, k_1, x_2)$in this case, but this will no longer be possible in the subsequent constructions.

Correctness and secrecy of α: Now, for $x_1 = \alpha$, it holds $\mathsf{cw}_{t_b} = \mathsf{cw}_{\tau_b} = k_b$. As before, the parties thus obtain $y_0 - y_1 = \mathsf{Eval}(0, k_0, x_2) - \mathsf{Eval}(1, k_1, x_2) = f(x_2)$ as required. If $x_1 \neq \alpha$, on the other hand, it holds $t_0 = t_1$ and thus $k_{t_0} = k_{t_1}$, implying $y_0 - y_1 = \mathsf{Eval}(0, k_{t_0}, x_2) - \mathsf{Eval}(1, k_{t_0}, x_2) = 0$, again by symmetry. The construction hides α from both parties by the secrecy of $(\mathsf{Gen}^\circ, \mathsf{Eval}^\circ)$ and the pseudorandomness of keys for $(\mathsf{Gen}, \mathsf{Eval})$ (which prevents the parties from learning when they use the real key at position α and when they use a "dummy key"), but still fully leaks f.

The construction of BGI16. The idea of BGI16 to overcome this, is to additionally use a pseudorandom generator to *blind* the keys k_0, k_1, such that party P_0 is only able to recover k_0 and party P_1 is only able to recover k_1 at the dedicated point $x_1 = \alpha$ (without being able to distinguish this from the case where both parties recover the same "dummy" key, to ensure that α remains hidden). To this end, assume that FSS° is now an FSS for point functions with output space $\{0,1\}^{\lambda+1}$. The idea of BGI16 is as follows: To generate a key for $F_{\alpha,f}$, the key generation algorithm starts by choosing $s \leftarrow_R \{0,1\}^\lambda$ at random and generating $(k_0^\circ, k_1^\circ) \leftarrow \mathsf{Gen}^\circ(\hat{f}_\alpha^{s,1})$. Further, the key generation algorithm generates the corresponding "seed values" $\sigma_b \in \{0,1\}^\lambda$ and, again, "tag values" $\tau_b \in \{0,1\}$ as $(\sigma_b, \tau_b) \leftarrow \mathsf{Eval}^\circ(b, k_b^\circ, \alpha)$ (i.e., $\sigma_0 \oplus \sigma_1 = s$ and $\tau_0 \oplus \tau_1 = 1$ by construction). Further, given a pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \to \mathcal{K}$, the full "correction words" are generated as $CW_{\tau_b} := k_b + \mathsf{PRG}(\sigma_b)$ (where $(k_0, k_1) \leftarrow \mathsf{Gen}(\hat{f})$ as before) and the keys defined as $K_b := (k_0^\circ, CW_0, CW_1)$. To evaluate on point (x_1, x_2), the parties now compute $(s_b, t_b) \leftarrow \mathsf{Eval}^\circ(b, k_b^\circ, x_1)$, "correct" their keys to $\kappa_b := CW_{t_b} - \mathsf{PRG}(s_b)$ and evaluate to $y_b \leftarrow \mathsf{Eval}(b, \kappa_b, x_2)$.

Correctness and secrecy of BGI16: If $x_1 = \alpha$, it holds $\kappa_b = CW_{t_b} - \mathsf{PRG}(s_b) = CW_{\tau_b} - \mathsf{PRG}(\sigma_b) = k_b$ and thus $y_0 - y_1 = \mathsf{Eval}(0, k_0, x_2) - \mathsf{Eval}(1, k_1, x_2) = f(x_2)$ as required. If $x_1 \neq \alpha$, on the other hand, then $t_0 = t_1$ and thus $\kappa_0 = \kappa_1$ (i.e., both parties recover the same "dummy" key), and $y_0 - y_1 = \mathsf{Eval}(0, \kappa_0, x_2) - \mathsf{Eval}(1, \kappa_0, x_2) = 0$ by symmetry. Full secrecy holds by the above considerations and because $(\mathsf{Gen}, \mathsf{Eval})$ satisfies secrecy and pseudorandomness of keys, which prevents the parties from learning where the true FSS keys are embedded.

2.1.3 Limitation of BGI16 to tensoring with point functions

The issue with extending the above approach even slightly beyond point functions (e.g., to function which take a non-zero value at two points) is that it would incur an exponential blow-up in the key size (and run time of the key generation), since the parties have to recover *different* key pairs (K_0, K_1) and (K_0', K_1') for different non-zero points α, α' (as reusing a key would allow the parties to locally derive information about the position of non-zero points). Note that this includes the "correction word" part CW_0, CW_1 of the key, since keys with different first component require different correction words in construction of BGI16. The key generation time and key length thus (at least) *double* at each tensoring. Recursive tensoring is therefore limited to at most a logarithmic number of times, which is not sufficient for most applications.

This issue could be overcome, if the keys could be "randomized" in order to hide that the *same* key is reused. For additive secret sharing this is trivially the case: Namely, assume an output value is shared as $y = k_0 - k_1 \in \mathcal{K}$ (for some additive group $\mathcal{K}$). Then, for any $\Delta \in \mathcal{K}$, the secret can be re-shared as $(k_0 + \Delta, k_1 + \Delta)$, which looks like perfectly fresh keys from the view of the adversary. Unfortunately, the construction of BGI16 does not satisfy this property of "shift-invariance", even if the underlying FSS schemes $\mathcal{F}^\circ$ and $\mathcal{F}$ were to satisfy these properties: Namely, even if $\sigma_0 - \sigma_1 = \sigma_0' - \sigma_1'$ (where $\sigma_b \leftarrow \mathsf{Eval}^\circ(b, k_b, \alpha)$ and $\sigma_b' \leftarrow \mathsf{Eval}^\circ(b, k_b, \alpha')$), the PRG outputs $\mathsf{PRG}(\sigma_b)$ and $\mathsf{PRG}(\sigma_b')$ are in general uncorrelated.

This could be resolved by using an ideal *homomorphic* PRG, ensuring that the correlation is preserved to $\mathsf{PRG}(\sigma_0) - \mathsf{PRG}(\sigma_1) = \mathsf{PRG}(\sigma_0') - \mathsf{PRG}(\sigma_1')$. Unfortunately, perfectly homomorphic PRGs with both the domain and image being additive groups in the typical sense are not known to exist. For simplicity, we still start by outlining our tensor product construction assuming access to a perfectly homomorphic PRG $\mathsf{PRG} \colon \{0,1\}^\lambda \to \mathcal{K}$, before giving our full construction.

2.1.4 Overcoming the limitations via an ideal homomorphic PRG

We start by simplifying the construction of BGI16 assuming access to a perfectly homomorphic PRG $\mathsf{PRG} \colon \{0,1\}^\lambda \to \mathcal{K}$, and assuming a *shift-invariant* FSS $\mathsf{FSS} = (\mathsf{Gen}, \mathsf{Eval})$ for $\mathcal{F}$ with key space $\mathcal{K}$,[7] i.e.,

[7]Note, that for correctness of the simplified construction outlined below, we would actually require $(\mathcal{K}, +) := (\{0,1\}^k, \oplus)$, for some $k \in \mathcal{K}$. To be aligned with the general construction, we will still use th notation $(\mathcal{K}, +)$ in the following.

for $(k_0, k_1) \leftarrow \mathsf{Gen}(\hat{f})$, we assume that any shifted tuple $(k_0 + \Delta, k_1 + \Delta)$ for $\Delta \in \mathcal{K}$ constitutes a valid key pair for f. Then, the construction of BGI16 can be simplified to a construction requiring only one correction word CW:

Again, to generate a key for $F_{\alpha,f}$, the key generation algorithm samples $s \leftarrow_R \{0,1\}^\lambda$ and generates keys $(k_0^\circ, k_1^\circ) \leftarrow \mathsf{Gen}^\circ(f_\alpha^{s,1})$. Instead of pre-computing the tag and seed values at position α, the key generation algorithm simply generates $(k_0, k_1) \leftarrow \mathsf{Gen}(\hat{f})$, sets $CW := k_0 - k_1 + \mathsf{PRG}(s)$ and outputs (K_0, K_1), where $K_b := (k_b^\circ, CW)$. To evaluate, the parties now compute $(s_b, t_b) \leftarrow \mathsf{Eval}^\circ(b, k_b^\circ, x_1)$ and then obtain the "corrected" keys as $\kappa_b := t_b \cdot CW - \mathsf{PRG}(s_b)$. (Note that $t_b \in \{0,1\}$, and thus the multiplication simply corresponds to adding 0 or $\mathcal{CW}$.)

Correctness holds, since at position α it holds $s_0 - s_1 = s$ and $t_0 - t_1 = 1$, and thus

$$\kappa_0 - \kappa_1 = (t_0 \cdot CW - \mathsf{PRG}(s_0)) - (t_1 \cdot CW - \mathsf{PRG}(s_1)) = CW - \mathsf{PRG}(s) = k_0 - k_1.[8]$$

As FSS is shift-invariant, the above implies that (κ_0, κ_1) and (k_0, k_1) are functionally equivalent, and thus $y_0 - y_1 = \mathsf{Eval}(0, \kappa_0, x_2) - \mathsf{Eval}(1, \kappa_1, x_2) = f(x_2)$ as required. If $x_1 \neq \alpha$, on the other hand, we obtain $s_0 = s_1$ and $t_0 = t_1$ and thus $\kappa_0 = \kappa_1$ as before, and correctness follows from the symmetry of FSS. Secrecy holds by the secrecy of the underlying FSS schemes, together with the pseudorandomness of PRG.

Note that this simplified scheme readily extends beyond point functions.

2.2 Our Constructions

2.2.1 EOH-PRG

In order to instantiate the above construction, we introduce the notion of *PRG with encoded-output homomorphism* (EOH-PRG) and show that the above construction can be extended to support instantiation from this weaker notion. Roughly, an EOH-PRG has "encoding" and "conversion" (or "decoding") functions Encode and Conv such that it satisfies the following: given additive secret shares (s_0, s_1) of a seed s, and additive secret shares (y_0, y_1) of a blinded encoding $\mathsf{PRG}(s) + \mathsf{Encode}(m)$, we require $\mathsf{Conv}(y_0 - \mathsf{PRG}(s_0)) - \mathsf{Conv}(y_1 - \mathsf{PRG}(s_1)) = m$ (except with negligible probability over the random choice of the secret shares). Intuitively, this is sufficient to instantiate (a variant of) the tensor product FSS above, by encoding the key difference $k_0 - k_1$ to $\mathsf{Encode}(k_0 - k_1)$ before adding $\mathsf{PRG}(s)$.

More formally, a EOH-PRG $\mathsf{PRG}\colon \mathbb{S} \to \widetilde{\mathbb{H}}$ as required for our tensor product construction is parametrized by $S, H, \mathbb{H}$, together with (efficiently computable) maps $\mathsf{Encode}\colon H \to \widetilde{\mathbb{H}}$ and $\mathsf{Conv}\colon \widetilde{\mathbb{H}} \to \mathbb{H}$ such that:

- $\mathsf{PRG}\colon \mathbb{S} \to \widetilde{\mathbb{H}}$ is a PRG.
- $S \subset \mathbb{S}$ is such that PRG restricted to S is still a PRG and $0 \in S$. Note that S will serve as the seed space of our tensor product (recall that before $S = \{0,1\}^\lambda$). 0 has to be included in S to account for the case where both parties recover the same "dummy" seed value $s_0 = s_1$, i.e., $s_0 - s_1 = 0$.
- $\mathbb{H}$ is a finite abelian group containing the set $H \subset \mathbb{H}$. Note that $\mathbb{H}$ will correspond to the key space $\mathcal{K}$ of the FSS FSS before encoding (i.e., recovery of the keys is relative to addition in $\mathcal{K}$). $H \subset \mathbb{H}$ will correspond to the set of actual key differences $k_0 - k_1$ for $(k_0, k_1) \leftarrow \mathsf{Gen}(\hat{f})$ for $f \in \mathcal{F}$. Having separate $H \subset \mathbb{H}$ stems from the intantiations of EOH-PRG – given a truly homomorphic PRG, one could simply choose $H = \mathbb{H} = \widetilde{\mathbb{H}}$.
- $\widetilde{\mathbb{H}}$ is a finite abelian group containing the image of the PRG.

Finally, $\mathsf{Enc}\colon H \to \widetilde{\mathbb{H}}$ and $\mathsf{Conv}\colon \widetilde{\mathbb{H}} \to \mathbb{H}$ are such that for all $s \in S$, for all $m \in H$, for random secret shares $s_0, s_1 \leftarrow_R \mathbb{S}$ with $s_0 - s_1 = s$, and for random secret shares $y_0, y_1 \leftarrow_R \widetilde{\mathbb{H}}$ with $y_0 - y_1 = \mathsf{PRG}(s) + \mathsf{Encode}(m)$ it holds

$$\mathsf{Conv}(y_0 - \mathsf{PRG}(s_0)) - \mathsf{Conv}(y_1 - \mathsf{PRG}(s_1)) = m$$

in $\mathbb{H}$ except with negligible probability.

Further, we require a tag space which operates on $\widetilde{\mathbb{H}}$. More formally, we require the following:

- $\mathbb{T}$ is a finite abelian group containing $\{0,1\} \subset \mathbb{T}$. Note that $T = \{0,1\}$ will serve as the tag space of our tensor product FSS, which is embedded in the group $\mathbb{T}$, i.e., recovery of the tag will now be additive over $\mathbb{T}$, rather then additive over $(\{0,1\}, \oplus)$.

[8]Note that to obtain $t_0 \cdot CW - t_1 \cdot CW = CW$ we use $(\mathcal{K}, +) = (\{0,1\}^k, \oplus)$. We will later show how to generalize this.

- $\cdot\colon \mathbb{T} \times \tilde{\mathbb{H}} \to \tilde{\mathbb{H}}$ is an efficiently computable non-trivial (left) homomorphic group operation of $\mathbb{T}$ on $\tilde{\mathbb{H}}$, i.e., $\forall t_0, t_1 \in \mathbb{T}$ and $\forall h \in \tilde{\mathbb{H}}$, $t_0 \cdot h + t_1 \cdot h = (t_0 + t_1) \cdot h$. Note that we need to define an operation of $\mathbb{T}$ on $\mathbb{H}$ in order to "correct" the keys based on the tag values. This allows to support more general key spaces than $\mathcal{K} = \{0,1\}^k$.

As mentioned above, In the following, we will assume this as part of the definition of an EOH-PRG and simply add $\mathbb{T}$ to the parametrization.

2.2.2 A first step: tensor product FSS for arbitrary predicates from EOH-PRG

Our tensor product construction from a EOH-PRG is essentially the same as the one from a perfectly homomorphic PRG (assuming the underlying FSS satisfy some additional properties), except that the correction word is computed as $CW := \mathsf{PRG}(s) + \mathsf{Enc}(k_0 - k_1)$, and the keys are recovered as $\kappa_b := \mathsf{Conv}(t_b \cdot CW - \mathsf{PRG}(s_b))$.[9]

Note that for construction to satisfy correctness, it is now required that the FSS scheme FSS for $\mathcal{F}$ satisfies $k_0 - k_1 \in H$ for all $f \in \mathcal{F}$ and $(k_0, k_1) \in \mathsf{Gen}(\hat{f})$ (since the encoding function Encode takes inputs in H), but satisfies shift invariance relative to the additive group $\mathbb{H}$ (since the decoding function Conv returns elements in $\mathbb{H}$).

An example for such an FSS can be obtained by additively secret-sharing the truth table with values in H over $\mathbb{H}$ if the function class is sufficiently small, this can be done efficiently.

With this we obtain the following theorem.

Theorem 2.1 (Theorem 5.1)**.** *Assume* $\mathsf{PRG} : \mathbb{S} \to \tilde{\mathbb{H}}$ *is a EOH-PRG parametrized by* $(S, \mathbb{T}, H, \mathbb{H})$. *Further, let* $\mathsf{FSS}^{\mathcal{P}} = (\mathsf{Gen}^{\mathcal{P}}, \mathsf{Eval}^{\mathcal{P}})$ *be an FSS for a function family* $f_P^\beta\colon \{0,1\}^{n_1} \to \mathbb{S} \times \mathbb{T}$*, where*

$$f_P^\beta(x_1) = \begin{cases} \beta & \textit{if } P(x_1) = 1 \\ 0 & \textit{else} \end{cases},$$

for $\beta \in S \times \{1\} \subseteq S \times T \subseteq \mathbb{S} \times \mathbb{T}$ *(i.e., recovery is additive in* $\mathbb{S} \times \mathbb{T}$*) and* $P \in \mathcal{P}$*, and let* $\mathsf{FSS} = (\mathsf{Gen}, \mathsf{Eval})$ *be a symmetric and shift-invariant FSS for some class of functions* $\mathcal{F}$ *of the form* $f\colon \{0,1\}^{n_2} \to \mathcal{R}$ *with key space* $\mathcal{K} = \mathbb{H}$*, such that for any pair of keys* $(k_0, k_1) \leftarrow \mathsf{Gen}(\hat{f})$ *it holds* $k_0 - k_1 \in H$*.*

Then, there exists an FSS $\mathsf{FSS}^{\otimes} = (\mathsf{Gen}^{\otimes}, \mathsf{Eval}^{\otimes})$ *for the class* $\mathcal{F}^{\mathcal{P}} \otimes \mathcal{F}$ *of functions*

$$F_{P,f}\colon \{0,1\}^{n_1+n_2} \to \mathbb{G},\ (x_1, x_2) \mapsto \begin{cases} f(x_2) & \textit{if } P(x_1) = 1 \\ 0 & \textit{else} \end{cases},$$

for $P \in \mathcal{P}, f \in \mathcal{F}$*, where the resulting keys consist of a "secret" part corresponding to the key space in* $\mathsf{FSS}^{\mathcal{P}}$ *and a "correction word" space* $\tilde{\mathbb{H}}$*.*

Theorem 2.1 yields the following corollary.

Corollary 2.2. *Assume* $\mathsf{PRG} : \mathbb{S} \to \tilde{\mathbb{H}}$ *is a EOH-PRG relative to* $(S, \mathbb{T}, H, \mathbb{H})$ *with* $H = (S \times T)^2, \mathbb{H} = (\mathbb{S} \times \mathbb{T})^2$*. Assume* $\mathcal{P}$ *is a family of predicates* $\{0,1\} \to \{0,1\}$*. Then, there exists an FSS for the function class*

$$F_{P_1 \wedge \cdots \wedge P_\ell}\colon \{0,1\}^\ell \to \{0,1\},\ \mathbf{x} \mapsto \bigwedge_{i=1}^{\ell} P_i(\mathbf{x}[i]),$$

where $P_1, \ldots, P_\ell \in \mathcal{P}$*.*

2.2.3 FSS for branching programs

In general, a branching program can be described as a finite directed acyclic graph with one source node and two sink nodes, *accept* and *reject*. In this section, we focus on giving an FSS construction for oblivious ℓ-layered branching programs with out-degree 2, i.e., each width-w layer uses a fixed position

[9] A subtlety is that in the definition of EOH-PRG we only require correctness relative to random shifts, but here we rely on correctness relative to shifts of the form $t_b \cdot CW - \mathsf{PRG}(s_b)$. This can be solved by having the parties re-randomize their shares with random offset $\mathsf{PRF}(s, i)$ (where each time a fresh index i is used and both parties hold the key s). This is necessary anyway for applying tensoring recursively, and allows for a simpler definition of EOH-PRG. Note though that this requires us to settle with a form of "non-adaptive" correctness as used e.g. in [BKS19], where the inputs are assumed to be chosen independently of the keys.

of the input, each non-sink node has two outgoing edges labeled by 0 and 1, and edges only go from one level to the next level, as specified by a transition function $f\colon [\ell] \times [w] \times \{0,1\} \to [w]$.

Roughly, the idea to obtain an FSS for the class of such branching programs is to extend the tag value $t \in \{0,1\}$ to a *tag vector* $\mathbf{t} \in \{0,1\}^w$, allowing to pick the "right" correction word in going from the i-th to the $(i+1)$-th level.

More precisely, for the i-th level, the key generation algorithm essentially chooses a seed $\mathbf{s}_i \in S^w$, i.e., the j-th node on the i-th level is "labelled" by a seed value $s_{i,j} \in S$ together with a fixed tag vector $\mathbf{e}_j \in \{0,1\}^w$, where $\mathbf{e}_j$ corresponds to the j-th unit vector. Recall that each node has two outgoing edges, 0 and 1. In other words, for each node $(s_{i,j}, \mathbf{e}_j)$ on level i, there exist two possible nodes $j_0 := f(i,j,0)$ and $j_1 := f(i,j,1)$ that can be reached in level $i+1$ with corresponding labels $(s_{i+1,j_0}, \mathbf{e}_{j_0})$ and $(s_{i+1,j_1}, \mathbf{e}_{j_1})$. To go from the i-th to the $(i+1)$-th level, the idea is now to encode the transitions into correction words. More precisely, we define

$$CW_i[j] := \mathsf{PRG}\left(s_{i,j}\right) + \mathsf{Encode}\left((s_{i+1,j_0}, \mathbf{e}_{j_0}), (s_{i+1,j_1}, \mathbf{e}_{j_1})\right).$$

Thus, the correction word for each node can be computed in this way. The FSS key consists of a sharing of the label of start node and correction words.

During evaluation, the tag vector allows to pick the right correction word to proceed to the next level. Namely, given secret shares $(s_b, \mathbf{t}_b)$ such that

$$(s_0, \mathbf{t}_0) + (s_1, \mathbf{t}_1) = (s_{i,j}, \mathbf{e}_j)$$

the parties can compute

$$y_b = \mathsf{Conv}\left(\sum_{k=1}^{w} \mathbf{t}_b[k] \cdot CW_i[k] - \mathsf{PRG}(s_b)\right)$$

to obtain

$$\begin{aligned} y_0 - y_1 = {} & \mathsf{Conv}\left(\sum_{k=1}^{w} \mathbf{t}_0[k] \cdot CW_i[k] - \mathsf{PRG}(s_0)\right) \\ & - \mathsf{Conv}\left(\sum_{k\in[w_i]} \mathbf{t}_1[k] \cdot CW_i[k] - \mathsf{PRG}(s_1)\right) \\ = {} & \mathsf{Conv}(\mathbf{t}_0[j] \cdot CW_i[j] - \mathsf{PRG}(s_0)) + \mathsf{Conv}(\mathbf{t}_1[j] \cdot CW_i[j] - \mathsf{PRG}(s_1)) \\ = {} & \left((s_{i+1,j_0}, \mathbf{e}_{j_0}), (s_{i+1,j_1}, \mathbf{e}_{j_1})\right), \end{aligned}$$

by the property of the EOH-PRG. The parties can now continue the evaluation with the left or right part of the output, depending on the i-th input bit.

Note that in the described inductive construction of FSS for branching programs, we assumed that each level has exactly w nodes. This can be achieved by virtually adding dummy nodes, for which the correction words for dummy nodes can be sampled uniformly at random, since these are never reached during evaluation.

Comparison with FSS via universal branching programs. Previous constructions of FSS for branching programs rely on homomorphic secret sharing (HSS) [BGI16a, Theorem 4.15] or FSS for all functions from fully homomorphic encryption(FHE) [DHRW16, Section 6.3]. Given a branching program P, the construction of [BGI16a] encodes P to $\hat{P}$ and generates a universal branching program (UBP) for P such that $UBP(\hat{P}, x) = P(x)$ for arbitrary x. Next, $\hat{P}$ is secret-shared via the underlying HSS to hide the transition function of P. Note that the transformation via UBPs incurs a considerate efficiency blow-up which is at least quadratic in the number of levels. Refer to Section H.2 for details.

In contrast, the evaluation of our FSS construction directly emulates the evaluation of the branching program. For each level of the program P, the PRG needs to be evaluated once. Moreover, our FSS for branching programs naturally supports multi-edges. For the FSS via universal branching programs, on the other hand, the multi-edges need to be splitted into plain edges, incurring an additional blow-up. Finally, for branching programs with polynomial out-degree, say d, our FSS construction only increases the correction word for each node from two elements to d elements.

2.2.4 FSS for approximate matching functions

It is possible to merge the FSS for bit-fixing predicates and the FSS for branching programs to implement an approximate matching function. The approximate matching function is defined as

$$f_{\mathbf{a},b}(\mathbf{x}) := (\mathsf{dist}(\mathbf{x}, \mathbf{a}) < b)$$

where $\mathbf{a} \in \{0, 1, *\}^\ell, \mathbf{x} \in \{0, 1\}^\ell, b \leq \ell$ and $\mathsf{dist}(\mathbf{x}, \mathbf{a}) = \sum_{i \in [\ell]} (\mathbf{a}[i] \neq * \wedge \mathbf{a}[i] \neq \mathbf{x}[i])$. This can be viewed as a generalization of the bit-fixing predicates, where the distance is compared. In fact, the bit-fixing predicate is the special case $\mathsf{dist}(\mathbf{x}, \mathbf{a}) < 1$. The underlying idea is essentially as follows: The key for each level is sampled as in the key generation algorithm for bit-fixing predicates. The transition functions between two consecutive levels are determined by the distance function, i.e., keeping the last distance or increasing the distance by 1, depending on the special matching state for current level. In the last level, the key are sampled according to the threshold value b, thereby allowing to recover either 0 or 1, depending on the input.

2.2.5 FSS for DFAs

Given a DFA $M := (Q, \Sigma, \delta, q_0, F)$, we first transform the set of accepting states F to a single accept state A via appending a special symbol ϵ to the end of each input. Similarly, we can transfer the remaining states to a rejection state R. The resulting DFA has alphabet $\Sigma \cup \{\epsilon\}$ and states set $Q \cup \{A, R\}$.

Similarly to the FSS for branching programs, the FSS for DFAs focuses on hiding the transition function. For each state $s \in Q$, the label for s consists of a uniformly random seed and a tag vector assigned according to a designated order of the states. The correction word for s hides the labels of one-step reachable states from s. The key for the FSS is a random sharing of the label for q_0 and the correction word for every state in Q. Since a state may be transferred to itself via some symbols, a KDM secure variant of EOH-PRG is necessary. With EOH-PRG, the key size of this FSS is independent of the length of string to be evaluated by the DFA.

2.3 Instantiating the EOH-PRG

In the following we explain our instantiations of EOH-PRG from LWE and a small-exponent DCR variant.

2.3.1 EOH-PRG from LWE

Recall that the LWE assumption naturally provides an almost-homomorphic PRG (AH-PRG). Let $p, q \in \mathbb{N}$ with $p|q$ and let $\lceil \cdot \rfloor_{q \to p}$ be defined as $\lceil \cdot \rfloor_{q \to p} : \mathbb{Z}_q \to \mathbb{Z}_p, x \mapsto \lceil (p/q) \cdot x \rfloor$. Suppose $A \leftarrow_R \mathbb{Z}_q^{n \times m}$. Then, $\mathsf{PRG}_A : \mathbb{Z}_q^n \to \mathbb{Z}_p^m, \mathbf{s} \mapsto \lceil \mathbf{s}A \rfloor_{q \to p}$ is an AH-PRG [BLMR13]. The security of PRG_A follows from the pseudorandomness of LWR distributions and the almost homomorphic property naturally follows from the rounding operation. It is easy to verify that

$$\mathsf{PRG}_A(\mathbf{s}_0 + \mathbf{s}_1) = \mathsf{PRG}_A(\mathbf{s}_0) + \mathsf{PRG}_A(\mathbf{s}_1) + \mathbf{e}$$

with $\|\mathbf{e}\|_\infty \leq 1$.

Note though that an AH-PRG is not sufficient to instantiate our construction, since the small error vector would lead to correctness errors with too high probability. To overcome this problem and obtain an EOH-PRG, we rely on the distributed rounding and lifting technique as introduced in [BKS19]. Concretely, let $r \in \mathbb{N}$ be an integer such that $r|p$ and $1 \ll r \ll p$. Then, [BKS19] observed that for $\mu \in \mathbb{Z}_p$ the following holds. Given $y = (p/r) \cdot \mu + e \mod p$ for small error e, and random additive secret shares $y_0 - y_1 = y \mod p$, it holds

$$\lceil y_0 \rfloor_{p \to r} - \lceil y_1 \rfloor_{p \to r} = \mu \mod r,$$

except with negligible probability. Further, if $|\mu| \ll r$, then this secret sharing holds with overwhelming probability *over the integers* and thus also modulo p:

$$\lceil y_0 \rfloor_{p \to r} - \lceil y_1 \rfloor_{p \to r} = \mu \mod p.$$

Towards obtaining a EOH-PRG, our idea is thus to encode a vector $\mathbf{x} \in \{0, 1\}^m$ as $(p/r) \cdot \mathbf{x}$, which then allows to remove errors potentially introduced via the AH-PRG using the conversion function $\mathbf{y} \mapsto \lceil \mathbf{y} \rfloor_{p \to r}$. More precisely, we instantiate the EOH-PRG as in Theorem 8.1 and 8.2.

2.3.2 EOH-PRG from small-exponent DCR

Next, we outline our EOH-PRG instantiation from a variant of the DCR assumption. Recall that the DCR assumption induces a homomorphic PRG mapping the additive group $(\mathbb{Z}_{\phi(N)}, +)$ to the multiplicative group $(\mathbb{Z}^*_{N^2})^4$. Namely, assume $\mathbf{g} := (g_0, g_1, g_2, g_3) \in \mathbb{Z}^4_{N^2}$, where each g_i is uniformly sampled from the N-th residue group mod N^2. Then, based on the DCR assumption, $\mathsf{PRG}_{\mathbf{g}} : \mathbb{Z}_{\phi(N^2)} \to (\mathbb{Z}^*_{N^2})^4, r \mapsto (g_0^r, g_1^r, g_2^r, g_3^r)$ defines a homomorphic PRG [BG10], for which it holds $G_{\mathbf{g}}(s_0 - s_1) = G_{\mathbf{g}}(s_0)/G_{\mathbf{g}}(s_1)$ for any $s_0, s_1 \in \mathbb{Z}_{\phi(N^2)}$. However, in order to use this recursively in our constructions, we need to be able to recover a homomorphism over $(\mathbb{Z}_{\phi(N^2)}, +)$ (while not revealing $\phi(N^2)$).

To that end, we follow the techniques of [OSY21], who showed that given $z_0 = z_1 \cdot (1+N)^x \mod N^2$ for $x \in \mathbb{Z}_N$, there exists an efficiently computable map $\mathsf{DDLog}\colon \mathbb{Z}^*_{N^2} \to \mathbb{Z}_N$, which satisfies

$$\mathsf{DDLog}(z_0) - \mathsf{DDLog}(z_1) = x \mod N.$$

Further, if $|x| \leq \frac{N}{2^\lambda}$, then $\mathsf{DDLog}(z_0) - \mathsf{DDLog}(z_1) = x$ over $\mathbb{Z}$, and thus in particular it holds

$$\mathsf{DDLog}(z_0) - \mathsf{DDLog}(z_1) = x \mod \phi(N^2),$$

allowing to recover the homomorphism over $(\mathbb{Z}_{\phi(N^2)}, +)$. Note that this allows to generate secret shares of a value x mod $\phi(N^2)$ *without* knowing $\phi(N^2)$, whenever $|x|$ is sufficiently small.

Our idea is thus to build on a small-exponent variant of the DCR assumption which states that $\mathsf{PRG}_{\mathbf{g}}(r)$ remains a PRG restricted to seeds r with $|r| \leq \frac{N}{2^\lambda}$. It is pointed out in [ADOS22] that this variant of the DCR assumption is reasonable as long as the domain of the small exponent is still exponentially large. This kind of low exponent assumption dates back to [KK04].

With this we can state our EOH-PRG from small-exponent DCR as follows. Let B be an integer such that $B \cdot 2^\lambda \leq N$ and $B > 2^\lambda$. Let $m := \ell(1+w)$ (where, again, ℓ, w are determined by the underlying application). Assume the DCR variant assumption holds relative to exponents in B. Then, we can instantiate the EOH-PRG as in Theorem 8.3.

We would like to point out though that to evaluate the PRG, it is not necessary to know $\phi(N^2)$. The computation mod $\phi(N^2)$ or $\phi(N)$ in the exponent is automatic because of the structure of the Paillier group, and to sample from $\mathbb{Z}_{\phi(N^2)}$, we can sample from $\mathbb{Z}_{N^2}$ instead, as the two distributions are statistically close.

3 Preliminaries

In this section, we recall the preliminaries for function secret sharing from [GI99, BGI15]. For the remaining preliminaries we refer to Section A in the Supplementary Material. Here, we only consider *two-party* function secret sharing as all of our constructions are in the two-party setting.

Definition 3.1 (Function Secret Sharing (FSS))**.** *A function secret sharing scheme for function a function class $\mathcal{F}$ consists of two PPT algorithms* $(\mathsf{Gen}, \mathsf{Eval})$*:*

- $\mathsf{Gen}(1^\lambda, f)$ *outputs a pair of keys* (k_0, k_1) *and correction word* CW *upon the security parameter and* $f \in \mathcal{F}$*.*
- $\mathsf{Eval}(b, k_b, CW, x)$ *outputs the corresponding share of* $f(x)$ *upon the party index* b*,* k_b *and input* x*.*

$(\mathsf{Gen}, \mathsf{Eval})$ *is a secure function secret sharing if it satisfies the correctness and security requirements:*

– **Correctness** *For all* $f \in \mathcal{F}$ *and all* $x \in D_f$*,*

$$Pr\left[\mathsf{Eval}(0, k_0, CW, x) - \mathsf{Eval}(1, k_1, CW, x) = f(x) : (k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, f)\right] \\ \geq 1 - \mathsf{negl}(\lambda),$$

where D_f *is the domain of* f*.*

– **Security** *Assume party* z *is corrupted by an adversary* $\mathcal{A}$*. Consider the following experiment.*

1. *The adversary* $\mathcal{A}$ *outputs* $(f_0, f_1) \leftarrow \mathcal{A}(1^\lambda, \mathcal{F})$*.*
2. *The challenger samples* $b \leftarrow \{0,1\}$ *and computes* $(k_0, k_1, CW) \leftarrow \mathsf{Gen}(1^\lambda, f_b)$*.*
3. *The adversary outputs* $b' \leftarrow \mathcal{A}(k_z, CW)$*.*

Let $\mathsf{Adv}(1^\lambda, \mathcal{A})$ *be the advantages of* $\mathcal{A}$ *in guessing* b'*, i.e.,* $\mathsf{Adv}(1^\lambda, \mathcal{A}) := \left|Pr[b = b'] - \frac{1}{2}\right|$*. Then* $(\mathsf{Gen}, \mathsf{Eval})$ *is a secure FSS if* $\mathsf{Adv}(1^\lambda, \mathcal{A})$ *is negligible for every* $b \in \{0, 1\}$ *and every PPT adversary* $\mathcal{A}$*.*

We remark that the FSS definition differs from the FSS in [GI99, BGI15] in a formal sense, since here the correction word is viewed as an independent part whereas in literature the correction word is part of the party's key.

4 FSS with Additional Properties and EOH-PRGs

In this section, we define *shift-invariant* FSS and *symmetric* FSS, which will serve as a basis for our recursive constructions. We further introduce the notion of a PRG with encoded-output homomorphism (EOH-PRG).

Shift-invariance essentially means that the keys remain functional when shifted by an arbitrary shift s. Note that the shift-invariance does not affect the correction word space $\mathcal{CW}$, which is thus listed separately in Definition 4.1. Further, note that all of the FSS constructions in this work satisfy shift-invariance.

Definition 4.1 (Shift-invariant FSS)**.** *Let* $(\mathsf{Gen}, \mathsf{Eval})$ *be an FSS for a function class* $\mathcal{F}$*. Assume the key space* $\mathcal{K}$ *of* $(\mathsf{Gen}, \mathsf{Eval})$ *is a finite abelian group and the correction word space is* $\mathcal{CW}$*. For any* $f \in \mathcal{F}$*, let* D_f *be the domain of* f*. We say* $(\mathsf{Gen}, \mathsf{Eval})$ *is* shift-invariant*, if there exists a negligible function* $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ *such that for all* $\lambda \in \mathbb{N}$*,* $f \in \mathcal{F}$*,* $x \in D_f$*,* $(k_0, k_1, CW) \leftarrow \mathsf{Gen}(1^\lambda, f)$*, and* $s \leftarrow_R \mathcal{K}$*,*

$$\Pr\left[\mathsf{Eval}(0, k_0 + s, CW, x) - \mathsf{Eval}(1, k_1 + s, CW, x) = f(x)\right] \geq 1 - \mathsf{negl}(\lambda),$$

where the probability is taken over the randomness of Gen *and* s*.*

Next, we introduce the notion of symmetric FSS. Note that the FSS schemes for point functions in [BGI15, BGI16b] are also symmetric.

Definition 4.2 (Symmetric FSS)**.** *An FSS is symmetric if for all* $k \in \mathcal{K}$*, for all* $x \in D_f$*,*

$$\mathsf{Eval}(0, k, CW, x) = \mathsf{Eval}(1, k, CW, x).$$

4.1 PRG with Encoded-Output Homomorphism

We now define the notion *PRG with encoded-output homomorphism* (EOH-PRG), which is central to our work. EOH-PRG corresponds to an approximate substitution of ideal homomorphic PRG.

Note that in the following we consider all entities implicitly parametrized by λ (e.g., by a set S we denote an ensemble of sets $S = \{S_\lambda\}_{\lambda \in \mathbb{N}}$). In Section 8, we show how to obtain EOH-PRGs from the (ring)-LWE or DCR assumption.

Definition 4.3 (EOH-PRG)**.** *Let* $\mathbb{S}, \mathbb{H}$ *be finite abelian additive groups, and* $\widetilde{\mathbb{H}}$ *a finite abelian group. Let* $H \subset \mathbb{H}$ *and* $S \subset \mathbb{S}$ *be subsets such that* $0 \in S$*. A function* $\mathsf{PRG}\colon \mathbb{S} \to \widetilde{\mathbb{H}}$ *is a* PRG with encoded-output homomorphism (EOH-PRG) relative to $(S, H, \mathbb{H})$ *if it is a secure PRG relative to* S *and* $\mathbb{S}$*, and there exists a deterministic polynomial-time encoding function* $\mathsf{Encode} : H \to \widetilde{\mathbb{H}}$ *and a* deterministic *polynomial-time conversion function* $\mathsf{Conv} : \widetilde{\mathbb{H}} \to \mathbb{H}$ *such that for all* $m \in H$*, for* $s \leftarrow_R S$ *and*

$$y := \mathsf{PRG}(s) + \mathsf{Encode}(m),$$

$s_0 \leftarrow_R \mathbb{S}$*,* $y_0 \leftarrow_R \widetilde{\mathbb{H}}$*,* $s_1 := s_0 - s$*,* $y_1 := y_0 - y$ *it holds that*

$$\mathsf{Conv}(y_0 - \mathsf{PRG}(s_0)) - \mathsf{Conv}(y_1 - \mathsf{PRG}(s_1)) = m$$

in $\mathbb{H}$ *except with negligible probability over the choice of* s_0 *and* y_0*.*

Note that for $y_0 = y_1, s_0 = s_1$*, we have* $\mathsf{Conv}(y_0 - \mathsf{PRG}(s_0)) = \mathsf{Conv}(y_1 - \mathsf{PRG}(s_1))$ *as* Conv *is deterministic.*

Since for our instantiations we will typically have to work with a "tag space" $\mathbb{T}$ operating on $\widetilde{\mathbb{H}}$, we will slightly extend the definition of EOH-PRG, and typically refer to the below when we talk about an EOH-PRG.

Definition 4.4 (EOH-PRG with "tag-space" $\mathbb{T}$)**.** *Let* $\mathsf{PRG}\colon \mathbb{S} \to \widetilde{\mathbb{H}}$ *be a EOH-PRG relative to* $(S, H, \mathbb{H})$*. We say that it is an EOH-PRG relative to* $(S, \mathbb{T}, H, \mathbb{H})$*, if* $\mathbb{T}$ *is an additive group such that* $T := \{0, 1\} \subset \mathbb{T}$ *and there exists a (non-trivial)*[10] *efficiently computable (left) group operation* $\cdot\colon \mathbb{T} \times \widetilde{\mathbb{H}} \to \widetilde{\mathbb{H}}$ *of* $\mathbb{T}$ *on* $\widetilde{\mathbb{H}}$*.*

[10] I.e., $1 \cdot h \neq 0$ for $h \neq 0$.

Definition 4.5 (EOH-PRG with KDM security)**.** *Let* $\mathsf{PRG}\colon \mathbb{S} \to \widetilde{\mathbb{H}}$ *be a EOH-PRG relative to* $(S, \mathbb{T}, H, \mathbb{H})$*. Let* Ψ *a family of embeddings* $\psi\colon S \to H$*. We say PRG is* KDM secure relative to Ψ*, if for all* $\psi \in \Psi$*,* $\mathsf{PRG}^{\psi}\colon s \mapsto \mathsf{PRG}(s) + \mathsf{Encode}(\psi(s))$ *is a PRG relative to* S *and* $\mathbb{S}$*.*

Remark 4.6. *Note that the share obtained from* Conv *for* m *may be not pseudorandom. In order to ensure that the homomorphic property can be recursively applied, the two parties can use a PRF with shared key to re-randomize the share of* m*.*

Remark 4.7. *For the tensor-product FSS, we need* $H = (S \times T)^2$ *and* $\mathbb{H} = (\mathbb{S} \times \mathbb{T})^2$*, for out-degree* 2 *branching programs we need* $H = (S \times \{\mathbf{e}_i\}_{i=1}^{w})^2$ *and* $\mathbb{H} = (\mathbb{S} \times \mathbb{T}^w)^2$ *where* w *is the width of the branching program and* $\mathbf{e}_i$ *is the* i*-th standard basis of* $\mathbb{T}^w$*.*

Remark 4.8. *We further need that operations over* $\mathbb{S}, \mathbb{H}$ *and* $\mathbb{T}$ *are efficiently computable. While this is trivially the case for our instantiation from LWE, this can be sufficiently emulated for our instantiation with DCR (where* $\mathbb{S}$ *and* $\mathbb{T}$ *are additive modulo an unknown* $\phi(N)$*), by building on techniques of [OSY21].*

5 Tensor Product FSS for Arbitrary Predicates from EOH-PRGs

In this section, we present our tensor product FSS, which allows to tensor FSS schemes for arbitrary predicates, as long as the second FSS is symmetric and shift-invariant.

Theorem 5.1 (Tensor Product FSS)**.** *Let* $n_1, n_2 \in \mathbb{N}$*, and* $\mathbb{S}$*,* $\mathbb{T}$ *be two finite abelian groups. Let* $\mathcal{P}_1$ *be a family of predicates mapping* $\{0,1\}^{n_1}$ *to* $\{0,1\}$ *and* $\mathcal{P}_2$ *be a family of predicates mapping* $\{0,1\}^{n_2}$ *to* $\{0,1\}$*. Let* $\mathcal{F}_{\mathcal{P}_1} : \{0,1\}^{n_1} \to S \times T, \mathcal{F}_{\mathcal{P}_2} : \{0,1\}^{n_2} \to S \times T$ *be the function families induced by* $\mathcal{P}_1, \mathcal{P}_2$ *as*

$$f_{P_1,\beta}\colon \{0,1\}^{n_1} \to S \times T,\ x \mapsto P_1(x) \cdot \beta = \begin{cases} \beta & \text{if } P_1(x) = 1 \\ 0 & \text{else} \end{cases},$$

$$f_{P_2,\gamma}\colon \{0,1\}^{n_2} \to S \times T,\ x \mapsto P_2(x) \cdot \gamma = \begin{cases} \gamma & \text{if } P_2(x) = 1 \\ 0 & \text{else} \end{cases},$$

respectively, with $P_1 \in \mathcal{P}_1, P_2 \in \mathcal{P}_2$ *and* $\beta \in S \times \{1\}, \gamma \in S \times \{1\}$*.*

Assume

1. $\mathsf{PRG} : \mathbb{S} \to \widetilde{\mathbb{H}}$ *is a EOH-PRG relative to* $(S, \mathbb{T}, H, \mathbb{H})$ *(as in Definition 4.4), where* $\mathbb{H} := (\mathbb{S} \times \mathbb{T})^2$ *and* $H := (S \times \{0,1\})^2$
2. $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_1}}(\mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_1}}, \mathsf{Eval}^{\mathcal{F}_{\mathcal{P}_1}})$ *is an FSS for* $\mathcal{F}_{\mathcal{P}_1}$ *over key space* $\mathcal{K}_1$*, correction word space* $\mathcal{CW}_1$ *with pseudorandom correction words and pseudorandom output shares.*
3. $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_2}}(\mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_2}}, \mathsf{Eval}^{\mathcal{F}_{\mathcal{P}_2}})$ *is a symmetric and shift-invariant FSS for* $\mathcal{F}_{\mathcal{P}_2}$ *over key space* $\mathcal{K}_2 := \mathbb{H}$*, such that for all* $(u_0, u_1) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_2}}$ *it holds* $u_0 - u_1 \in H$*, with correction word space* $\mathcal{CW}_2$*, and with pseudorandom correction words and output shares.*
4. $\mathsf{PRF}\colon \{0,1\}^{\lambda} \times [N] \to \mathbb{H}$ *is a PRF (for* N *sufficiently large).*

Then there exists $\mathsf{FSS}^{\otimes}(\mathsf{Gen}^{\otimes}, \mathsf{Eval}^{\otimes})$ *for* $\mathcal{G} := \mathcal{F}_{\mathcal{P}_1} \otimes \mathcal{F}_{\mathcal{P}_2} = \{g_{P_1,P_2,\gamma} : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to S \times T\}$ *over key space* $\mathcal{K}_1$*, correction word space* $\mathcal{CW}_1 \times \mathcal{CW}_2 \times \widetilde{\mathbb{H}}$*, with pseudorandom correction words and pseudorandom output shares, where*

$$g_{P_1,P_2,\gamma}(x_1, x_2) := P_1(x_1) \cdot P_2(x_2) \cdot \gamma = \begin{cases} \gamma & \text{if } P_1(x_1) = 1 \wedge P_2(x_2) = 1 \\ 0 & \text{else} \end{cases}.$$

In particular, $\mathsf{FSS}^{\otimes}$ *is symmetric and shift-invariant if* $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_1}}$ *is symmetric and shift-invariant.*

The construction for $(\mathsf{Gen}^{\otimes}, \mathsf{Eval}^{\otimes})$ is shown in Figure 1. For the proof we refer to Section B in the Supplementary Material. The FSS for bit-fixing predicates from EOH-PRG in Section C can be viewed as the tensor product of FSS for length 1 predicates.

We further explain how to obtain FSS schemes for the for negation and disjunction of predicates in Section B.2.

Function secret sharing scheme $\mathsf{FSS}^{\otimes} = (\mathsf{Gen}^{\otimes}, \mathsf{Eval}^{\otimes})$ from EOH-PRG:

Parameters: Let $\mathsf{PRG}\colon \mathbb{S} \to \widetilde{\mathbb{H}}$ be a EOH-PRG relative to $(S, \mathbb{T}, H, \mathbb{H})$, where $\mathbb{H} := (\mathbb{S} \times \mathbb{T})^2$ and $H := (S \times \{0,1\})^2$, and corresponding functions $\mathsf{Encode}, \mathsf{Conv}$. Let $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_1}}(\mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_1}}, \mathsf{Eval}^{\mathcal{F}_{\mathcal{P}_1}})$ be an FSS for $\mathcal{F}_{\mathcal{P}_1}$ over key space $\mathcal{K}_1$ and correction word space $\mathcal{CW}_1$. Let $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_2}}(\mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_2}}, \mathsf{Eval}^{\mathcal{F}_{\mathcal{P}_2}})$ be a symmetric shift-invariant FSS for $\mathcal{F}_{\mathcal{P}_2}$ over key space $\mathcal{K}_2 = \mathbb{H}$ and correction word space $\mathcal{CW}_2$ such that for any two keys u_0, u_1 in the image of $\mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_2}}$ it holds $u_0 - u_1 \in H$. Further let $N \in \mathbb{N}$ (sufficiently large) and $\mathsf{PRF}\colon \{0,1\}^\lambda \times [N] \to \mathbb{H}$ a PRF. We assume that both parties have access to a global key $K \leftarrow_R \{0,1\}^\lambda$ and global state $\mathsf{st} \in [N]$.

```
Gen^⊗(1^λ, g_{P_1,P_2,γ}) :
1: Sample $s \leftarrow_R S$ and let $\beta := (s, 1)$. Then $\beta \in S \times T \subset \mathbb{S} \times \mathbb{T}$.      ▷ $\gamma =: (\sigma, 1) \in S \times T$.
2: Let $(k_0, k_1, CW_1) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_1}}(1^\lambda, f_{P_1,\beta})$.
3: Let $(u_0, u_1, CW_2) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_2}}(1^\lambda, f_{P_2,\gamma})$.      ▷ $u_0, u_1 \in \mathbb{H}$ s.t. $u_0 - u_1 \in H$.
4: $CW \leftarrow \mathsf{PRG}(s) + \mathsf{Encode}(u_0 - u_1)$.
5: Let $CW^{\otimes} := (CW_1, CW_2, CW)$ be the new correction word.
6: **Return** $(k_0, k_1, CW^{\otimes})$.
Eval^⊗(b, k_b, CW, (x_1, x_2)) :
1: Parse $CW^{\otimes}$ as $CW^{\otimes} =: (CW_1, CW_2, CW)$.
2: Let $(s_b, t_b) = \mathsf{Eval}^{\mathcal{F}_{\mathcal{P}_1}}(b, k_b, CW_1, x_1)$.      ▷ $(s_b, t_b) \in \mathbb{S} \times \mathbb{T}$ and $(s_0 - s_1, t_0 - t_1) \in S \times T$.
3: Compute $v_b \leftarrow t_b \cdot CW - \mathsf{PRG}(s_b)$.
4: Compute $w_b \leftarrow \mathsf{Conv}(v_b) + \mathsf{PRF}(K, \mathsf{st})$.      ▷ $w_b \in \mathbb{H}$ and $w_0 - w_1 \in H$.
5: Update the state $\mathsf{st} \leftarrow \mathsf{st} + 1$.
6: **Return** $\mathsf{Eval}^{\mathcal{F}_{\mathcal{P}_2}}(b, w_b, CW_2, x_2)$.
```

Figure 1: FSS $(\mathsf{Gen}^{\otimes}, \mathsf{Eval}^{\otimes})$ for $\mathcal{F}_{\mathcal{P}_1} \times \mathcal{F}_{\mathcal{P}_2}$ from $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_1}}$, $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_2}}$ and EOH-PRG.

Remark 5.2. *As instantiations of the EOH-PRG for N-parties seem out of reach with current techniques without relying on (multi-key) FHE, we did not give the details of the N-party tensor product construction. Roughly, the requirement on shift-invariance of the underlying FSS would become that for $\sum_{i\in[N]} s_i = 0$ it holds $\sum_{i\in[N]} \mathsf{Eval}(i, k_i + s_i, CW, x) = f(x)$ with overwhelming probability, and the requirement on symmetry would become that for $\sum_{i\in[N]} k_i = 0$, it holds $\sum_{i\in[N]} \mathsf{Eval}(i, k_i, CW, x) = 0$. To achieve these properties, the N-party EOH-PRG has to satisfy that (i) $\sum_{i\in[N]} \mathsf{Conv}(c_i - \mathsf{PRG}(s_i)) = m$ for $\sum_{i\in[N]} c_i = \mathsf{PRG}(s) + \mathsf{Encode}(m)$, as well as (ii) $\mathsf{PRG}(0) = 0$ and $\mathsf{Encode}(0) = 0$. Here, requirement (i) is necessary to achieve shift invariance, and the additional requirement (ii) is necessary to achieve symmetry (which is satisfied in both our LWE or DCR instantiation in the two-party case).*

6 FSS for Branching Programs

In this section, we generalize the FSS for tensor products to FSS for branching programs. Concretely, the one-bit tag is extended to a w-bit tag, which supports polynomially many possible choices (corresponding to the number of nodes in one level of the branching program). We also generalize the FSS for branching program to FSS for DFAs, approximate matching functions and multivariate polynomials. For details, we refer to Section 7 and E.

Recall that given a branching program P, the size is the number of nodes in V, the length is ℓ, and the width is the maximal number of nodes of every level. Note that every branching program can be converted to a layered, *input-oblivious* branching program with polynomial blowup in size [Pip79, BGI16a].

Now, we start to construct an FSS for branching programs. Let P be a layered, oblivious branching program of width w, and let $P_i : \{0,1\}^n \to [w_i]$ be the function which evaluates P to level i (i.e., to the state of the branching program at level i). We start by explaining how to obtain the FSS for the "first level" function

$$f_{P_1,\gamma=(\gamma_0,\gamma_1)} \colon \{0,1\}^n \to (S \times T^{w_1})^2, \mathbf{x} \mapsto \gamma_{x_1},$$

Note that there is one node in level 0 (the initial node) and two nodes in level 1 (one for the choice of 0 and 1 for the choice of 1), i.e., $w_1 = 2$ and P_1 considers only the first bit x_1 of the input $\mathbf{x} \in \{0,1\}^n$.

In order to be able to recurse, we set $\gamma_b := (s_b, \mathbf{t}_b)$, where $s_b \in S$ is some random seed and $\mathbf{t}_0 = (1,0)$ and $\mathbf{t}_1 = (0,1)$ are the unit vectors over $\{0,1\}^2$. A function secret sharing scheme for $f_{P_1,\gamma=(\gamma_0,\gamma_1)}$ which satisfies shift-invariance over $(\mathbb{S}, \mathbb{T}^{w_1})^2$ for some abelian groups $\mathbb{S}, \mathbb{T}$ with $S \subset \mathbb{S}$, $T \subset \mathbb{T}$ can be obtained via a direct truth table sharing.

Lemma 6.1 (Base case)**.** *Let $\mathbb{S}$ and $\mathbb{T}$ be finite abelian groups, and let $S \subset \mathbb{S}$, $T \subset \mathbb{T}$ be arbitrary subsets. Then, there exists a shift-invariant FSS for the family of functions $f_{P_1,\gamma}$ over key space $(\mathbb{S} \times \mathbb{T}^2)^2$.*

Next, we show an inductive lemma to construct an FSS for P which extends an FSS for P_i to an FSS for P_{i+1}.

Lemma 6.2 (Inductive)**.** *Assume*

1. *$\mathsf{FSS}^i = (\mathsf{Gen}^i, \mathsf{Eval}^i)$ is a shift-invariant FSS for P_i over key space $(\mathbb{S} \times \mathbb{T}^2)^2$, correction word space $\mathcal{CW}_i$ with pseudorandom correction word and pseudorandom output share. FSS^i maps the input $\mathbf{x}$ with index set $\{\tau(V_0), \tau(V_1) \ldots, \tau(V_{i-1})\}$ to the $P_i(\mathbf{x})$-th position of a given array β.*

2. *$\mathsf{PRG}_i : \mathbb{S} \to \widetilde{\mathbb{H}}_i$ is a EOH-PRG relative to $(S, \mathbb{T}, H_i, \mathbb{H}_i)$ as in Definition 4.4 where $H_i = (S \times T^{w_{i+1}})^2, \mathbb{H}_i = (\mathbb{S} \times \mathbb{T}^{w_{i+1}})^2$.*

Then, there exists a shift-invariant FSS for P_{i+1} over key space $(\mathbb{S}\times\mathbb{T}^2)^2$, correction word space $\mathcal{CW}_i \times \widetilde{\mathbb{H}}_i^{w_i}$ with pseudorandom correction word and pseudorandom output share. Again, FSS^{i+1} maps the input $\mathbf{x}$ with index set $\{\tau(V_0), \tau(V_1) \ldots \tau(V_{i-1}), \tau(V_i)\}$ to the $P_{i+1}(\mathbf{x})$-th position of a given array γ.

The FSS $\mathsf{FSS}^{i+1} = (\mathsf{Gen}^{i+1}, \mathsf{Eval}^{i+1})$ for P_{i+1} is shown in Figure 2. For the proof of this lemma we refer to Section D.

Function secret sharing scheme $\mathsf{FSS}^{i+1} = (\mathsf{Gen}^{i+1}, \mathsf{Eval}^{i+1})$ from EOH-PRG for P_{i+1}:

Parameters: Let $\mathsf{PRG}_i : \mathbb{S} \to \widetilde{\mathbb{H}}_i$ be a EOH-PRG relative to $(S, \mathbb{T}, (S \times T^{w_{i+1}})^2, (\mathbb{S} \times \mathbb{T}^{w_{i+1}})^2)$. Let $\mathsf{FSS}^i = (\mathsf{Gen}^i, \mathsf{Eval}^i)$ be a shift-invariant FSS for P_i over key space $\mathcal{K}$ and correction word space $\mathcal{CW}$. Further let $N \in \mathbb{N}$ (sufficiently large) and $\mathsf{PRF}\colon \{0,1\}^\lambda \times [N] \to \mathbb{H}$ is a PRF. We assume that both parties have access to a global key $K \leftarrow_R \{0,1\}^\lambda$ and global state $\mathsf{st} \in [N]$.

```
Gen^{i+1}(1^λ, P_{i+1}, γ ∈ (S × T^{w_{i+1}})^{w_{i+1}}) :
 1: Parse $\gamma$ as $\gamma =: ((\sigma[1], \mathbf{e}_1), (\sigma[2], \mathbf{e}_2) \ldots (\sigma[w_{i+1}], \mathbf{e}_{w_{i+1}}))$ with $\sigma \in S^{w_{i+1}}$ and $\mathbf{e}_j \in T^{w_{i+1}}$ the $j$-th
    basis in $\mathbb{T}^{w_{i+1}}$.                                    ▷ $\sigma$ is the seed value for level $i+1$.
 2: Sample $\mathbf{s} \leftarrow_R S^{w_i}$.                                  ▷ $\mathbf{s}$ is the seed value for level $i$.
 3: Let $\beta \leftarrow ((\mathbf{s}[1], \mathbf{e}_1), (\mathbf{s}[2], \mathbf{e}_2) \ldots (\mathbf{s}[w_i], \mathbf{e}_{w_i})) \in (S \times T^{w_i})^{w_i}$ with $\mathbf{e}_j \in T^{w_i}$ the $j$-th basis in $\mathbb{T}^{w_i}$.
 4: Let $(k_0, k_1, CW_i) \leftarrow \mathsf{Gen}^i(1^\lambda, P_{i,\beta})$.
 5: Let $CW[1:w_i] \in \widetilde{\mathbb{H}}_i^{w_i}$ be the correction word for level $i$ computed as follows.
 6: for $j \in [w_i]$ do
 7:     Let $\mathbf{u}_j \in (S \times T^{w_{i+1}})^2$ be the key for level $i+1$ with 2 elements.
 8:     Set $\mathbf{u}_j[0] \leftarrow \gamma[f(i,j,0)]$ and $\mathbf{u}_j[1] \leftarrow \gamma[f(i,j,1)]$. ▷ Map 0 and 1 to the corresponding element of
    level $i+1$ in the array $\gamma$.
 9:     Set $CW[j] \leftarrow \mathsf{PRG}(\mathbf{s}[j]) + \mathsf{Encode}(\mathbf{u}_j)$.
10: end for
11: Let $CW_{i+1} := (CW_i, CW[1:w_i])$ be the new correction word.
12: Return $(k_0, k_1, CW_{i+1})$.
Eval^{i+1}(b, k_b, CW_{i+1}, x) :
 1: Parse $CW_{i+1}$ as $CW_{i+1} =: (CW_i, CW[1:w_i])$.
 2: Let $(s_b, \mathbf{t}_b) = \mathsf{Eval}^i(b, k_b, CW_i, \mathbf{x})$.                        ▷ $(s_b, \mathbf{t}_b) \in \mathbb{S} \times \mathbb{T}^{w_i}$.
 3: Compute $v_b \leftarrow \mathsf{Conv}(\sum_{j\in[w]} \mathbf{t}_b[j] \cdot CW[j] - \mathsf{PRG}(s_b)) + \mathsf{PRF}(K, \mathsf{st})$.      ▷ $v_b \in (\mathbb{S} \times \mathbb{T}^{w_{i+1}})^2$.
 4: Update the state $\mathsf{st} \leftarrow \mathsf{st} + 1$.
 5: Return $v_b[\mathbf{x}[\tau(V_{i+1})]] \in \mathbb{S} \times \mathbb{T}^{w_{i+1}}$.     ▷ Use the input $\mathbf{x}[\tau(V_{i+1})]$ to choose the key for level $i+1$.
```

Figure 2: FSS $(\mathsf{Gen}^{i+1}, \mathsf{Eval}^{i+1})$ for P_{i+1} from FSS $(\mathsf{Gen}^i, \mathsf{Eval}^i)$ and EOH-PRG PRG.

With this, we can obtain an FSS for branching programs of arbitrarily polynomially-bounded width and length, as captured in the following theorem.

Theorem 6.3 (FSS for BP from EOH-PRG)**.** *Let P be a branching program with width $(w_0, w_1, \ldots, w_\ell)$ for each level, where $w_0 = 1, w_1 = 2, w_\ell = 2, w_i \le w$ for $i \in [0, \ell]$. Assume $\mathsf{PRG}_i : \mathbb{S} \to \widetilde{\mathbb{H}}_i$ is a EOH-PRG relative to $S, \mathbb{T}, H_i = (S \times T^{w_{i+1}})^2, \mathbb{H}_i = (\mathbb{S} \times \mathbb{T}^{w_{i+1}})^2$ for $i \in [1, \ell]$.*

Then, there exists an FSS for P over key space $(\mathbb{S}\times\mathbb{T}^2)^2$ and correction word space $\widetilde{\mathbb{H}}_1^{w_1}\times\widetilde{\mathbb{H}}_2^{w_2}\cdots\times\widetilde{\mathbb{H}}_{\ell-1}^{w_{\ell-1}}$, i.e., with key size bounded by $2(\log|\mathbb{S}|+2\log|\mathbb{T}|)+\sum_{i\in[1,\ell-1]} w_i\log\left|\widetilde{\mathbb{H}}_i\right|$.

Note that the FSS construction for branching program in Theorem 6.3 only hides the transition function f whereas the topology of the branching program, i.e., the number of nodes of each level, is revealed. For a topology-hiding construction we refer to Section D.4 in the Supplementary Material.

7 FSS for DFAs

Similar to the FSS for branching programs, the FSS for DFAs mainly hides the transition function for each state. Given a DFA $M := (Q,\Sigma,\delta,F,q_0)$, the set of accepts states F could be transferred to one accept state A via appending an empty string ϵ to the input whereas the set of other states is transferred to one rejection state R. The transformation leads to a DFA with $|Q|+2$ states and with alphabet $\Sigma\cup\{\epsilon\}$. The construction relies on a KDM secure EOH-PRG.

Theorem 7.1 (FSS for DFAs)**.** *Let $M := (Q,\Sigma,\delta,q_0,F)$ be a DFA. Let $\mu := |Q\cup\{A,R\}| = |Q|+2$. Assume $\mathsf{PRG}:\mathbb{S}\to\widetilde{\mathbb{H}}$ be a KDM secure EOH-PRG relative to $(S,\mathbb{T},(S\times T^{\mu})^{|\Sigma|+1},(\mathbb{S}\times\mathbb{T}^{\mu})^{|\Sigma|+1})$.*

There exists a FSS for M over key space $\mathbb{S}\times\mathbb{T}^{\mu}$ and correction word space $\widetilde{\mathbb{H}}^{|Q|}$. Futhermore, the key size is bounded by $\log|\mathbb{S}|+\mu\log|\mathbb{T}|+|Q|\log\left|\widetilde{\mathbb{H}}\right|$.

The FSS construction works as follows. Let $\mathsf{PRG}:\mathbb{S}\to\widetilde{\mathbb{H}}$ be a KDM secure EOH-PRG relative to $(S,\mathbb{T},(S\times T^{\mu})^{|\Sigma|+1},(\mathbb{S}\times\mathbb{T}^{\mu})^{|\Sigma|+1})$. For each state s, there exists a uniformly sampled seed and an assigned tag vector according to the order of the states. The correction word for state s hides the keys of one-step reachable states from s according to the order of the alphabet specified by the transition function. The key of the FSS consists of a random sharing of the key for q_0 and the correction words for all of the meaningful states, which are sorted according to the order of the states in the tag vector. Note there are totally $|Q|$ correction words and no correction words for $\{A,R\}$ as the two states are two end states.

Given an input $\mathbf{x} := (x_1\ldots x_\ell)$, the evaluation appends an empty string ϵ before evaluation. During evaluation, the two-party will first obtain the shared correction word for current state, which is achieved via the shared tag vector, the consistent orders of the tag vectors and the correction words. Next, the two-party obtains the shared keys of the one-step reachable states from the shared correction word and the shared seed for current state. Next, the two-party uses the current input to obtain the shared key for next state and recursively evaluates the FSS for next symbol of the input until encounters the empty string. The FSS for M is shown in Figure 3.

The correctness is easy to verify as the the tag vector is used in Lemma 6.2. The security follows from the pseudorandomness of the KDM secure PRG. We explain why KDM secure EOH-PRG is required. For a state s, it is possible that $\delta(s,\alpha)=s$ for some input $\alpha\in\Sigma$, which leads to the correction word $CW[s]\leftarrow\mathsf{PRG}(\sigma[s])+\mathsf{Encode}(\mathbf{u})$ and $\sigma[s]$ is also contained in the α-th entry of $\mathbf{u}$. The KDM secure EOH-PRG can be instantiated from LWE or DCR for free without assuming circular security(Remark 8.4). The running time of Eval relies on the input length as DFAs.

Remark 7.2. *The FSS for DFAs could be viewed as one level of the FSS for branching programs and there is no level change during the evaluation.*

8 EOH-PRG Instantiated from LWE or DCR Assumption

In this section, we show EOH-PRG instantiated from LWE or a DCR variant assumption. As remarked following Definition 4.4, if a PRG is homomorphic then the PRG itself is a EOH-PRG. The LWE assumption implies an almost homomorphic PRG and the DCR assumption implies a homomorphic PRG mapping an additive group to a multiplicative group. We show how the AH-PRG from LWE or the H-PRG from DCR cooperate with other tools to implement the EOH-PRG.

Here we show the three instantiations from LWE, Ring-LWR and DCR. The preliminaries for the assumption, proof of the instantiations and remarks appear in Section F.

Theorem 8.1 (EOH-PRG from LWE)**.** *Let $n=n(\lambda), p=p(\lambda), q=q(\lambda), r=r(\lambda), B=B(\lambda)\in\mathbb{N}$ such that $r|p, p|q, 2\lambda^{\omega(1)}\le r$, $2Br\lambda^{\omega(1)}\le p$ and $n\log q\le\ell(n+w)\log p$. Let w,ℓ be parameters depending on concrete applications.*

Function secret sharing scheme for DFA from KDM secure EOH-PRG

Parameters: Let $\mathsf{PRG} : \mathbb{S} \to \widetilde{\mathbb{H}}$ be a KDM secure EOH-PRG relative to $(S, \mathbb{T}, (S \times T^{\mu})^{|\Sigma|+1}, (\mathbb{S} \times \mathbb{T}^{\mu})^{|\Sigma|+1})$ and $\mu := |Q \cup \{A, R\}| = |Q|+2$. Further let $N \in \mathbb{N}$ (sufficiently large) and $\mathsf{PRF}\colon \{0,1\}^{\lambda}\times[N] \to \mathbb{H}$ is a PRF. We assume that both parties have access to a global key $K \leftarrow_R \{0,1\}^{\lambda}$ and global state $\mathsf{st} \in [N]$.

```
Gen(1^λ, M) :
 1: Assign a fixed order to $Q \cup \{A, R\}$ and $\Sigma \cup \{\epsilon\}$.
 2: Sample $\sigma \leftarrow_R S^{\mu}$.
 3: Let $\beta \leftarrow ((\sigma[1], \mathbf{e}_1) \dots (\sigma[\mu], \mathbf{e}_{\mu})) \in (S \times T^{\mu})^{\mu}$ with $\mathbf{e}_j \in T^{\mu}$ the $j$-th basis in $\mathbb{T}^{\mu}$.
 4: Let $CW[1 : |Q|]$ be the correction words for meaningful states computed as follows.
 5: for each state $s \in Q$ do
 6:     Let $\mathbf{u} \in (S \times T^{\mu})^{|\Sigma|+1}$ be the keys for one-step reachable states from $s$.
 7:     for symbol $\alpha \in \Sigma \cup \{\epsilon\}$ do
 8:         Let $t \leftarrow \delta(s, \alpha)$.
 9:         Set $\mathbf{u}[\alpha] \leftarrow \beta[t]$.                          ▷ $\beta[t]$ is the key for $t$.
10:     end for
11:     Set $CW[s] \leftarrow \mathsf{PRG}(\sigma[s]) + \mathsf{Encode}(\mathbf{u})$.          ▷ No correction words for $\{A, R\}$.
12: end for
13: Sample $k_0, k_1 \leftarrow_R \mathbb{S} \times \mathbb{T}^{\mu}$ such that $k_0 - k_1 = \beta[q_0]$.      ▷ Share the key for $q_0$.
14: Return $(k_0, k_1, CW)$.
Eval(b, k_b, CW, x, i) :
 1: Return $k_b$ if $i > \mathsf{len}(\mathbf{x})$.
 2: Parse $k_b$ as $k_b =: (s_b, \mathbf{t}_b) \in \mathbb{S} \times \mathbb{T}^{\mu}$.
 3: Compute $v_b \leftarrow \mathsf{Conv}(\sum_{j\in[\mu]} \mathbf{t}_b[j] \cdot CW[j] - \mathsf{PRG}(s_b)) + \mathsf{PRG}(K, \mathsf{st}) \in (\mathbb{S} \times \mathbb{T}^{\mu})^{|\Sigma|+1}$.
 4: Update the state $\mathsf{st} \leftarrow \mathsf{st} + 1$.
 5: Return $\mathsf{Eval}(b, v_b[\mathbf{x}[i]], CW, \mathbf{x}, i + 1)$.     ▷ Use the input $\mathbf{x}[i]$ to choose the key for next state.
```

Figure 3: FSS for a DFA from KDM secure EOH-PRG.

Assume $\mathsf{LWE}_{n,\ell(n+w),q}$ *is hard. Let* $S = \{0,1\}^n, T = \{0,1\}, \mathbb{S} = \mathbb{Z}_p^n, \mathbb{T} = \mathbb{Z}_p, H = (S \times T^w)^{\ell} = \{0,1\}^{\ell(n+w)}, \mathbb{H} = (\mathbb{S}\times\mathbb{T}^w)^{\ell} = \mathbb{Z}_p^{\ell(n+w)}, \widetilde{\mathbb{H}} = \mathbb{Z}_p^{\ell(n+w)}$ *and the corresponding functions for the instantiation be*

- *The homomorphic group operation* $\cdot\colon \mathbb{Z}_p \times \mathbb{Z}_p^{\ell(n+w)} \to \mathbb{Z}_p^{\ell(n+w)}, (t, \mathbf{s}) \mapsto t \cdot \mathbf{s}$.
- $\mathsf{PRG}\colon \mathbb{Z}_p^n \to \mathbb{Z}_p^{\ell(n+w)}, \mathbf{s} \mapsto \lceil \mathbf{s}A \rfloor_{q\to p}$, *where* $A \in \mathbb{Z}_q^{n\times \ell(n+w)}$ *and the vector-matrix multiplication* $\mathbf{s}A$ *is performed modulo* q*;*
- $\mathsf{Encode}\colon \{0,1\}^{\ell(n+w)} \to \mathbb{Z}_p^{\ell(n+w)}, \mathbf{s} \mapsto \frac{p}{r} \cdot \mathbf{s}$*;*
- $\mathsf{Conv}\colon \mathbb{Z}_p^{\ell(n+w)} \to \mathbb{Z}_p^{\ell(n+w)}, \mathbf{t} \mapsto \lceil \mathbf{t} \rfloor_{p\to r}$.

Then PRG *is a EOH-PRG relative to* $(S, \mathbb{T}, H, \mathbb{H})$.

Similarly, we show the EOH-PRG instantiation from Ring-LWE. As pointed out in Lemma A.6, to work with a binary secret Module-LWE instances, the rank of the Module-LWE should be at least $\log q$ (basing on the Ring-LWE pseudorandomness). However, small secret Module-LWR has been used in the NIST post-quantum cryptography submissions including Saber [DKR+20], Kyber [SAB+22] for constant rank. Based on this, we show the instantiation for good efficiency relying on the Ring-LWE assumption.

Note it is possible that the number $\ell(n+w)$ is not exactly a multiple of n. Assume $\ell\cdot(n+w) = n\cdot\mu+\gamma$ with $0 \le \gamma < n$. The EOH-PRG from Ring-LWE output has μ elements from $\mathcal{R}_p$ and γ elements from $\mathbb{Z}_p$.

Theorem 8.2 (EOH-PRG from Ring-LWR)**.** *Let* $n = n(\lambda), p = p(\lambda), q = q(\lambda), r = r(\lambda), B = B(\lambda) \in \mathbb{N}$ *such that* $r|p, p|q, 2\lambda^{\omega(1)} \le r$, $2Br\lambda^{\omega(1)} \le p$ *and* $n \log q \le \ell(n+w) \log p$*. Let* w, ℓ *be parameters depending on concrete applications. Denote* $\mathcal{R}$ *as the algebraic ring with degree* n*.*

Assume binary secret $\mathsf{Ring\text{-}LWR}_{\mathcal{R},\ell+\lceil \frac{\ell w}{n}\rceil,q,p}$ *is hard. Let* $S = \{0,1\}^n, T = \{0,1\}, \mathbb{S} = \mathcal{R}_p, \mathbb{T} = \mathbb{Z}_p, H = (S \times T^w)^{\ell} = \{0,1\}^{\ell(n+w)}, \mathbb{H} = (\mathbb{S} \times \mathbb{T}^w)^{\ell} = \mathcal{R}_p^{\ell+\lfloor \frac{\ell w}{n}\rfloor} \times \mathbb{Z}_p^{\ell w-n\lfloor \frac{\ell w}{n}\rfloor}, \widetilde{\mathbb{H}} = \mathcal{R}_p^{\ell+\lfloor \frac{\ell w}{n}\rfloor} \times \mathbb{Z}_p^{\ell w-n\lfloor \frac{\ell w}{n}\rfloor}$ *and the corresponding functions for the instantiation be*

- *The homomorphic group operation* $\cdot : \mathbb{Z}_p \times \left(\mathcal{R}_p^\ell \times \mathbb{Z}_p^{\ell w - n\lfloor \frac{\ell w}{n} \rfloor}\right) \to \mathcal{R}_p^\ell \times \mathbb{Z}_p^{\ell w - n\lfloor \frac{\ell w}{n} \rfloor}, (t, s) \mapsto t \cdot s$, *where* $\cdot$ *is the scalar multiplication mod* p.

- PRG: $\mathcal{R}_p \to \mathcal{R}_p^\ell \times \mathbb{Z}_p^{\ell w - n\lfloor \frac{\ell w}{n} \rfloor}, s \mapsto \psi(\lceil s \cdot \mathbf{a} \rfloor_{q \to p})$, *where* $\mathbf{a} \in \mathcal{R}_q^{\ell + \lceil \frac{\ell w}{n} \rceil}$, *the multiplication* $s \cdot \mathbf{a}$ *is performed modulo* $\mathcal{R}_q$, *and* $\psi(\cdot)$ *takes the ring elements except the last one and the first* $\ell w - n \left\lceil \frac{\ell w}{n} \right\rfloor$ *coefficients of the last ring element;*

- Encode: $\{0,1\}^{\ell(n+w)} \to \mathcal{R}_p^\ell \times \mathbb{Z}_p^{\ell w - n\lfloor \frac{\ell w}{n} \rfloor}, \mathbf{s} \mapsto \frac{p}{r} \cdot s$;

- Conv: $\mathcal{R}_p^\ell \times \mathbb{Z}_p^{\ell w - n\lfloor \frac{\ell w}{n} \rfloor} \to \mathcal{R}_p^\ell \times \mathbb{Z}_p^{\ell w - n\lfloor \frac{\ell w}{n} \rfloor}, t \mapsto \lceil t \rfloor_{p \to r}$.

Then PRG *is a EOH-PRG relative to* $(S, \mathbb{T}, H, \mathbb{H})$.

Next we show how the EOH-PRG is instantiated from the DCR variant assumption It is pointed out in [ADOS22, Section 4.1], the DCR variant assumption is sound if the domain of the small exponent is exponentially large. This kind of low exponent assumption dates back to [KK04] and was also used in [BCG+17]. To enable the homomorphic group operation, here we use $\mathbb{Z}_{\phi(N^2)}$ instead of $\mathbb{Z}_{\phi(N)}$ for the additive group.

Theorem 8.3 (EOH-PRG from DCR)**.** *Let* B *be an integer such that* $B \cdot 2^\lambda \le N$ *and* $B > 2^\lambda$.
Assume the DCR variant assumption holds. Let $S = [-\frac{B}{2}, \frac{B}{2}], T = \{0,1\}, \mathbb{S} = \mathbb{Z}_{\phi(N^2)}, \mathbb{T} = \mathbb{Z}_{\phi(N^2)}, H = (S \times T^w)^\ell, \mathbb{H} = (\mathbb{S} \times \mathbb{T}^w)^\ell = \mathbb{Z}_{\phi(N^2)}^{\ell(1+w)}, \widetilde{\mathbb{H}} = (\mathbb{Z}_{N^2}^*)^{\ell(1+w)}$ *and the corresponding functions for the instantiation be*

- *The homomorphic group operation* $\cdot : \mathbb{Z}_{\phi(N^2)} \times (\mathbb{Z}_{N^2}^*)^{\ell(1+w)} \to (\mathbb{Z}_{N^2}^*)^{\ell(1+w)}, (t, \mathbf{s}) \mapsto \mathbf{s}^t \mod N^2$.

- PRG : $\mathbb{Z}_{\phi(N^2)} \to (\mathbb{Z}_{N^2}^*)^{\ell(1+w)}, s \mapsto \mathbf{g}^s$, *where* $\mathbf{g} \in (\mathbb{Z}_{N^2}^*)^{\ell(1+w)}$ *and each entry of* $\mathbf{g}$ *is a* N*-th residue;*

- Encode : $([-B/2, B/2] \times \{0,1\}^w)^\ell \to (\mathbb{Z}_{N^2}^*)^{\ell(1+w)}, \mathbf{m} \mapsto (1+N)^{\mathbf{m}} \mod N^2$;

- Conv : $(\mathbb{Z}_{N^2}^*)^{\ell(1+w)} \to \mathbb{Z}_{\phi(N^2)}^{\ell(1+w)}, \mathbf{t} \mapsto \mathsf{DDLog}(\mathbf{t})$.

Then PRG *is a EOH-PRG relative to* $(S, \mathbb{T}, H, \mathbb{H})$.

Note that the secret key $\phi(N)$ for Paillier encryption is not *explicitly* used in the operations of instantiation of EOH-PRG from DCR. The computation mod $\phi(N^2)$ or $\phi(N)$ in the exponent is automatic because of the structure of the Paillier group, and to sample from $\mathbb{Z}_{\phi(N^2)}$, we can sample from $\mathbb{Z}_{N^2}$ instead, as the two distributions are statistically close.

Remark 8.4 (KDM Security)**.** *The FSS constructions for DFAs in Section 7 rely on the KDM security of EOH-PRG. It is straightforward to prove the pseudorandomness for LWE or DCR following the method to prove the KDM security in [ACPS09, Theorem 6] or [BHHO08, Section 3.2] as detailed in Remark F.9.*

Acknowledgements

E. Boyle's research is supported in part by AFOSR Award FA9550-21-1-0046 and ERC Project HSS (852952). L. Kohl is funded by NWO Talent Programme Veni (VI.Veni.222.348) and by NWO Gravitation project QSC. Z. Li is funded by NWO Gravitation project QSC and ERC ADG ALGSTRONGCRYPTO (740972). P. Scholl is funded by the Independent Research Fund Denmark under project number 0165-00107B (C3PO).

References

[AB06] Shai Avidan and Moshe Butman. Efficient methods for privacy preserving face detection. In *NIPS 2006*, 2006. 43

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, USA, 1st edition, 2009. 47

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 595–618, August 2009. 24, 42

[ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 421–452, August 2022. 7, 17, 24, 42

[AHLR18] Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin. Privacy-preserving search of similar patients in genomic data. *PoPETs*, 2018(4):104–124, October 2018. 45

[AMPR19] Navid Alamati, Hart Montgomery, Sikhar Patranabis, and Arnab Roy. Minicrypt primitives with algebraic structure and applications. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 55–82, May 2019. 5

[BBC$^+$21] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy*, pages 762–776. IEEE Computer Society Press, May 2021. 4

[BCG$^+$17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2105–2122. ACM Press, October / November 2017. 7, 24, 42

[BCG$^+$19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518, August 2019. 4

[BCG$^+$21] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 871–900, October 2021. 4, 8, 9, 10, 48

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018. 4

[BDIR18] Fabrice Benhamouda, Akshay Degwekar, Yuval Ishai, and Tal Rabin. On the local leakage resilience of linear secret sharing schemes. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 531–561, August 2018. 9

[BFK$^+$09] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In Michael Backes and Peng Ning, editors, *ESORICS 2009*, volume 5789 of *LNCS*, pages 424–439, September 2009. 10

[BFL$^+$11] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Privacy-preserving ECG classification with branching programs and neural networks. *IEEE Trans. Inf. Forensics Secur.*, 6(2):452–468, 2011. 10

[BG10] Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 1–20, August 2010. 7, 17, 42

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367, April 2015. 4, 9, 10, 11, 17, 18, 39, 47, 48

[BGI16a] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539, August 2016. 4, 6, 8, 10, 15, 20, 36, 43, 47, 48, 50

[BGI16b] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016. 4, 6, 9, 10, 11, 18, 36, 47, 48

[BGI19] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 341–371, December 2019. 4, 8

[BGM$^+$16] Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. On the hardness of learning with rounding over small modulus. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 209–224, January 2016. 7, 30

[BHHO08] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 108–125, August 2008. 24, 42

[BJRW23] Katharina Boudgoust, Corentin Jeudy, Adeline Roux-Langlois, and Weiqiang Wen. On the hardness of module learning with errors with short distributions. *Journal of Cryptology*, 36(1):1, January 2023. 30

[BKS19] Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 3–33, May 2019. 4, 5, 6, 7, 8, 9, 10, 14, 16, 40, 41, 47, 48, 49, 50

[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, August 2013. 10, 16, 31

[Boy22] Elette Boyle. Function Secret Sharing, 2022. The 12th BIU Winter School on cryptography - Advances in Secure Computation. 4

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 719–737, April 2012. 10, 30

[BPSW07] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 498–507. ACM Press, October 2007. 10

[BPTG15] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS 2015*. The Internet Society, February 2015. 10

[CBM15] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE Computer Society Press, May 2015. 4

[CCD$^+$20] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya P. Razenshteyn, and M. Sadegh Riazi. SANNS: Scaling up secure approximate k-nearest neighbors search. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2111–2128. USENIX Association, August 2020. 9, 45

[CLSF10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF: binary robust independent elementary features. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision - ECCV 2010, 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV*, volume 6314 of *Lecture Notes in Computer Science*, pages 778–792. Springer, 2010. 43

[CZZM07] Rui Cai, Chao Zhang, Lei Zhang, and Wei-Ying Ma. Scalable music recommendation by search. In Rainer Lienhart, Anand R. Prasad, Alan Hanjalic, Sunghyun Choi, Brian P. Bailey, and Nicu Sebe, editors, *Proceedings of the 15th International Conference on Multimedia 2007, Augsburg, Germany, September 24-29, 2007*, pages 1065–1074. ACM, 2007. 44

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122, August 2016. 4, 10, 15

[DKN+20] Alex Davidson, Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Adaptively secure constrained pseudorandom functions in the standard model. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 559–589, August 2020. 9

[DKR+20] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions`. 23

[DRPS22] Emma Dauterman, Mayank Rathee, Raluca Ada Popa, and Ion Stoica. Waldo: A private time-series database from function secret sharing. In *SP 2022*, pages 2450–2468. IEEE, 2022. 4

[Ds17] Jack Doerner and abhi shelat. Scaling ORAM for secure computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 523–535. ACM Press, October / November 2017. 4

[EFG+09] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In Ian Goldberg and Mikhail J. Atallah, editors, *PETS 2009*, volume 5672 of *LNCS*, pages 235–253, August 2009. 44

[FGJS17] Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith III. Homomorphic secret sharing from paillier encryption. In Tatsuaki Okamoto, Yong Yu, Man Ho Au, and Yannan Li, editors, *ProvSec 2017*, volume 10592 of *LNCS*, pages 381–399, October 2017. 4, 10

[GHS16] Rosario Gennaro, Carmit Hazay, and Jeffrey S. Sorensen. Automata evaluation and text search protocols with simulation-based security. *Journal of Cryptology*, 29(2):243–282, April 2016. 44

[GI99] Niv Gilboa and Yuval Ishai. Compressing cryptographic resources. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 591–608, August 1999. 10, 17, 18

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658, May 2014. 4, 9

[glo] glob – Linux Programmer's Manual – Library Functions. `https://man7.org/linux/man-pages/man3/glob.3.html`. Accessed: 2023-01-16. 44

[HMEK11] Yan Huang, Lior Malka, David Evans, and Jonathan Katz. Efficient privacy-preserving biometric identification. In *NDSS 2011*. The Internet Society, February 2011. 45

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *30th ACM STOC*, pages 604–613. ACM Press, May 1998. 45

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 575–594, February 2007. 4

[JB22] T. Janani and M. Brindha. Secure similar image matching (SESIM): an improved privacy preserving image retrieval protocol over encrypted cloud database. *IEEE Trans. Multim.*, 24:3794–3806, 2022. 8, 44

[KG09] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, 2009. 44

[KK04] Takeshi Koshiba and Kaoru Kurosawa. Short exponent Diffie-Hellman problems. In Feng Bao, Robert Deng, and Jianying Zhou, editors, *PKC 2004*, volume 2947 of *LNCS*, pages 173–186, March 2004. 7, 17, 24, 42

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition.* CRC Press, 2014. 31

[KM21] Anunay Kulshrestha and Jonathan R. Mayer. Identifying harmful media in end-to-end encrypted communication: Efficient private membership computation. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 893–910. USENIX Association, August 2021. 8, 44

[KNL$^+$19] Ágnes Kiss, Masoud Naderpour, Jian Liu, N. Asokan, and Thomas Schneider. SoK: Modular and efficient private decision tree evaluation. *PoPETs*, 2019(2):187–208, April 2019. 10

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998. 45

[LLR94] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. In *35th FOCS*, pages 577–591. IEEE Computer Society Press, November 1994. 45

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23, May / June 2010. 30

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. 75(3):565–599, 2015. 30

[MJF$^+$21] Jiayi Ma, Xingyu Jiang, Aoxiang Fan, Junjun Jiang, and Junchi Yan. Image matching from handcrafted to deep features: A survey. *Int. J. Comput. Vis.*, 129(1):23–79, 2021. 43

[MNSS12] Payman Mohassel, Salman Niksefat, Seyed Saeed Sadeghian, and Babak Sadeghiyan. An efficient protocol for oblivious DFA evaluation and applications. In Orr Dunkelman, editor, *CT-RSA 2012*, volume 7178 of *LNCS*, pages 398–415, February / March 2012. 44

[MyS] MySQL 8.0 Reference Manual: 3.3.4.7 Pattern Matching. `https://dev.mysql.com/doc/refman/8.0/en/pattern-matching.html`. Accessed: 2023-01-16. 44

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708, October 2021. 4, 5, 6, 7, 8, 9, 10, 17, 19, 41, 48, 49, 50, 51

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238, May 1999. 8, 31, 44

[PAM22] Kittiphop Phalakarn, Nuttapong Attrapadung, and Kanta Matsuura. Efficient oblivious evaluation protocol and conditional disclosure of secrets for DFA. In Giuseppe Ateniese and Daniele Venturi, editors, *ACNS 22*, volume 13269 of *LNCS*, pages 605–625, June 2022. 44

[Pip79] Nicholas Pippenger. On simultaneous resource bounds (preliminary version). In *FOCS*, pages 307–311. IEEE Computer Society, 1979. 20, 36

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. 30

[RPH05] Deepak Ravichandran, Patrick Pantel, and Eduard H. Hovy. Randomized algorithms and NLP: using locality sensitive hash functions for high speed noun clustering. In *ACL 2005*, 2005. 45

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R. Bradski. ORB: an efficient alternative to SIFT or SURF. In Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc Van Gool, editors, *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, pages 2564–2571. IEEE Computer Society, 2011. 43

[RS59] Michael O Rabin and Dana Scott. Finite automata and their decision problems. *IBM journal of research and development*, 3(2):114–125, 1959. 4

[RS21] Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 687–717, Virtual Event, August 2021. 4, 6, 7, 10, 50

[SAB$^+$22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`. 23

[Sip97] Michael Sipser. *Introduction to the theory of computation.* PWS Publishing Company, 1997. 4, 32

[SKSJ08a] Jagarlamudi Shashank, Palivela Kowshik, Kannan Srinathan, and C. V. Jawahar. Private content based image retrieval. In *CVPR 2008*. IEEE Computer Society, 2008. 43

[SKSJ08b] Jagarlamudi Shashank, Palivela Kowshik, Kannan Srinathan, and C. V. Jawahar. Private content based image retrieval. In *CVPR*. IEEE Computer Society, 2008. 44

[SLD22] Sacha Servan-Schreiber, Simon Langowski, and Srinivas Devadas. Private approximate nearest neighbor search with sublinear communication. In *SP*, pages 911–929. IEEE, 2022. 4, 9, 45, 46

[SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society Press, May 2000. 8, 44

[Tcl] New Regular Expression Features in Tcl 8.1. `https://www.tcl.tk/doc/howto/regexp81.tml`. Accessed: 2023-01-16. 44

[TKK19] Anselme Tueno, Florian Kerschbaum, and Stefan Katzenbeisser. Private evaluation of decision trees using sublinear cost. *PoPETs*, 2019(1):266–286, January 2019. 10

[WFNL16] David J. Wu, Tony Feng, Michael Naehrig, and Kristin E. Lauter. Privately evaluating decision trees and random forests. *PoPETs*, 2016(4):335–355, October 2016. 10

[WW05] Ingo Wegener and Philipp Woelfel. New results on the complexity of the middle bit of multiplication. In *CCC*, pages 100–110. IEEE Computer Society, 2005. 8

[WYG$^+$17] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *NSDI 2017*, 2017. 4, 9, 43, 44, 48

A Supplementary Materials for Preliminaries

Notations. We use λ to denote the security parameter. For integers $p < q$, the rounding function $\lceil\cdot\rfloor_{q\to p}$ is defined as $\lceil\cdot\rfloor_{q\to p} : \mathbb{Z}_q \to \mathbb{Z}_p, x \mapsto \lceil(p/q)\cdot x\rfloor$.

A.1 LWE and LWR

Definition A.1 (LWE)**.** *[Reg05] Let $n, m, q \in \mathbb{N}$ and χ be a noise distribution over $\mathbb{Z}$. Assume $\mathbf{s} \leftarrow_R \mathbb{Z}_q^n$. Let $m = \mathsf{poly}(n)$, $A \leftarrow_R \mathbb{Z}_q^{n\times m}$ and $\mathbf{e} \leftarrow \chi^m$. Given*

$$(A, \mathbf{s}A + \mathbf{e}),$$

the $\mathsf{LWE}_{n,m,q,\chi}$ problem is to output the vector $\mathbf{s}$.

Recall that $\mathsf{LWE}_{n,m,q,\chi}$ problem is exactly the search LWE problem. Denote η-LWE as the LWE instance with secret sampled from $\{-\eta\ldots 0\ldots\eta\}^n$.

Definition A.2 (LWR)**.** *[BPR12] Let $n, m, q, p \in \mathbb{N}$. The $\mathsf{LWR}_{n,m,q,p}$ problem is to distinguish between the following two distributions:*

$$(A, \lceil\mathbf{s}A\rfloor_{q\to p}) \text{ and } (A, \lceil\mathbf{u}\rfloor_{q\to p}),$$

where $\mathbf{s} \leftarrow_R \mathbb{Z}_q^n, m = \mathsf{poly}(n), A \leftarrow_R \mathbb{Z}_q^{n\times m}$ and $\mathbf{u} \leftarrow_R \mathbb{Z}_q^m$.

For super-polynomial ratio q/p, [BPR12] proved that LWR is as hard as LWE. Bogdanov et al. [BGM+16] established the hardness of binary secret LWR for polynomial ratio q/p. A similar result can be directly achieved via applying the reduction of [BPR12] to binary secret LWE.

Lemma A.3. *[BGM+16, Theorem 3] Assume $\mathbf{s}$ is sampled from $\{0,1\}^n$. For polynomial ratio q/p, the $\mathsf{LWR}_{n,m,q,p}$ distribution is pseudorandom under the assumption that the corresponding $\mathsf{LWE}_{n,m,q,\chi}$ is hard.*

If each entry of the matrix and the vector is sampled from $\mathcal{R}_q := \mathbb{Z}_q[X]/(X^n + 1)$ rather than $\mathbb{Z}_q$, then the problem is Module-LWE.

Definition A.4 (Module-LWE)**.** *[LS15] Let $n, m, q, d \in \mathbb{N}$. Let $\mathcal{R}_q := \mathbb{Z}_q[X]/(X^n + 1)$ and χ be a noise distribution over $\mathcal{R}$. Assume $\mathbf{s} \leftarrow_R \mathcal{R}_q^d$. Let $m = \mathsf{poly}(n, d)$, $A \leftarrow_R \mathcal{R}_q^{d\times m}$ and $\mathbf{e} \leftarrow \chi^d$. Given*

$$(A, \mathbf{s}A + \mathbf{e}),$$

the $\mathsf{Module\text{-}LWE}_{d,m,q,\chi}$ problem is to output the vector $\mathbf{s}$.

Here d is the rank of Module-LWE problem. If $d = 1$, then it is Ring-LWE problem [LPR10].

Definition A.5 (Module-LWR)**.** *Let $d, n, m, q, p \in \mathbb{N}$. Let $\mathcal{R}_q := \mathbb{Z}_q[X]/(X^n+1)$. The $\mathsf{Module\text{-}LWR}_{n,m,q,p}$ problem is to distinguish between the following two distributions:*

$$(A, \lceil\mathbf{s}A\rfloor_{q\to p}) \text{ and } (A, \lceil\mathbf{u}\rfloor_{q\to p}),$$

where $\mathbf{s} \leftarrow_R \mathcal{R}_q^d, m = \mathsf{poly}(n), A \leftarrow_R \mathcal{R}_q^{d\times m}$ and $\mathbf{u} \leftarrow_R \mathcal{R}_q^m$.

Boudgoust et al. [BJRW23] proved the small secret Module-LWE with larger rank is at least as hard as random secret Module-LWE with smaller rank.

Lemma A.6. *[BJRW23, Theorem 3.2] For any cyclotomic field of degree n, there exists a PPT reduction from the decisional $\mathsf{Module\text{-}LWE}_{k,m,q,D_\alpha}$ assumption to decisional η-$\mathsf{Module\text{-}LWE}_{d,m,q,D_\beta}$ assumption, where $d\cdot\log_2(2\eta+1) \geq (k+1)\cdot\log_2 q + \omega(\log_2 n)$, $d \leq m \leq \mathsf{poly}(n)$, $\alpha \geq \sqrt{n}/q\cdot\sqrt{\ln(2nm(1+\epsilon^{-1}))/\pi}$, $\beta \geq \alpha\cdot n\eta\sqrt{2d}\sqrt{4n^2\eta^2+1}$ and D_α is a Gaussian distribution with s.d. α.*

A.2 Almost Homomorphic PRGs(AH-PRG)

Definition A.7 (Homomorphic PRGs)**.** *Let $\mathbb{G}_0, \mathbb{G}_1$ be two finite abelian additive groups. A function $G : \mathbb{G}_0 \to \mathbb{G}_1$ is a* homomorphic *PRG if it is a secure PRG and for all $x_0, x_1 \in \mathbb{G}_0$,*

$$G(x_0 + x_1) = G(x_0) + G(x_1).$$

There are no known PRGs such that both $\mathbb{G}_0$ and $\mathbb{G}_1$ are *additive* groups in the typical sense. Thus, the definition is relaxed to *almost* homomorphic.

Definition A.8 (Almost Homomorphic PRGs(B-AH-PRG))**.** *Let $\mathbb{G}_0, \mathbb{G}_1$ be two finite abelian additive groups where $\mathbb{G}_2$ is equipped with norm $\|\cdot\|$. Let $B \in \mathbb{N}$. A function $G : \mathbb{G}_0 \to \mathbb{G}_1$ is B-AH-PRG if for all $x_0, x_1 \in \mathbb{G}_0$,*

$$G(x_0 + x_1) = G(x_0) + G(x_1) + e,$$

where $e \in \mathbb{G}_2$ and $\|e\| \leq B$.

Next we show an instantiation of 1-AH-PRG from LWR problem [BLMR13].

Example A.9 (AH-PRG from LWR)**.** *Assume $A \leftarrow_R \mathbb{Z}_q^{n \times m}$. Then*

$$G_A(\mathbf{s}) := \lceil \mathbf{s}A \rfloor_{q \to p}$$

is an 1-AH-PRG under the assumption that $\mathsf{LWE}_{n,m,q}$ is hard relative to $\|\cdot\|_\infty$.

The security of G_A follows from the pseudorandomness of LWR distributions and the almost homomorphic property naturally follows from the rounding operation. Note that here almost-homomorphic property does not require the super-polynomial ratio q/p.

A.3 DCR Assumption

Definition A.10 (DCR Assumption [Pai99])**.** *Let $N = PQ$ be an RSA modulus [KL14]. The DCR assumption is that for any PPT adversary there is no advantage to distinguish between an N-th residue over $\mathbb{Z}_{N^2}^*$ and an arbitrary element of $\mathbb{Z}_{N^2}^*$.*

A.4 Bit-fixing Predicates and CNF/DNF Formulae

Definition A.11 (Bit-fixing Predicates)**.** *Given a vector $\mathbf{v} \in \{0, 1, *\}^\ell$, the bit-fixing predicates $P_{\mathbf{v}}^{\mathsf{BF}} : \{0,1\}^\ell \to \{0,1\}$ specified by the vector $\mathbf{v}$ is defined as*

$$P_{\mathbf{v}}^{\mathsf{BF}}(\mathbf{x}) := \bigwedge_{i \in [\ell]} (\mathbf{v}[i] = \mathbf{x}[i] \vee \mathbf{v}[i] = *).$$

The family of bit-fixing predicates specified by length ℓ predicates is defined as

$$\mathcal{P}_\ell^{\mathsf{BF}} := \{P_{\mathbf{v}}^{\mathsf{BF}} : \mathbf{v} \in \{0, 1, *\}^\ell\}.$$

Definition A.12 (CNF/DNF formulae)**.** *Given a set of n variables $\{u_1, \ldots, u_n\}$, a conjunctive normal form(CNF) formula is an AND of ORs of literals, e.g., a CNF formula φ is defined as*

$$\varphi := \bigwedge_i \bigvee_j v_{i_j},$$

where each v_{i_j} is either a variable u_k or its negation $\neg u_k$. Each $\bigvee_j v_{i_j}$ is called a clause of φ. A CNF formula φ is called k-CNF if every clause of φ has at most k literals.
A disjunctive normal form (DNF) formula is an OR of ANDs of literals, i.e., a DNF formula ϕ is defined as

$$\phi := \bigvee_i \bigwedge_j w_{i_j}.$$

Similarly, a DNF formula is called k-DNF if every clause of ϕ has at most k literals.

A.5 DFA

Definition A.13 (Deterministic Finite Automata (DFA)). *[Sip97, Definition 1.15] A DFA M is defined by $(Q, \Sigma, \delta, q_0, F)$ where*

- *Q is the state set,*
- *Σ is the alphabet,*
- *$\delta : Q \times \Sigma \to Q$ a transition function,*
- *$q_0 \in Q$ is the start state,*
- *$F \subset Q$ is the set of accept states.*

A DFA M accepts a string $\mathbf{x} = (x_1 \dots x_\ell) \in \Sigma^\ell$ if there exists a sequence of states $(s_0 \dots s_\ell)$ such that

1. $s_0 = q_0$,
2. $\delta(s_i, x_{i+1}) = s_{i+1}$ for $i \in \{0, 1 \dots \ell - 1\}$,
3. $s_\ell \in F$.

DFAs recognize the set of regular languages [Sip97, Definition 1.16].

B Tensor Product and FSS for Negation and Disjunction of Predicates

In this section, we present the missing proofs of Section 5 and the FSS for negation and disjunction of predicates.

B.1 Proof of Theorem 5.1

Proof. We directly prove $\mathsf{FSS}^{\otimes}$ is symmetric and shift-invariant if $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_1}}$ is symmetric and shift-invariant. We first prove correctness and shift-invariance. Let $\sigma \in \mathcal{K}_1$ be an arbitrary random shift for the key of the two-party. Thus, the target is to prove that for all $P_1 \in \mathcal{P}_1, P_2 \in \mathcal{P}_2, x$,

$$\Pr[\mathsf{Eval}^{\otimes}(0, k_0 + \sigma, CW^{\otimes}, x) - \mathsf{Eval}^{\otimes}(1, k_1 + \sigma, CW^{\otimes}, x) = g_{P_1,P_2}(x) \cdot \gamma : \\ (k_0, k_1, CW^{\otimes}) \leftarrow \mathsf{Gen}^{\otimes}(1^\lambda, g_{P_1,P_2})] \geq 1 - \mathsf{negl}(\lambda).$$

Recall that in $\mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_1}}$, it is set to output $\beta := (s, 1)$ if $P_1(x) = 1$ whereas to output $(0, 0)$ if $P_1(x_1) = 0$. We continue the proof by distinguish between two cases depending on x_1.

Case $P_1(x_1) = 1$: Recall that in $\mathsf{Eval}^{\otimes}$, we use (s_b, t_b) to denote the intermediate evaluation result according to the input x_1. Because $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_1}}$ is shift-invariant, thus $(s_0, t_0) - (s_1, t_1) = (s, 1)$ over $(\mathbb{S}, \mathbb{T})$ with overwhelming probability. That the share (s_b, t_b) is pseudorandom follows from the definition of $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_1}}$. Recall that $CW \leftarrow \mathsf{PRG}(s) + \mathsf{Encode}(u_0 - u_1)$. From the definition of EOH-PRG, as (s_0, s_1) is a random sharing of s, the key (w_0, w_1) for $\mathsf{FSS}^{\mathcal{P}_2}$ satisfies

$$\begin{aligned} w_0 - w_1 &= (\mathsf{Conv}(t_0 \cdot CW - \mathsf{PRG}(s_0)) + \mathsf{PRG}(K, \mathsf{st})) \\ &\quad - (\mathsf{Conv}(t_1 \cdot CW - \mathsf{PRG}(s_1)) + \mathsf{PRF}(K, \mathsf{st})) \\ &= u_0 - u_1 \end{aligned}$$

except with negligible probability. Recall that (u_0, u_1) is the key for the shift-invariant FSS $(\mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_2}}, \mathsf{Eval}^{\mathcal{F}_{\mathcal{P}_2}})$. Thus, for $P_1(x_1) = 1$,

$$\Pr[\mathsf{Eval}^{\mathcal{F}_{\mathcal{P}_2}}(b, w_0, CW_2, x_2) - \mathsf{Eval}^{\mathcal{F}_{\mathcal{P}_2}}(b, w_1, CW_2, x_2) = P_2(x_2) \cdot \gamma] \geq 1 - \mathsf{negl}(\lambda).$$

Hence, for $P_1(x_1) = 1$, the FSS satisfies shift-invariance and correctness holds with probability $1 - \mathsf{negl}(\lambda)$.

Case $P_1(x_1) = 0$: Then $(s_0, t_0) = (s_1, t_1)$ with overwhelming probability. Thus, the key (w_0, w_1) for $\mathsf{FSS}^{\mathcal{P}_2}$ satisfies

$$w_0 - w_1 = \mathsf{Conv}(t_0 \cdot CW - \mathsf{PRG}(s_0)) - \mathsf{Conv}(t_1 \cdot CW - \mathsf{PRG}(s_1)) = 0.$$

as Conv is deterministic. Therefore for $P_1(\mathbf{x}_1) = 0$,

$$\Pr[\mathsf{Eval}^{\mathcal{F}_{\mathcal{P}_2}}(0, w_0, CW_2, x_2) - \mathsf{Eval}^{\mathcal{F}_{\mathcal{P}_2}}(1, w_1, CW_2, x_2) = 0] \geq 1 - \mathsf{negl}(\lambda)$$

as $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_2}}$ is symmetric.

Hence, with overwhelming probability, the resulting FSS satisfies correctness and shift-invariance.

Next we show that the FSS also satisfies symmetry. If $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_1}}$ is symmetric, for identical keys the two-party obtains identical (s_b, t_b) on arbitrary input x_1 and identical (v_b, w_b) as PRG and Conv are both deterministic. Hence, the two-party finally obtains same value as also $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_2}}$ is symmetric.

Next we prove that the final output share is pseudorandom. From the assumption that $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_2}}$ has pseudorandom output share, once the evaluation for $\mathsf{FSS}^{\otimes}$ is correct, the $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_2}}$ leads to pseudorandom final output share.

The security of $\mathsf{FSS}^{\otimes}$ follows the security of $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_1}}$, the security of $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_2}}$ and the pseudorandomness of PRG via a series of games. It is formally proved in Lemma B.1. □

Lemma B.1 (Security)**.** *If* PRG *is a PRG with encoded-output homomorphism,* $(\mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_1}}, \mathsf{Eval}^{\mathcal{F}_{\mathcal{P}_1}})$ *is a secure FSS for* $\mathcal{F}_{\mathcal{P}_1}$*, and* $(\mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_2}}, \mathsf{Eval}^{\mathcal{F}_{\mathcal{P}_2}})$ *is a secure FSS for* $\mathcal{F}_{\mathcal{P}_2}$ *then* $(\mathsf{Gen}^{\otimes}, \mathsf{Eval}^{\otimes})$ *is a secure FSS for* $\mathcal{F}_{\mathcal{P}_1} \times \mathcal{F}_{\mathcal{P}_2}$*.*

Furthermore, the correction word $CW^{\otimes}$ *is pseudorandom.*

Proof of Lemma B.1. We prove the security via a series of hybrid games. Concretely, for any two functions $g_{P_1,P_2,\gamma}$ and $g_{Q_1,Q_2,\gamma'}$ with $P_1, Q_1 \in \mathcal{P}_1$ and $P_2, Q_2 \in \mathcal{P}_2$, we transfer the key distribution of g_{P_1,P_2} to the key distribution of g_{Q_1,Q_2} step-by-step. For each party index b, the party key consists of k_b and $CW^{\otimes}$. Now we define the hybrid distributions. The changed part of each distribution is explicitly represented with boxes.

H_0 : The key distribution for $g_{P_1,P_2,\gamma}$.

$$H_0(b, g_{P_1,P_2,\gamma}) := \left\{ (k_b, CW^{\otimes}) : \begin{array}{c} s \leftarrow_R S \\ (k_0, k_1, CW_1) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_1}}(1^\lambda, f_{P_1,(s,1)}) \\ (u_0, u_1, CW_2) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_2}}(1^\lambda, f_{P_2,\gamma}) \\ CW \leftarrow \mathsf{PRG}(s) + \mathsf{Encode}(u_0 - u_1) \in \widetilde{\mathbb{H}} \\ CW^{\otimes} := (CW_1, CW_2, CW) \end{array} \right\}$$

H_1 : (Security of $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_1}}$) Change the key for P_1 to Q_1. Note that the correction word CW is still computed with respect to the seed s for $g_{P_1,P_2,\gamma}$.

$$H_1(b, g_{P_1,P_2,\gamma}, g_{Q_1,Q_2,\gamma'}) := \left\{ (k_b, CW^{\otimes}) : \begin{array}{c} \boxed{s, s' \leftarrow_R S} \\ \boxed{(k_0, k_1, CW_1) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_1}}(1^\lambda, f_{Q_1,(s',1)})} \\ (u_0, u_1, CW_2) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_2}}(1^\lambda, f_{P_2,\gamma}) \\ CW \leftarrow \mathsf{PRG}(s) + \mathsf{Encode}(u_0 - u_1) \in \widetilde{\mathbb{H}} \\ CW^{\otimes} := (CW_1, CW_2, CW) \end{array} \right\}$$

From the security of $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_1}}$, the key distributions for P_1 and Q_1 are identical. Thus the two distributions $H_0(b, g_{P_1,P_2,\gamma})$ and $H_1(b, g_{P_1,P_2,\gamma}, g_{Q_1,Q_2,\gamma'})$ are identical.

H_2 : (PRG security) Change $\mathsf{PRG}(s)$ to random $\mathcal{S}$. Note that s is not used in the remaining part.

$$H_2(b, g_{P_1,P_2,\gamma}, g_{Q_1,Q_2,\gamma'}) := \left\{ (k_b, CW^{\otimes}) : \begin{array}{c} \boxed{\mathcal{S} \leftarrow_R \widetilde{\mathbb{H}}} \\ s' \leftarrow_R S \\ (k_0, k_1, CW_1) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_1}}(1^\lambda, f_{Q_1,(s',1)}) \\ (u_0, u_1, CW_2) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_2}}(1^\lambda, f_{P_2,\gamma}) \\ \boxed{CW \leftarrow \mathcal{S} + \mathsf{Encode}(u_0 - u_1) \in \widetilde{\mathbb{H}}} \\ CW^{\otimes} := (CW_1, CW_2, CW) \end{array} \right\}$$

By the pseudorandomness of the output of PRG, the two distributions $H_1(b, g_{P_1,P_2,\gamma}, g_{Q_1,Q_2,\gamma'})$ and $H_2(b, g_{P_1,P_2,\gamma}, g_{Q_1,Q_2,\gamma'})$ are computational indistinguishable.

H_3 : (Security of $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_2}}$) Change CW_2 for P_2 to Q_2.

$$H_3(b, g_{P_1,P_2,\gamma}, g_{Q_1,Q_2,\gamma'}) := \left\{ (k_b, CW^{\otimes}) : \begin{array}{c} \boxed{\mathcal{S}' \leftarrow_R \widetilde{\mathbb{H}}} \\ s' \leftarrow_R S \\ (k_0, k_1, CW_1) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_1}}(1^\lambda, f_{Q_1,(s',1)}) \\ \boxed{(u_0, u_1, CW_2) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_2}}(1^\lambda, f_{Q_2,\gamma'})} \\ \boxed{CW \leftarrow \mathcal{S}' \in \widetilde{\mathbb{H}}} \\ CW^{\otimes} := (CW_1, CW_2, CW) \end{array} \right\}$$

The two distributions $H_2(b, g_{P_1,P_2,\gamma}, g_{Q_1,Q_2,\gamma'})$ and $H_3(b, g_{P_1,P_2,\gamma}, g_{Q_1,Q_2,\gamma'})$ are computational indistinguishable from the security of $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_2}}$. Actually, the distribution H_3 is completely independent of g_{P_1,P_2}. Moreover, CW_2 is pseudorandom as $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}_2}}$ outputs pseudorandom correction words. Next we can apply the first two steps via an reversed order to obtain the key distribution for $g_{Q_1,Q_2,\gamma'}$.

As CW is random, we can change it to a value relying on the key (u_0, u_1) without changing the distribution.

$$H_3(b, g_{Q_1,Q_2,\gamma'}) := \left\{ (k_b, CW^{\otimes}) : \begin{array}{c} \mathcal{S}' \leftarrow_R \widetilde{\mathbb{H}} \\ s' \leftarrow_R S \\ (k_0, k_1, CW_1) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_1}}(1^\lambda, f_{Q_1,(s',1)}) \\ (u_0, u_1, CW_2) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_2}}(1^\lambda, f_{Q_2,\gamma'}) \\ \boxed{CW \leftarrow \mathcal{S}' + \mathsf{Encode}(u_0 - u_1) \in \widetilde{\mathbb{H}}} \\ CW^{\otimes} := (CW_1, CW_2, CW) \end{array} \right\}$$

H_4 : (PRG security) Change $\mathcal{S}'$ to $\mathsf{PRG}(s')$.

$$H_4(b, g_{Q_1,Q_2,\gamma'}) := \left\{ (k_b, CW^{\otimes}) : \begin{array}{c} s' \leftarrow_R S \\ (k_0, k_1, CW_1) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_1}}(1^\lambda, f_{Q_1,(s',1)}) \\ (u_0, u_1, CW_2) \leftarrow \mathsf{Gen}^{\mathcal{F}_{\mathcal{P}_2}}(1^\lambda, f_{Q_2,\gamma'}) \\ \boxed{CW \leftarrow \mathsf{PRG}(s') + \mathsf{Encode}(u_0 - u_1) \in \widetilde{\mathbb{H}}} \\ CW^{\otimes} := (CW_1, CW_2, CW) \end{array} \right\}$$

Now the distribution $H_4(b, g_{Q_1,Q_2,\gamma'})$ is exactly the key distribution for $g_{Q_1,Q_2,\gamma'}$, namely, $H_4(b, g_{Q_1,Q_2,\gamma'}) = H_0(b, g_{Q_1,Q_2,\gamma'})$.

By combining the distributions from H_0 to H_4, for each party index b, the key distribution for $g_{P_1,P_2,\gamma}$ and $g_{Q_1,Q_2,\gamma'}$ are computational indistinguishable. □

B.2 FSS for Negation and Disjunction of Predicates

In this section, we show some basic FSS constructions for negation and disjunction of predicates. The tensor product construction can be viewed as the conjunction of predicates. Before the FSS construction for negation and disjunction, we first define the notion *anti-symmetric* FSS.

Definition B.2 (Anti-symmetric FSS). *An FSS is anti-symmetric if for $k_0 = k_1$, for all $x \in D_f$,*

$$\mathsf{Eval}(0, k_0, CW, x) - \mathsf{Eval}(1, k_1, CW, x) \neq 0.$$

If the anti-symmetric FSS is for a predicate, then $\mathsf{Eval}(0, k_0, CW, x) - \mathsf{Eval}(1, k_1, CW, x) = 1$ *for $k_0 = k_1$ and* $\mathsf{Eval}(0, k_0, CW, x) - \mathsf{Eval}(1, k_1, CW, x) = 0$ *for $k_0 \neq k_1$.*

Note that the tensor operation does not work for anti-symmetric FSS.

Lemma B.3 (FSS for Negation of Predicates). *Let $\mathcal{P}$ be a family of predicates mapping $\{0,1\}^n$ to $\{0,1\}$. Let $\mathcal{F}_{\mathcal{P}} : \{0,1\}^n \to S \times T$ be the function family induced by $\mathcal{P}$ as*

$$f_{P,\beta} \colon \{0,1\}^n \to S \times T,\ x \mapsto \beta \cdot P(x) = \begin{cases} \beta & \textit{if } P(x) = 1 \\ 0 & \textit{else} \end{cases}.$$

with $P \in \mathcal{P}$ and $\beta \in S \times \{1\} \subseteq S \times T$. Similarly, let $\mathcal{F}_{\neg\mathcal{P}} : \{0,1\}^n \to S \times T$ be the function family induced by $\neg\mathcal{P}$ as

$$f_{\neg P,\beta} \colon \{0,1\}^n \to S \times T,\ x \mapsto \beta \cdot (1 - P(x)) = \begin{cases} \beta & \text{if } P(x) = 0 \\ 0 & \text{else} \end{cases}.$$

Assume there exists a shift-invariant FSS for $\mathcal{F}_{\mathcal{P}}$ over key space $\mathcal{K}$ and correction word space $\mathcal{CW}$.

Then there exists a shift-invariant FSS for $\mathcal{F}_{\neg\mathcal{P}}$ over key space $\mathcal{K} \times (\mathbb{S} \times \mathbb{T})$ and correction word space $\mathcal{CW}$. In particular, if the FSS for $\mathcal{F}_{\mathcal{P}}$ is symmetric, then the FSS for $\mathcal{F}_{\neg\mathcal{P}}$ is anti-symmetric.

The FSS for $\mathcal{F}_{\neg\mathcal{P}}$ can be obtained by adding an extra step to $\mathcal{F}_{\mathcal{P}}$ to switch the between value β and 0. Concretely, assume (β_0, β_1) is a random sharing of β, namely $\beta_0 - \beta_1 = \beta$. Let (k_0, k_1, CW) be the key output by $\mathsf{FSS}^{\mathcal{F}_{\mathcal{P}}}$ for P. Then the key output by $\mathsf{FSS}^{\mathcal{F}_{\neg\mathcal{P}}}$ for $\neg P$ is $((k_0, \beta_0), (k_1, \beta_1), CW)$. The evaluation for $\mathsf{FSS}^{\mathcal{F}_{\neg\mathcal{P}}}$ becomes $\beta_b - \mathsf{Eval}(b, k_b, CW, x)$. It is easy to verify that

$$\begin{aligned} &(\beta_0 - \mathsf{Eval}(0, k_0, CW, x)) - (\beta_1 - \mathsf{Eval}(1, k_1, CW, x)) \\ =&(\beta_0 - \beta_1) - (\mathsf{Eval}(0, k_0, CW, x) - \mathsf{Eval}(1, k_1, CW, x)) \\ =&\beta - \beta \cdot P(x) = \beta \cdot (1 - P(x)). \end{aligned}$$

Hence, the correctness holds. Because (β_0, β_1) is a random sharing of β, the security holds as well.

Lemma B.4 (FSS for Disjunction of Predicates)**.** *Given an anti-symmetric and shift-invariant FSS for $\mathcal{P}_1$, an anti-symmetric and shift-invariant FSS for $\mathcal{P}_2$ and a EOH-PRG, there exists anti-symmetric and shift-invariant FSS construction for $\mathcal{P}_1 \vee \mathcal{P}_2$.*

Proof. Given $P_1 \vee P_2$, the FSS first takes negation to $P_1 \vee P_2$ to obtain $\neg P_1 \wedge \neg P_2$, then run the FSS for tensor products of $\neg P_1$ and $\neg P_2$ and finally take negation to the FSS to obtain an FSS for $P_1 \vee P_2$. □

C FSS for Bit-fixing Predicates from EOH-PRG

In this section, we show function secret sharing (FSS) schemes for bit-fixing predicates from EOH-PRG.

We first construct a symmetric and shift-invariant FSS for one-bit predicate

$$P_\alpha^{\mathsf{BF}} \colon \{0,1\} \to \{0,1\},\ x \mapsto \begin{cases} 1 & \text{if } x = \alpha \text{ or } \alpha = * \\ 0 & \text{else} \end{cases}.$$

For this, we use a naive secret sharing of the truth table. The construction is shown in Figure 4.

Function secret sharing scheme $\mathsf{FSS}^1 = (\mathsf{Gen}^1, \mathsf{Eval}^1)$

Parameters: Let $\mathbb{S}, \mathbb{T}$ be two finite abelian groups such that $S \subseteq \mathbb{S}$.

```
Gen^1(1^λ, α):
1: Sample $s \leftarrow_R S$.
2: Sample $s_0^0, s_0^1, s_1^0, s_1^1 \leftarrow_R \mathbb{S}$ and $t_0^0, t_0^1, t_1^0, t_1^1 \leftarrow_R \mathbb{T}$ according to the following three cases:
3: case $\alpha = *$: $s_0^0 - s_0^1 = s_1^0 - s_1^1 = s$ and $t_0^0 - t_0^1 = t_1^0 - t_1^1 = 1$.
4: case $\alpha = 0$: $s_0^0 - s_0^1 = s$, $s_1^0 - s_1^1 = 0$, and $t_0^0 - t_0^1 = 1$, $t_1^0 - t_1^1 = 0$.
5: case $\alpha = 1$: $s_0^0 - s_0^1 = 0$, $s_1^0 - s_1^1 = s$, and $t_0^0 - t_0^1 = 0$, $t_1^0 - t_1^1 = 1$.
6: Return $k_b := (s_0^b, s_1^b, t_0^b, t_1^b) \in \mathbb{S}^2 \times \mathbb{T}^2$ for $b \in \{0,1\}$.
Eval^1(b, k_b, x):
1: Parse $k_b$ as $k_b = (s_0^b, s_1^b, t_0^b, t_1^b)$.
2: Return $(s_x^b, t_x^b)$.
```

Figure 4: One-bit FSS FSS^1. Superscripts b for s and t represent the party id.

Lemma C.1 (One-bit FSS)**.** *Let $\mathbb{S}, \mathbb{T}$ be two finite abelian groups. Then $(\mathsf{Gen}^1, \mathsf{Eval}^1)$ is a symmetric and shift-invariant FSS over key space $\mathcal{K} := \mathbb{S}^2 \times \mathbb{T}^2$.*

Proof. Security is straightforward as the key k_b for Party P_b is fully uniform. Correctness and the shift-invariant property directly follow from the nature of additive sharing of seed and tag. Symmetry follows from the evaluation. $\square$

Remark C.2. *From the construction, it is obvious that the tag part indicates the output of the one-bit FSS. To keep consistent with the tensor operation, we keep the seed part and the tag part for each bit. Furthermore, it is natural to map the matched output s to a prior value β via an extra operation as in [BGI16b].*

Theorem C.3 (FSS from EOH-PRG)**.** *Assume* $\mathsf{PRG} : \mathbb{S} \to \widetilde{\mathbb{H}}$ *is a EOH-PRG relative to* $(S, \mathbb{T}, (S \times T)^2, (\mathbb{S} \times \mathbb{T})^2)$*.*

There exists a symmetric and shift-invariant FSS for the family of bit-fixing predicates of length ℓ with key space $\mathbb{S}^2 \times \mathbb{T}^2$ and correction word space $\widetilde{\mathbb{H}}^{\ell-1}$. Moreover, the key size is $2(\log|\mathbb{S}| + \log|\mathbb{T}|) + (\ell - 1)\log\left|\widetilde{\mathbb{H}}\right|$.

Proof. Iteratively applying Theorem 5.1 to Lemma C.1, i.e., tensoring the one-bit FSS to itself, $\ell - 2$ times, yields FSS for bit-fixing predicates. The correctness and security follow from Theorem 5.1 and Lemma C.1. The correction word consists of $\ell - 1$ elements from $\widetilde{\mathbb{H}}$ and the key k_b is an element of $\mathbb{S}^2 \times \mathbb{T}^2$. Thus the key size is exactly $2(\log|\mathbb{S}| + \log|\mathbb{T}|) + (\ell - 1)\log\left|\widetilde{\mathbb{H}}\right|$. $\square$

Remark C.4. *Note that the key size is a polynomial of the security parameter and the input length ℓ whereas the naive sharing of the truth table of bit-fixing predicates is of exponential size.*

As presented in Section 8, the EOH-PRG could be instantiated from the LWE assumption or the DCR variant assumption. The comparsions of the resulting FSS schemes from EOH-PRG and FSS for bit-fixing predicates from HSS are shown in Table 2. It is clear that whatever for instantiations from LWE or from DCR, our FSS from EOH-PRG has better key size and running time.

D Supplementary Materials for FSS for Branching Programs

In this section, we show the missing preliminaries, constructions, proofs and remarks of Section 6.

D.1 Branching Programs

We start by recalling the definition of branching programs.

Definition D.1 (Oblivious, layered branching program)**.** *A layered branching program P for a function $g : \{0,1\}^n \to \{0,1\}$ is a directed acyclic graph (V, E), where V is divided into $\ell + 1$ disjoint levels $V_0, V_1, \ldots, V_\ell$ such that every node $u \in V_i, i < \ell$ has out-degree 2, and such that edges only points from nodes in level V_i to nodes in level V_{i+1}. Further, we assume that level i consists of w_i nodes, upper bounded by the width w of the branching program. Further, there are two adjoint functions τ, f such that*

- $V_0 = \{v_0\}$ *contains the only initial node.*
- $V_\ell = \{v_a, v_r\}$ *contains the accept node v_a and the rejection node v_r.*
- $\tau : V \backslash V_\ell \to [n]$ *is the index-to-input map which maps all the non-sink nodes to the input of P.*
- $f : [\ell] \times [w] \times \{0,1\} \to [w]$, $f_i(i, j, \mathbf{x}[\tau(i)]) \mapsto k$ *is the transition function, which maps the j-node in level i to the k-th node in level $i + 1$ upon the input $\mathbf{x}[\tau(i)]$. (Note that here we actually consider maps $f_i \colon [w_i] \times \{0,1\} \to [w_{i+1}]$, since different levels might have a different amount of nodes, but for simplicity we use f as above, where we assume w to be an upper bound on the number of nodes.)*

The branching program is called input-oblivious *if every level i node $v \in V_i$ is mapped to the same input index via the index-to-input map τ.*

Each branching program can be converted to a layered, *input-oblivious* branching program with polynomial blowup in size [Pip79, BGI16a].

D.2 Supplementary Constructions for FSS for Branching Programs

The FSS for P_1 is show in Figure 5. It is a naive sharing of the labels of the nodes in level 1. Each label for the two nodes in level 1 consists of a seed s and a standard basis of $\mathbb{T}^2$ as the tag vector $\mathbf{t}$. The two labels for the two nodes are rearranged according to the transition function for the root node. In the FSS^1, Gen^1 directly shares the rearranged labels and Eval^1 chooses the shared label according to the input $\mathbf{x}[\tau(V_0)]$. The transition function for the root node is hidden in the *rearrangement* of the labels. Correctness and security naturally follow from the random sharing of the reordered labels.

Function secret sharing scheme $\mathsf{FSS}^1 = (\mathsf{Gen}^1, \mathsf{Eval}^1)$ for P_1:

Parameters: Let $\mathbb{S}, \mathbb{T}$ be two finite abelian groups.

```
Gen^1(1^λ, P_1, γ ∈ (S × T^2)^2) :
 1: Parse $\gamma$ as $\gamma =: ((\sigma[1], \mathbf{e}_1), (\sigma[2], \mathbf{e}_2))$ with $\sigma \in S^2$ and $(\mathbf{e}_1, \mathbf{e}_2) \in T^2$ the standard basis of $\mathbb{T}^2$.  ▷ $\sigma$ is the seed value for level 1.
 2: Set $\mathbf{u} \leftarrow (\gamma[f(0,1,0)], \gamma[f(0,1,1)]) \in (S \times T^2)^2$.  ▷ Map 0 and 1 to the corresponding element of level 1 in the array $\gamma$.
 3: Sample $\mathbf{u}_0, \mathbf{u}_1 \leftarrow_R (\mathbb{S} \times \mathbb{T}^2)^2$ such that $\mathbf{u}_0 - \mathbf{u}_1 = \mathbf{u}$.
 4: Return $k_b \leftarrow \mathbf{u}_b$ for $b \in \{0,1\}$.
Eval^1(b, k_b, x) :
 1: Return $k_b[\mathbf{x}[\tau(V_0)]] \in \mathbb{S} \times \mathbb{T}^2$.  ▷ Use input $\mathbf{x}[\tau(V_0)]$ to choose the label for level 1.
```

Figure 5: FSS $(\mathsf{Gen}^1, \mathsf{Eval}^1)$ for P_1. Superscripts for $k, s, \mathbf{t}, v$ represent the party id.

We omit the proof of Lemma 6.1 as it is straightforward.

D.3 Proof of Lemma 6.2

Proof. We first address the correctness of $(\mathsf{Gen}^{i+1}, \mathsf{Eval}^{i+1})$. For correctness, the target is to show that for all $\mathbf{x} \in \{0,1\}^n$,

$$\Pr[\mathsf{Eval}^{i+1}(0, k_0, CW_{i+1}, \mathbf{x}) - \mathsf{Eval}^{i+1}(1, k_1, CW_{i+1}, \mathbf{x}) = \gamma[P_{i+1}(\mathbf{x})] : (k_0, k_1, CW_{i+1}) \leftarrow \mathsf{Gen}^{i+1}(1^\lambda, P_{i+1}, \gamma)] = 1 - \mathsf{negl}(\lambda).$$

From the definition of P_i and P_{i+1}, $P_{i+1}(\mathbf{x}) = f(i, P_i(\mathbf{x}), \mathbf{x}[\tau(V_i)])$. From the correctness of $(\mathsf{Gen}^i, \mathsf{Eval}^i)$, we have that

$$\Pr[\mathsf{Eval}^i(0, k_0, CW_i, \mathbf{x}) - \mathsf{Eval}^i(1, k_1, CW_i, \mathbf{x}) = \beta[P_i(\mathbf{x})] : (k_0, k_1, CW_i) \leftarrow \mathsf{Gen}^i(1^\lambda, P_i, \beta)] = 1 - \mathsf{negl}(\lambda).$$

It is exactly that

$$s_0 - s_1 = \mathbf{s}[P_i(\mathbf{x})] \text{ and } \mathbf{t}_0 - \mathbf{t}_1 = \mathbf{e}_{P_i(\mathbf{x})} \in T^{w_i}.$$

In particular, $\mathbf{t}_0[P_i(\mathbf{x})] - \mathbf{t}_1[P_i(\mathbf{x})] = 1$ and $\mathbf{t}_0[j] - \mathbf{t}_1[j] = 0$ for $j \neq P_i(\mathbf{x})$. It is easy to verify that

$$\sum_{j \in [w_i]} \mathbf{t}_0[j] \cdot CW[j] - \sum_{j \in [w_i]} \mathbf{t}_1[j] \cdot CW[j] = CW[P_i(\mathbf{x})].$$

Recall that $CW[j] \leftarrow \mathsf{PRG}(\mathbf{s}[j]) + \mathsf{Encode}(\mathbf{u}[j])$. From the definition of EOH-PRG, as (s_0, s_1) is a random sharing of $\mathbf{s}[P_i(\mathbf{x})]$, the key (v_0, v_1) for level $i+1$ satisfies

$$\begin{aligned} v_0 - v_1 &= \mathsf{Conv}\left(\sum_{j \in [w_i]} \mathbf{t}_0[j] \cdot CW[j] - \mathsf{PRG}(s_0)\right) - \mathsf{Conv}\left(\sum_{j \in [w_i]} \mathbf{t}_1[j] \cdot CW[j] - \mathsf{PRG}(s_1)\right) \\ &= \mathbf{u}_{P_i(\mathbf{x})} \end{aligned}$$

with overwhelming probability. Thus, (v_0, v_1) is a pseudorandom sharing of $\mathbf{u}_{P_i(\mathbf{x})}$ as each v_b is re-randomized by the PRF. From the Gen^{i+1}, $v_0 - v_1 = \mathbf{u}_{P_i(\mathbf{x})} = (\gamma[f(i, P_i(\mathbf{x}), 0)], \gamma[f(i, P_i(\mathbf{x}), 1)])$. Thus, $(v_0[\mathbf{x}[\tau(V_{i+1})]], v_1[\mathbf{x}[\tau(V_{i+1})]])$ is exactly a share of $\gamma[P_{i+1}(\mathbf{x})] = \gamma[f(i, P_i(\mathbf{x}), \mathbf{x}[\tau(V_{i+1})])]$.

Because FSS^i is shift-invariant, arbitrary random shift to the key (k_0, k_1) will lead to the same value $\beta[P_i(\mathbf{x})]$, i.e., $((s_0, \mathbf{t}_0), (s_1, \mathbf{t}_1))$ is a sharing of $\beta[P_i(\mathbf{x})]$. Hence, FSS^{i+1} is also shift-invariant.

The security follows from the security of FSS^i and the pseudorandomness of PRG. The security can be formally proved as in the security proof of Lemma B.1 via hybrid games. We omit the details here. □

D.4 Topology-Hiding FSS for Branching Programs

It is easy to extend each level to w nodes via adding dummy nodes and then construct an FSS for the extended branching program. Concretely, level 1 and level ℓ inherently have 2 nodes for every out-degree 2 branching program. We only add dummy nodes from level 2 to level $\ell - 1$. (In fact, note that for any out-degree 2 branching programs, the i-th level has at most 2^i nodes, and thus we could only add small number of dummy nodes for the first $\lfloor \log w \rfloor$ levels. For simplicity, here we consider a simpler padding though, where we add dummy nodes to level i for $i \in [2, \ell - 1]$ such that level i has exactly w nodes.)

Formally, we obtain the following theorem.

Theorem D.2 (Topology Hiding FSS for BP from EOH-PRG)**.** *Let P be a branching program with length ℓ and width w. Assume $\mathsf{PRG} : \mathbb{S} \to \widetilde{\mathbb{H}}$ and $\mathsf{PRG}_{\ell-1} : \mathbb{S} \to \widetilde{\mathbb{H}}_{\ell-1}$ are EOH-PRGs.*

Then, there exists a topology-hiding FSS for P over key space $(\mathbb{S} \times \mathbb{T}^2)^2$ and correction word space $\widetilde{\mathbb{H}}^{w(\ell-2)} \times \widetilde{\mathbb{H}}^w_{\ell-1}$. Moreover, the key size is bounded by $2(\log|\mathbb{S}| + 2\log|\mathbb{T}|) + w(\ell-2)\log\left|\widetilde{\mathbb{H}}\right| + w\log\left|\widetilde{\mathbb{H}}_{\ell-1}\right|$.

Note that here dummy nodes are used to hide the level width spectrum of the branching program. Correction words for dummy nodes can be uniformly sampled, since these nodes will never be reached during evaluation. This allows to achieve topology hiding at little extra cost.

E FSS for Approximate Matching Functions and Polynomials

In this section, we present the the FSS for approximate matching functions and polynomials.

E.1 FSS for Approximate Matching Functions

In this section, we show an FSS for approximate matching function using the tag vector technique introduced for branching programs. Concretely the approximate matching function is defined as

$$f_{\mathbf{a},b}(\mathbf{x}) := (\mathsf{dist}(\mathbf{x}, \mathbf{a}) < b)$$

where $\mathbf{a} \in \{0, 1, *\}^\ell, \mathbf{x} \in \{0, 1\}^\ell, b \le \ell$ and $\mathsf{dist}(\mathbf{x}, \mathbf{a}) = \sum_{i\in[\ell]}(\mathbf{a}[i] \neq * \wedge \mathbf{a}[i] \neq \mathbf{x}[i])$.

The approximate matching function first counts the number of unmatched bits and then compare with a threshold value b. It is a distance comparing generalization of the bit-fixing predicate. Consequently, the bit-fixing predicate is the special case $\mathsf{dist}(\mathbf{x}, \mathbf{a}) < 1$.

We construct a branching program for $f_{\mathbf{a},b}$ first and then run the FSS for branching program to build an FSS for $f_{\mathbf{a},b}$. Now we sketch the branching program for $f_{\mathbf{a},b}$. The branching program consists of $\ell + 1$ levels i.e., $\{V_0, V_1, \dots V_\ell\}$, and for $i < \ell$, level i has $i + 1$ nodes, which correspond to the possible $i + 1$ distance values. For level $i < \ell - 1$, assume the distance for the current node is j. If $\mathbf{a}[i] = *$, the distance is still j for level $i + 1$, which implies the both edge 0 and edge 1 point to the distance j node in level $i + 1$. If $\mathbf{a}[i] = 0$, edge 0 points to the distance j node in level $i + 1$ and edge 1 points to the distance $j + 1$ node in level $i + 1$ whereas if $\mathbf{a}[i] = 1$ the two edges for 0 and 1 are exchanged. In the last level, from level $\ell - 1$ to level ℓ, the matching for $\mathbf{a}[\ell]$ and the comparison operation are merged. Therefore level ℓ has only two nodes. If the node in level $\ell - 1$ stands for the distance $\ge b$, whatever $\mathbf{a}[\ell]$ is, the node points to the 0 node in level ℓ. If the node in level $\ell - 1$ stands for the distance $\le b - 1$, whatever $\mathbf{a}[\ell]$ is, the node points to the 1 node in level ℓ. Only if the node in level $\ell - 1$ exactly stands for the distance $b - 1$, the unmatched $\mathbf{a}[\ell]$ will change the distance from $b - 1$ to b, which points to the 0 node in level ℓ. In summary, the branching program has $\ell + 1$ levels. For $i \in [0, \ell - 1]$, level i has $i + 1$ nodes and level ℓ has only two nodes.

With the branching program for $f_{\mathbf{a},b}$, we construct an FSS for it.

Theorem E.1. *Given $\mathbf{a} \in \{0, 1, *\}^\ell$ and $b < \ell$, let $f_{\mathbf{a},b}$ be an approximate matching function. Let $P_{f_{\mathbf{a},b}}$ be the corresponding branching program of $f_{\mathbf{a},b}$ and $\{1, 2, \dots, \ell - 1, \ell, 2\}$ the level width spectrum of $P_{f_{\mathbf{a},b}}$. Assume that for $i \in [1, \ell - 2]$, $\mathsf{PRG}_i : \mathbb{S} \to \widetilde{\mathbb{H}}_i$ relative to $(S, \mathbb{T}, (S \times T^{i+2})^2, (\mathbb{S} \times \mathbb{T}^{i+2})^2)$ and $\mathsf{PRG}_{\ell-1} : \mathbb{S} \to \widetilde{\mathbb{H}}$ relative to $(S, \mathbb{T}, (S \times T^2)^2, (\mathbb{S} \times \mathbb{T}^2)^2)$ are EOH-PRGs.*

There exists an FSS for $f_{\mathbf{a},b}$ over key space $(\mathbb{S} \times \mathbb{T}^2)^2$ and correction word space $\prod_{i\in[1,\ell-2]} \widetilde{\mathbb{H}}_i^{i+1} \times \widetilde{\mathbb{H}}^{\ell}$. Moreover, the key size is bounded by $2\log|\mathbb{S}| + 4\log|\mathbb{T}| + \sum_{i\in[1,\ell-2]}(i+1)\log\left|\widetilde{\mathbb{H}}_i\right| + \ell\log\left|\widetilde{\mathbb{H}}\right|$.

Remark E.2. *Note that although the construction reveals the topology structure of the branching program $P_{f_{\mathbf{a},b}}$, the bit-fixing predicate $\mathbf{a}$ and the threshold value b are still hidden. In fact, for length ℓ predicates, the approximate matching functions share the same topology structure.*

Note that the approximate matching function is a little different from the distributed comparison function(DCF) in [BGI15]. In DCF, the operand to compare is explicitly given as input and the DCF only hides the threshold value b. In our approximate matching function, the value to be compared is a shared result of another FSS, which also explains the reason that DCF enables super-polynomial possible threshold values whereas our approximate matching function only supports polynomially possible threshold values because we need to use a branching program to record the distance of each step and map the distance to the comparison result in the final step.

The FSS for branching program is readily to extend to out-degree q to support q-ary string.

Remark E.3.
- *The distance comparison condition can also be changed to an equality condition, which only changes the operation of the branching program of last level.*
- *For q-ary strings, the distance for each position can be defined for approximate equality, for instance, the two numbers are viewed as equal if the difference is less than some bound B.*

E.2 FSS for Polynomials over a Ring

In this section, FSS schemes for polynomials over a ring are presented and the working ring has polynomial size. Before the FSS constructions for polynomials, we recall some definitions for polynomials over a ring.

Definition E.4 (Total Degree)**.** *Let $k, w \in \mathbb{N}$. Let R be a ring with w elements. For a monomial $\mathbf{x}^{\mathbf{d}} = x_1^{d_1} \cdot x_2^{d_2} \ldots x_k^{d_k}$, the total degree of $\mathbf{x}^{\mathbf{d}}$ is defined as $d_1 + d_2 + \ldots d_k$. The total degree of a polynomial $P(\mathbf{x}) = \sum_i c_i \mathbf{x}^{\mathbf{d}_i}$ with $c_i \in R$ is the maximal total degree of monomials with $c_i \neq 0$.*

Given a polynomial $P(\mathbf{x}) = \sum_{i\in[m]} c_i \mathbf{x}^{\mathbf{d}_i}$, the FSS securely shares each monomial $c_i \mathbf{x}^{\mathbf{d}_i}$ and sums up all the local shares of the monomials after evaluation. The FSS for a monomial is built upon the branching program induced by the monomial. Given $c\mathbf{x}^{\mathbf{d}}$, the first level of the FSS corresponds to $cx_1^{d_1}$, the second level $x_2^{d_2}$, and the i-th level $x_i^{d_i}$, etc. Given an input $\mathbf{y} \in R^k$, the shared value of the FSS for $P(\mathbf{y})$ corresponds to the seed and tag for $P(\mathbf{y})$ in level k. Assume the ring R has a set of element $\{0, 1, \ldots, w-1\}$. Then $\langle \mathbf{t}_{P(\mathbf{y})}, (0, 1, \ldots, w-1)\rangle$ is exactly the value $P(\mathbf{y})$. After the evaluation of each monomial, the two-party locally sums up the shares for the monomials.

We briefly sketch the branching program from a monomial.The monomial $c\mathbf{x}^{\mathbf{d}}$ can be viewed as an out-degree w branching program with k levels and w nodes in each level. The level 1 nodes are rearranged according to $c \cdot x_1^{d_1}$, i.e., the order of the elements $(c \cdot 0^{d_1}, c \cdot 1^{d_1} \ldots c \cdot (w-1)^{d_1})$. Inductively, for a node j in level i, assuming v_j being the corresponding value of node j in ring R, the edges to next level from node j is rearranged according to $v_j \cdot x_{i+1}^{d_{i+1}}$, i.e., $(v_j \cdot 0^{d_{i+1}}, v_j \cdot 1^{d_{i+1}} \ldots v_j \cdot (w-1)^{d_{i+1}})$.

Thus, we have an FSS for a monomial.

Lemma E.5 (FSS for a Monomial)**.** *Let $k, w \in \mathbb{N}$. Assume R is a ring with w elements. Let $\mathsf{PRG} : \mathbb{S} \to \widetilde{\mathbb{H}}$ be a EOH-PRG relative to $(S, \mathbb{T}, (S \times T^w)^w, (\mathbb{S} \times \mathbb{T}^w)^w)$. Let $c \cdot \mathbf{x}^{\mathbf{d}}$ be a monomial defined over ring R and k variables.*

There exists an FSS for $c \cdot \mathbf{x}^{\mathbf{d}}$ over key space $(\mathbb{S} \times \mathbb{T}^w)^w$ and correction word space $\widetilde{\mathbb{H}}^{(k-1)w}$ with key size $w(\log|\mathbb{S}| + w\log|\mathbb{T}|) + (k-1)w\log\left|\widetilde{\mathbb{H}}\right|$.

Remark E.6. *Note that the coefficient c is embedded into level 1 of the induced branching program and thus is hidden by the FSS. Similarly, the degree d_i is embedded into level i of the branching program and thus is hidden by the FSS. For $d_i = 0$, x_i does not appear in $c \cdot \mathbf{x}^{\mathbf{d}}$, the branching program directly multiplies the previous value by 1 to hide this d_i as $x_i^{d_i}$ is exactly 1.*

Remark E.7. *Note that there is no real multiplication happened during the FSS evaluation. Each multiplication is essentially a move from one node to another node in the induced branching program. For the same reason, the multiplication over the ring R can be non-commutative.*

Remark E.8. *Different from the t-CNF/t-DNF cases, the FSS for polynomials of* constant *total degree can be achieved via directly sharing the coefficient of each possible monomial. It is an* information-theoretic *FSS.*

Theorem E.9 (FSS for Polynomials)**.** *Let $k, w \in \mathbb{N}$. Assume R is a ring with w elements. Let* $\mathsf{PRG} : \mathbb{S} \to \widetilde{\mathbb{H}}$ *be a EOH-PRG relative to $(S, \mathbb{T}, (S \times T^w)^w, (\mathbb{S} \times \mathbb{T}^w)^w)$. Let $P(\mathbf{x})$ be a polynomial with m monomials defined over R and k variables.*

There exists a shift-invariant FSS for $P(\mathbf{x})$ over key space $(\mathbb{S} \times \mathbb{T}^w)^{mw}$ and correction word space $\widetilde{\mathbb{H}}^{mw(k-1)}$ with key size $mw(k-1)\log\left|\widetilde{\mathbb{H}}\right| + mw(\log|\mathbb{S}| + w\log|\mathbb{T}|)$.

The FSS construction is obtained via sharing each monomial according to Lemma E.5 and the Eval sums up all the tag values.

Remark E.10. *The FSS for polynomials $(c_1x_1^{d_1} + c_2x_2^{d_2} + \cdots + c_kx_k^{d_k})^d$ of this form is not captured by Theorem E.9. By following the same idea as in Lemma E.5, the FSS for $(c_1x_1^{d_1} + c_2x_2^{d_2} + \cdots + c_kx_k^{d_k})^d$ is straightforward to obtain, which computes the power d in the last level. The FSS for this type of polynomials can be combined with the FSS in Theorem E.9.*

F Supplementary Materials for EOH-PRG Instantiated from LWE or DCR

We show the missing preliminaries, proofs and remarks of Section 8.

F.1 EOH-PRG from LWE

Before presenting the proof of EOH-PRG instantiated from LWE, we recall some basic results from [BKS19].

Lemma F.1 (Distributed Rounding)**.** *[BKS19, Lemma 1] Let $p, q, B \in \mathbb{N}$ with $p|q$ and $2Bp\lambda^{\omega(1)} \leq q$. Assume*

$$s = \frac{q}{p}m + e \mod q$$

for some $m \in \mathbb{Z}_p$ with $|e|_\infty \leq B$. Assume $s = s_0 - s_1$ for random $s_0, s_1 \in \mathbb{Z}_q$. Then

$$Pr\left[m = \lceil s \rfloor_{q\to p} = \lceil s_0 \rfloor_{q\to p} - \lceil s_1 \rfloor_{q\to p} \mod p\right] \geq 1 - \frac{2Bp}{q} \geq 1 - \frac{1}{\lambda^{\omega(1)}}.$$

The probability is over the randomness of s_0 and s_1.

Lemma F.2 (Lifting)**.** *[BKS19, Lemma 2] Let $B, p \in \mathbb{N}$ such that $B\lambda^{\omega(1)} \leq p$. Given $z \in \mathbb{Z}$ with $|z|_\infty \leq B$ and z being additively shared as $z = z_0 - z_1 \mod p$, then*

$$\Pr\left[z = z_0 - z_1 \textit{ over } \mathbb{Z}\right] \geq 1 - \frac{B}{p} \geq 1 - \frac{1}{\lambda^{\omega(1)}},$$

where the probability is taken over the randomness of z_0, z_1.

Moreover, for any $q \in \mathbb{N}$, $\Pr\left[z = z_0 - z_1 \pmod q\right] \geq 1 - \frac{1}{\lambda^{\omega(1)}}$. Note that here q is completely independent of B, p.

With the two lemmas, we show the proof of the instantiation from LWE.

Proof of Theorem 8.1. We first verify that $\cdot : \mathbb{Z}_p \times \mathbb{Z}_p^{\ell(n+w)} \to \mathbb{Z}_p^{\ell(n+w)}$ is indeed a homomorphic group operation. $\forall t_0, t_1 \in \mathbb{Z}_p$ and $\forall h \in \mathbb{Z}_p^{\ell(n+w)}$, $t_0 \cdot h + t_1 \cdot h = (t_0 + t_1) \cdot h$ as $\widetilde{\mathbb{H}} = \mathbb{Z}_p^{\ell(n+w)}$ is an additive group of order p. That PRG is a secure PRG relative to $S = \{0,1\}^n$ follows from the assumption of $\mathsf{LWE}_{n,\ell(n+w),q}$ from Lemma A.3. Next, we prove PRG is a EOH-PRG relative to $(S, \mathbb{T}, H, \mathbb{H})$. For all $m \in H$, for $s \leftarrow_R S$, let $y := \mathsf{PRG}(s) + \mathsf{Encode}(m)$. Assume y is uniformly randomly shared as (y_0, y_1) and s is uniformly randomly shared as (s_0, s_1), i.e., $y_0 \leftarrow_R \widetilde{\mathbb{H}}$, $s_0 \leftarrow_R \mathbb{S}$, $y_1 := y_0 - y$ over $\widetilde{\mathbb{H}}$, $s_1 = s_0 - s$ over $\mathbb{S}$.

The target is to prove that $\mathsf{Conv}(y_0 - \mathsf{PRG}(s_0)) - \mathsf{Conv}(y_1 - \mathsf{PRG}(s_1)) = m \mod p$ with overwhelming probability. Thus,

$$\begin{aligned}
&\mathsf{Conv}(y_0 - \mathsf{PRG}(s_0)) - \mathsf{Conv}(y_1 - \mathsf{PRG}(s_1))\\
=&\lceil y_0 - \mathsf{PRG}(s_0)\rfloor_{p\to r} - \lceil y_1 - \mathsf{PRG}(s_1)\rfloor_{p\to r}\\
=&\lceil y + y_1 - \mathsf{PRG}(s_0)\rfloor_{p\to r} - \lceil y_1 - \mathsf{PRG}(s_1)\rfloor_{p\to r}\\
=&\lceil \mathsf{PRG}(s) + \mathsf{Encode}(m) + y_1 - \mathsf{PRG}(s_0)\rfloor_{p\to r} - \lceil y_1 - \mathsf{PRG}(s_1)\rfloor_{p\to r}\\
=&\lceil \mathsf{Encode}(m) + e + y_1 - \mathsf{PRG}(s_1)\rfloor_{p\to r} - \lceil y_1 - \mathsf{PRG}(s_1)\rfloor_{p\to r}\\
=&m \mod r
\end{aligned}$$

where $|e|_\infty \leq 1$. The last two steps follow the AH-PRG property of PRG and the distributed rounding lemma, respectively. Let $m_0 := \mathsf{Conv}(y_0 - \mathsf{PRG}(s_0)), m_1 := \mathsf{Conv}(y_0 - \mathsf{PRG}(s_0))$. Then $m_0 - m_1 = m$ over $\mathbb{Z}$ with overwhelming probability from the lifting lemma and thus $m_0 - m_1 = m \mod p$. □

For the EOH-PRG instantiation from Ring-LWR, we show the following Ring-LWR assumption.

Definition F.3 (Assumption)**.** *Let $n = n(\lambda), p = p(\lambda), q = q(\lambda), r = r(\lambda), B = B(\lambda), m = \ell + \lceil \frac{\ell w}{n} \rceil \in \mathbb{N}$ such that $r|p, p|q, 2\lambda^{\omega(1)} \leq r$, $2Br\lambda^{\omega(1)} \leq p$ and $n \log q \leq \ell(n+w)\log p$.*

Assume the binary secret $\mathsf{Ring\text{-}LWR}_{\mathcal{R},m,q,p}$ is pseudorandom, i.e., $(\mathbf{a}, \lceil \mathbf{s}\cdot\mathbf{a}\rfloor_{q\to p})$ is pseudorandom, where $\mathbf{s} \leftarrow_R \{0,1\}^n, \mathbf{a} \leftarrow_R \mathcal{R}_q^m$.

Note the function $\psi(\cdot)$ is additively homomorphic as it only operates on coefficients. Therefore the PRG is still a AH-PRG as only the rounding operation leads to errors and the other operations are homomorphic. The proof of Theorem 8.2 is very similar to the proof of Theorem 8.1. We omit the details here.

A few remarks follow.

Remark F.4.

- *It is possible to sample each entry of S from a larger domain rather than $\{0,1\}$. The only consequence is to adjust the parameter (p,r) such that the distributed rounding lemma and lifting lemma hold.*
- *The EOH-PRG can be reused for concrete applications like FSS for bit-fixing predicates, for branching programs and for DFAs.*
- *For instantiatation from Ring-LWE, the computation maybe become more efficient because of the NTT optimization.*
- *In the parameter setting, we require $n\log q \leq \ell(n+w)\log p$ rather than $p\lambda^{\omega(1)} \leq q$ as the HSS instantiated from LWE [BKS19].*
- *Although the PRG is performed over $\mathbb{Z}_q$ and the LWE assumption is over $\mathbb{Z}_q$, the FSS key is still shared over $\mathbb{Z}_p$.*

F.2 EOH-PRG from DCR

Before showing the proof for EOH-PRG from DCR assumption, we first recall some basic results for the DCR assumption.

Lemma F.5 (Distributed DLog)**.** *[OSY21, Lemma 3.3] Let $N = PQ$ be an RSA modulus. Assume z_0, z_1 are sampled from $\mathbb{Z}_{N^2}$ such that $\frac{z_0}{z_1} = (1+N)^x \mod N^2$ for some $x \in \mathbb{Z}_N$. Then there exists a PPT $\mathsf{DDLog} : \mathbb{Z}_{N^2}^* \to \mathbb{Z}_N$ such that*

$$\mathsf{DDLog}(z_0) - \mathsf{DDLog}(z_1) = x \mod N.$$

Moreover, if $|x| \leq \frac{N}{2^\lambda}$, then

$$\mathsf{DDLog}(z_0) - \mathsf{DDLog}(z_1) = x \text{ over } \mathbb{Z}$$

with probability at least $1 - \frac{1}{2^\lambda}$, where the probability is over the randomness of z_0, z_1.

Definition F.6 (DCR Variant Assumption). *[BG10, Lemma B.1] Let $\mu \in \mathbb{N}$. Assume $\mathbf{g} := (g_0, \ldots, g_\mu) \in \mathbb{Z}_{N^2}^{\mu}$ and every entry of $\mathbf{g}$ is uniformly sampled from the N-th residue group $\mod N^2$. For a random $r \in [N]$, $(g_0^r \ldots g_\mu^r) \in \mathbb{Z}_{N^2}^{\mu}$ is pseudorandom if the DCR assumption over $\mathbb{Z}_{N^2}$ holds.*

In particular, we use the assumption that for $r \in [\frac{N}{2^\lambda}]$, $\mathbf{g}^r$ is still pseudorandom for any PPT adversary.

It is pointed out in [ADOS22, Section 4.1], the DCR variant assumption is sound if the domain of the small exponent is exponentially large. This kind of low exponent assumption dates back to [KK04] and was also used in [BCG+17].

Remark F.7. *From the DCR variant assumption, the PRG maps additive group to multiplicative group. With the* DDLog *operation, the multiplicative group is converted back to the additive group again.*

The proof of Theorem 8.3 is very similar to the proof of Theorem 8.1. As the group $\widetilde{\mathbb{H}}$ is multiplicative group, we briefly show the techniques.

Proof. We first verify that $\cdot : \mathbb{Z}_{\phi(N^2)} \times (\mathbb{Z}_{N^2}^*)^{\ell(1+w)} \to (\mathbb{Z}_{N^2}^*)^{\ell(1+w)}$ is indeed a homomorphic group operation. $\forall t_0, t_1 \in \mathbb{Z}_{\phi(N^2)}$ and $\forall h \in (\mathbb{Z}_{N^2}^*)^{\ell(1+w)}$, it holds that $h^{t_0} \times h^{t_1} = h^{t_0+t_1}$ as each $\mathbb{Z}_{N^2}^*$ is exactly a multiplicative group of order $\phi(N^2)$. PRG being a secure PRG relative to $S = [-B/2, B/2]$ follows from the DCR variant assumption. Next, we prove PRG is a EOH-PRG relative to $(S, \mathbb{T}, H, \mathbb{H})$. For all $m \in H^\ell$, for $s \leftarrow_R S$, let $y := \mathsf{PRG}(s) \times \mathsf{Encode}(m)$. Assume $y_0 \leftarrow_R \widetilde{\mathbb{H}}, s_0 \leftarrow_R \mathbb{S}$, $y_1 := y_0 \div y, s_1 = s_0 - s$. The target is to prove that $\mathsf{Conv}(y_0 \div \mathsf{PRG}(s_0) - \mathsf{Conv}(y_1 \div \mathsf{PRG}(s_1)) = m \mod \phi(N^2)$ with overwhelming probability. Thus,

$$
\begin{aligned}
&\mathsf{Conv}(y_0 \div \mathsf{PRG}(s_0)) - \mathsf{Conv}(y_1 \div \mathsf{PRG}(s_1)) \\
=&\mathsf{DDLog}(y_0 \div \mathsf{PRG}(s_0)) - \mathsf{DDLog}(y_1 \div \mathsf{PRG}(s_1)) \\
=&\mathsf{DDLog}(y_1 \times y \div \mathsf{PRG}(s + s_1)) - \mathsf{DDLog}(y_1 \div \mathsf{PRG}(s_1)) \\
=&\mathsf{DDLog}(y_1 \times \mathsf{Encode}(m) \div \mathsf{PRG}(s_1)) - \mathsf{DDLog}(y_1 \div \mathsf{PRG}(s_1)) \\
=&m.
\end{aligned}
$$

The last two steps follow from the lifting operation and the distributed DLog lemma, respectively. Let $m_0 := \mathsf{Conv}(y_0 \div \mathsf{PRG}(s_0)), m_1 := \mathsf{Conv}(y_0 \div \mathsf{PRG}(s_0))$. Then $m_0 - m_1 = m$ over $\mathbb{Z}$ with overwhelming probability from the lifting operation and thus $m_0 - m_1 = m \mod \phi(N^2)$. □

A few remarks follow.

Remark F.8. *Note that for LWE or DCR instantiations, (m_0, m_1) output by* Conv *is not a pseudorandom sharing of m over $\mathbb{Z}_p$. In our FSS construction for tensor products, for branching programs, and for DFAs, the* Conv *output is re-randomized by a PRF.*

Remark F.9 (KDM Security). *The FSS constructions for DFAs in Section 7 rely on the KDM security of EOH-PRG. It is straightforward to prove the pseudorandomness for LWE or DCR following the method to prove the KDM security in [ACPS09, Theorem 6] or [BHHO08, Section 3.2]. We briefly shows the steps to prove the KDM security of* PRG *for LWE and DCR, repsectively.*

LWE : *Suppose s_i is used in the encoding part. The goal is to prove if there exists an adversary $\mathcal{A}$ breaking the pseudorandnomness of $(A, \lceil \mathbf{s}A \rfloor_{q \to p} + \frac{p}{r}(0 \ldots s_i \ldots 0))$, then there exists an adversary $\mathcal{B}$ breaking the pseudorandomness of* PRG. *Given an instance $(A, \lceil \mathbf{s}A \rfloor_{q \to p})$ of* PRG, *$\mathcal{B}$ transforms it to $(A - \frac{q}{r}B, \lceil \mathbf{s}A \rfloor_{q \to p})$ where B is a public matrix mapping $\mathbf{s}$ to the vector $(0 \ldots s_i \ldots 0)$, and then feeds it to $\mathcal{A}$. It is easy to verify $(A - \frac{q}{r}B, \lceil \mathbf{s}A \rfloor_{q \to p})$ is indeed an instance of the KDM security game. $\mathcal{B}$ uses the advantage of $\mathcal{A}$ to distinguish* PRG *from uniform distribution.*

DCR : *According to the DCR assumption, the distribution $(\mathbf{g}, \mathbf{g}^s)$ is computationally indistinguishable from the distribution $(\mathbf{h}, \mathbf{h}^s)$, where $\mathbf{g} := (g_1 \ldots g_\mu), \mathbf{h} := (h_1 \ldots h_i/(1+N) \ldots, h_\mu)$, and each entry of $\{g_i, h_i\}_{i \in [\mu]}$ is sampled from the N-th residue group. Given an instance of EOH-PRG instantiated from DCR, one is able to convert it to an instance of the KDM instance. If there exists an adversary breaking the KDM security game, then one is able to use that adversary distinguishing the EOH-PRG instantiated from DCR and the uniform distribution.*

G FSS Applications

In this section, we present some applications of the FSS for tensor operation, bit-fixing predicates, branching programs, DFAs and more. The general framework follows the model of the 2-server PIR applications of FSS [BGI16a]. There are two roles in the applications, clients and 2-server. Each server holds a replication of a database DB with N items. A client starts a query to the database while keeping the query hidden from the 2-server. The protocol with FSS works as follows. First, a client splits a private query into shares via FSS and sends each share to the corresponding server. Next, the servers use the FSS share to run the FSS evaluation and sends the result to the client. Finally, the client combines the responses from the two servers to get the final result of the query. Servers learn nothing on the query as long as the two servers do not collude. Note that only in the private nearest neighbour search(Section G.3), the server privacy is required. As our main intention was to show the direct applicability of our FSS constructions in various application scenarios, we do not focus on server privacy for the most part.

Lots of online sites provide query services to customers, for instance online shopping sites and travel sites. However, the query maybe reveals the privacy information of customers. The service provider is able to collect the user information and acts immorally or maliciously via making use of the collected information. For example, travel sites count the frequencies of searched flights and increase the prices for frequently searched flights. Splinter [WYG$^+$17] is a system designed to protect the query privacy and provides a subset of SQL queries capturing most useful applications with good performance. Splinter employs FSS to split each query into shares and sends the shares to the servers, respectively. Non-colluding servers are unable to obtain useful information of the query. Splinter provides $\{\mathsf{sum}, \mathsf{count}, \mathsf{min}, \mathsf{max}, \mathsf{top\text{-}k}\}$ queries for various conditions.

As pointed out in Section H, for bit-fixing predicates and Ring-LWE instantiation, the direct FSS constructions have key size reduced by a factor around 4 and comparable running time comparing with the previous FSS constructions from HSS. For bit-fixing preidates and DCR instantiation, the direct FSS constructions have key size reduced by a factor of 3.5 and running time reduced by a factor of 1.75 comparing with the previous FSS constructions from HSS.

The direct FSS constructions for branching programs avoid the heavy universal branching program transformation. Concretely, assuming n is the secret length of LWE instance and w the width of the branching program. For branching programs and Ring-LWE instantation, the direct FSS constructions have key size reduced from a factor $2w \times n$ to $w + n$ and running time reduced from a factor $8w^2$ to $2 + \lceil \frac{2w}{n} \rceil$ comparing with the existing FSS for branching programs from HSS. For branching programs and DCR instantiation, the direct FSS constructions have key size around 0.28 of the key size of the FSS from HSS and running time reduced from a factor $14w^2$ to $3w + 2$. Hence, the resulting applications have smaller key size and better running time.

It is worth to mention that the FSS for DFA is the first that allows key size independent of the length of the input(except for the generic constructions from FHE).

G.1 Private Image Matching on Public Data

It is mentioned that image matching is not covered by Splinter. Privacy-preserving image matching protocol has many critical applications, including face detection [AB06], logo patent search, patient CT image retrieval [SKSJ08a] as these types of query images contain sensitive information and thus leakage of the query images leads to severe impact. We use our FSS from EOH-PRG to design a protocol for image matching supporting $\{\mathsf{sum}, \mathsf{count}, \mathsf{min}, \mathsf{top\text{-}k}\}$ queries.

There are many image matching algorithms such as SIFT, SURF, BRIEF, ORB, etc [MJF$^+$21]. Each of the image matching algorithms consists of three steps, key point detection, feature descriptor construction, and feature matching. The BRIEF [CLSF10] and ORB [RRKB11] use Hamming distance to perform feature matching over binary strings. Actually, there are algorithms to convert the float point SIFT or SURF descriptors to binary string with Hamming distance to speed up the matching algorithm [CLSF10]. The FSS for approximate matching function for Hamming distance from Section E.1 can be exactly used here to do the image matching.

The private image matching algorithm works as follows. Before the protocol, the two servers preprocess the image database to obtain the binary feature descriptors for each image according to the corresponding matching algorithm. Given a query image, a client extracts the binary feature descriptor for the query image. Next, the client generates the FSS keys for the query image according to the FSS for approximate matching function and sends the FSS keys to the two servers, respectively. Next, the two servers run the FSS evaluation algorithm to obtain the share of the response and send the share to the client. Finally, the client receives the responses from the two servers to obtain the matching results. Next, the client and the two servers run the FSS for approximate matching function following the aforementioned framework to perform the query.

That the binary descriptor of the query image is hidden follows the security of the FSS scheme. Thus, the secrecy of the query image follows. Let ℓ be the length of binary feature descriptor. The computational complexity of the FSS evaluation is $O(\ell N)$ PRG evaluations and the communication cost for the FSS share scales with $O(\ell^2)$ and the cost for the FSS response is $\log N$ as it only contains the ID of the fetched image.

Most existing secure privacy-preserving image matching protocols [JB22] rely on the searchable encryption [SWP00] or homomorphic encryption [Pai99] to perform computation on encrypted images, which incur huge computation cost. Recently a private approximate membership computation protocol with perceptual hash matching was proposed in [KM21], which can also be used to perform image matching. However, the approximate membership computation protocol heavily relies on fully homomorphic encryption (FHE). It is worth to mention that PIR is used in [SKSJ08b] to protect the privacy of the query image. However, the similarity computation is not supported in the protocol with PIR.

The protocol naturally integrates with the Splinter system to enable $\{\mathsf{sum}, \mathsf{count}, \mathsf{min}, \mathsf{top}\text{-}\mathsf{k}\}$ query for images. The $\{\mathsf{count}, \mathsf{sum}\}$ query only takes one round whereas the $\{\mathsf{min}, \mathsf{top}\text{-}\mathsf{k}\}$ query takes $\log \ell$ rounds of communication.

G.2 Private Partial Text Matching

In private partial text matching protocol, the two servers hold a database of strings and a client would like to run the fuzzy pattern matching without leaking the query pattern. It was also pointed out that Splinter [WYG+17] does not support partial text matching. The like operator in SQL is a typical case of the partial text matching. Although there are many optimizations for the like operator in real SQL engines for instance MySQL [MyS], DFA is still used in partial text matching for not indexed fields. DFA is also used in the pattern matching in Tcl programming language library [Tcl](Henry Spencer's implementation) and in the filename glob syntax [glo]. Private DFA evaluation has many interesting applications, for example private searching on DNA sequences, pharmaceutical databases and malware detection. With the FSS for DFA in Section 7, it is natural to support the like operator or DFA evaluations. Furthermore, the FSS for DFA can be used to support regular expressions as DFAs recognize regular languages.

The private partial text matching protocol works as follows. Given a pattern string, a client first creates a DFA for the pattern string. Next, the client and the the two servers run the FSS for DFAs following the above-mentioned framework to execute the matching.

Remark G.1. *Following the security of the FSS scheme, the pattern string is hidden. However, the FSS construction reveals the number of states of the DFA which presumably reveals some information on the regular expression. This could be fixed by adding some dummy states at the expense of inflated key size and running time as in the FSS for branching programs.*

Previous protocols employ oblivious automation evaluation both hiding the DFA and the input text. For our FSS, we only care the DFA secrecy. However, the communication complexity of existing oblivious automation evaluation suffers an extra factor of the input text length whatever the constructions relying on garbled circuits and oblivious transfer [MNSS12], homomorphic encryption [GHS16] with several rounds of interactions, or conditional disclosure of secrets [PAM22]. The keysize of the FSS for is independent of the input length.

The protocol can also be integrated with the system Splinter [WYG+17] to enable $\{\mathsf{sum}, \mathsf{count}\}$ query for fuzzy pattern.

G.3 Nearest Neighbour Search

In private nearest neighbour search protocol, the two servers hold a database of feature vectors and a client wants to find the nearest neighbour to the query feature based on some metrics, e.g., Euclidean, Hamming, ℓ_1, without revealing information on the query feature vector. For nearest neighbour search, database privacy is also required, which says the client learns nothing on the database beyond the query answer after the interactions.

The private nearest neighbour search(NNS) protocol has many applications such as online music recommendation [CZZM07], image search [KG09], face recognition [EFG+09], biometric identification [HMEK11], image recognition especially for handwriting recognition [LBBH98], clustering in NLP [RPH05], patient genomic or data search [AHLR18] and so on.

There are many secure privacy-preserving protocols for nearest neighbour search, which behaves with high running time and large bandwidth. Most of the existing protocols rely on two-party secure computation or fully homomorphic encryptions. For instance, the protocol SANNS [CCD+20] uses oblivious RAM, garbled circuits and homomorphic encryptions. In the recent more efficient protocol [SLD22], locality sensitive hashing(LSH) is heavily involved, which incurs some accuracy loss. To achieve a good accuracy, say $> 95\%$, $O(\sqrt{N})$ number of queries to the database is necessary, which leads to an extra $O(\sqrt{N})$ factor to the communication cost.

For ease of exposition, assume there exists only one nearest neighbour for the query vector. Assume the query vector has dimension d, namely from $\{0,1\}^d$ and the protocol only returns the identifier(ID) of the nearest neighbour. Our privacy-preseving protcol for NNS only supports Hamming distance and the ℓ_1 distance as the ℓ_1 distance can be embedded into Hamming space with a small distortion [LLR94, IM98]. The Hamming distance based NSS is used in iamge search, biometric identification, clustering in NLP, patient genomic data search and so on. It is easy to observe that Euclidean distance computation is captured by NC^1. To naturally and efficiently support Euclidean distance, it requires further future work.

Our protocol follows from the framework of the protocol in [SLD22] without relying on LSH and behaves with improved communication cost. In fact, both LSH and DPF are replaced by the approximate matching function in our protocol. For a query vector $\mathbf{q}$, a client sends out d queries for the distance $\{1, 2 \dots d\}$, respectively via the FSS for approximate matching function from Section E.1. The two servers run the FSS evaluation algorithm to obtain the result for each distance. Assume the nearest neighbour vector has distance z with identifier id. Note that the two servers obtain a sharing of the vector $\mathbf{v} := [0 \dots 0, \mathsf{id}, \dots]$, where id appears exactly in the z-th position of $\mathbf{v}$. To suppress the information leakage from the answer vector $\mathbf{v}$, oblivious masking is employed [SLD22, Section 5.1], which maps each $\mathbf{v}[i]$ for $i < z$ to 0, to a uniformly random element for $i > z$, and keeps the $\mathbf{v}[z]$. The full protocol is shown in Figure 6. The correctness of the protocol is easy to verify. The client privacy follows from the

Private Nearest Neighbour Search Protocol

Server input: $\mathsf{DB} = (\mathsf{DB}[1], \dots, \mathsf{DB}[N])$
Client input: query $\mathbf{q}$
Preprocessing(server computation):

```
1: Output a common random string $K$ and a PRF.
```

Step 1(client computation):

```
1: for $i \in \{1 \dots d\}$ do
2:     $(k_{0,i}, k_{1,i}) \leftarrow \mathsf{FSS.Gen}(\mathbf{q}, i)$        ▷ The FSS for the exact distance $i$.
3: end for
4: Let $\mathbf{k}_b \leftarrow (k_{b,1} \dots k_{b,d})$
5: Send $\mathbf{k}_b$ to server $b$, respectively.
```

Step 2(server b computation):

```
1: Let $\mathbf{ID}_b, \widetilde{\mathbf{ID}}_b$ be two arrays of length $d$.
2: for $i \in \{1 \dots d\}$ do
3:     $\mathbf{ID}_b[i] \leftarrow \sum_{j \in [N]} \mathsf{DB}[j].\mathsf{id} \cdot \mathsf{FSS.Eval}(b, \mathbf{k}_b[i], \mathsf{DB}[j])$
4:     $r_i \leftarrow \mathsf{PRG}(K, i)$
5:     $\widetilde{\mathbf{ID}}_b[i] \leftarrow \mathbf{ID}_b[i] + r_i \cdot \sum_{j \in [i-1]} \mathbf{ID}_b[j]$        ▷ Oblivious masking
6: end for
7: Send $\widetilde{\mathbf{ID}}_b$ to client.
```

Step 3(client computation):

```
1: Receive $\widetilde{\mathbf{ID}}_b$ from Server $b$.
2: Let $\widetilde{\mathbf{ID}} := \widetilde{\mathbf{ID}}_0 + \widetilde{\mathbf{ID}}_1$
3: Output the first non-zero entry of $\widetilde{\mathbf{ID}}$.
```

Figure 6: Private nearest neighbour search protocol involving FSS for approximate matching function and oblivious masking.

security of the FSS scheme and the server privacy follows from the in-depth analysis of [SLD22, Section

7.2], which achieves asymptotically *optimal* leakage. The communication cost from the client has a d factor for FSS key as there are d FSS keys and the communication cost from the servers is $O(d \log N)$ as each masked id is of size $O(\log N)$. The computational cost for the server has a $d \cdot N$ factor of a single FSS evaluation and the computation cost for oblivious masking is $O(d)$. Note that the computation cost almost matches the computational cost of the protocol in [SLD22] with LSH and the communication cost is significantly improved over the $\sqrt{N}$ factor especially for $d \in \mathsf{polylog}(N)$ as in [SLD22, Table 6].

G.4 FSS for t-CNF/t-DNF and CNF/DNF

For the ease of exposition, we mainly show the FSS constructions for exact t-CNF formulae. Recall that in exact t-CNF formula, each clause contains exactly t literals. An FSS for t-CNF can be conjuncted from FSS for exact j-CNF for $j \in [t]$.

Assume the t-CNF/t-DNF formula through this section has k variables and m clauses.

Theorem G.2 (FSS for exact t-CNF)**.** *Assume* $\mathsf{PRG} : \mathbb{S} \to \widetilde{\mathbb{H}}$ *is a EOH-PRG relative to* $(S, \mathbb{T}, (S \times T)^2, (\mathbb{S} \times \mathbb{T})^2)$.

There exists a symmetric and shift-invariant FSS for the family of exact t-CNF formulae with key space $\mathbb{S}^2 \times \mathbb{T}^2$ *and correction word space* $\widetilde{\mathbb{H}}^{2^t \binom{k}{t} - 1}$.

To derive the FSS scheme, the given exact t-CNF formula φ is first transformed to an equivalent bit-fixing predicate $\mathbf{a}$ of length $2^t \cdot \binom{k}{t}$ and then the FSS scheme for the bit-fixing predicate $\mathbf{a}$ leads to an FSS scheme for φ. The construction is presented in Figure 7. Let f be a public canonical function that maps the clause of exactly t literals out of k variables to a number in range $\{1 \dots 2^t \cdot \binom{k}{t}\}$.

Function secret sharing scheme $\mathsf{FSS}^{t\text{-CNF}}$ for exact t-CNF:

Parameters: Let $\mathsf{PRG} : \mathbb{S} \to \widetilde{\mathbb{H}}$ be a EOH-PRG relative to $(S, \mathbb{T}, (S \times T)^2, (\mathbb{S} \times \mathbb{T})^2)$. Let $K = 2^t \cdot \binom{k}{t}$.

```
Gen^{t-CNF}(1^λ, φ) :
1: Construct a bit-fixing predicate $\mathbf{a}$ of length $K$ s.t. $P^{\mathsf{BF}}_{\mathbf{a}}$ is equivalent to $\varphi$ as follows.
2: for clause $\varphi_i$ of $\varphi$ do
3:     Set $\mathbf{a}[f(\varphi_i)] = 1$.
4: end for
5: Set other entries of $\mathbf{a}$ to $*$.
6: Return $\mathsf{Gen}^K(1^\lambda, \mathbf{a})$.                ▷ Run the bit-fixing Gen.
Eval^{t-CNF}(b, k_b, x) :
1: Map $\mathbf{x}$ to a length $K$ binary string $\mathbf{y}$ according to $f$.
2: for all clause $\rho$ with $t$ literals do
3:     Evaluate $\rho(\mathbf{x})$ and set $\mathbf{y}[f(\rho)] = \rho(\mathbf{x})$.
4: end for
5: Return $\mathsf{Eval}^K(b, k_b, \mathbf{y})$.                ▷ Run the bit-fixing Eval.
```

Figure 7: FSS $(\mathsf{Gen}^{t\text{-CNF}}, \mathsf{Eval}^{t\text{-DNF}})$ for exact t-CNF formula.

Proof. For the correctness part, we show that the exact t-CNF formula φ is equivalent to the bit-fixing predicate $P^{\mathsf{BF}}_{\mathbf{a}}$ induced by $\mathbf{a}$. From $\mathsf{Gen}^{t\text{-CNF}}$, we have $\mathbf{a}[f(\varphi_i)] = 1$ and other entries of $\mathbf{a}$ are set to $*$. Recall that $P^{\mathsf{BF}}_{\mathbf{a}}(\mathbf{y}) = 1$ if

$$\bigwedge_{j \in [K]} (\mathbf{y}[j] = \mathbf{a}[j] \text{ or } \mathbf{a}[j] = *) .$$

Thus, for the entry $\mathbf{a}[j]$ of $\mathbf{a}$ not corresponding to any clause of φ, $\mathbf{a}[j]$ is matched. For the clauses contained by φ, all matched $\mathbf{y}[j]$ lead to the matched state. It suffices to only consider the evaluations of the clauses contained by φ.

Let $\mathbf{x}$ be a satisfying assignment of φ. Thus, each clause φ_i of φ evaluated under $\mathbf{x}$ is satisfied, i.e., $\varphi_i(\mathbf{x}) = 1$, then the corresponding $\mathbf{y}[f(\varphi_i)] = \varphi_i(\mathbf{x}) = 1$ and thus $\mathbf{y}[f(\varphi_i)] = \mathbf{a}[f(\varphi_i)] = 1$. Hence, $\mathbf{y}$ matches the predicate $\mathbf{a}$.

Let $\mathbf{x}$ be an unsatisfying assignment of φ. There exists at least one clause of φ evaluated under $\mathbf{x}$ being unsatisfied. WLOG, let the φ_i be one of the unsatisfied clauses, i.e., $\varphi_i(\mathbf{x}) = 0$ and thus $\mathbf{y}[f(\varphi_i)] = \varphi_i(\mathbf{x}) = 0$. From $\mathsf{Gen}^{t\text{-CNF}}$, $\mathbf{a}[f(\varphi_i)]$ is set to 1. Thus, $\mathbf{y}[f(\varphi_i)] \neq \mathbf{a}[f(\varphi_i)]$. Therefore, $\mathbf{y}$

does not match the predicate $\mathbf{a}$. We have proved that the exact t-CNF formula φ is equivalent to the bit-fixing predicate $P^{\mathsf{BF}}_{\mathbf{a}}$.

Correctness of $\mathsf{FSS}^{t\text{-CNF}}$ follows from the transformation and the correctness of FSS for bit-fixing predicates. Security follows from the security of FSS for bit-fixing predicates. □

Corollary G.3. *Assume* $\mathsf{PRG} : \mathbb{S} \to \widetilde{\mathbb{H}}$ *is a EOH-PRG relative to* $(S, \mathbb{T}, (S \times T)^2, (\mathbb{S} \times \mathbb{T})^2)$.

There exists a symmetric and shift-invariant FSS for the family of t-CNF formulae.

Given the FSS constructions for exact t-CNF, one can take the tensor product operation to the FSS for exact j-CNF for $j \in [t]$ to obtain an FSS for t-CNF via Theorem 5.1.

Theorem G.4 (FSS for t-DNF)**.** *Assume* $\mathsf{PRG} : \mathbb{S} \to \widetilde{\mathbb{H}}$ *is a EOH-PRG relative to* $(S, \mathbb{T}, (S \times T)^2, (\mathbb{S} \times \mathbb{T})^2)$.

There exists an anti-symmetric and shift-invariant FSS for the family of exact t-DNF formulae.

Given a t-DNF formula ϕ, we first take an negation to ϕ to obtain a t-CNF formula $\neg\phi$. Next we call the FSS for t-CNF to obtain an FSS for $\neg\phi$ and then take the negation to the FSS to obtain an FSS for ϕ according to Lemma B.3.

Remark G.5. *Note that in the FSS construction for t-CNF and t-DNF formulae, the key size is completely independent of the clause number m. The key size only relies on (k, t) and the EOH-PRG.*

Remark G.6. *As shown in Table 2, our direct FSS constructions for t-CNF/t-DNF from bit-fixing predicates instantiated from Ring-LWR have key size around a quarter of the previous FSS from HSS instantiated from Ring-LWE and comparable running time. For DCR assumptions, the direct FSS constructions for t-CNF/t-DNF have key size reduced by a factor of* 3.5 *and running time improved by a factor of* 1.75 *comparing with the previous FSS for bit-fixing predicates from HSS.*

FSS for CNF/DNF It is obvious that we can not directly construct FSS for CNF as for t-CNF because each clause of a CNF formula has wide-ranging literals. However, there exists a polynomial time algorithm to transform a CNF formula to an equivalent t-CNF [AB09, Lemma 2.14] via adding variables and splitting long clauses to smaller clauses. Concretely, assume the given formula has k variables and m clauses. In the transformed t-CNF formula, there are at most $\frac{kn}{2(t-1)} + k$ variables. It is still $\mathsf{poly}(k, m)$ and thus the FSS is of key size $\mathsf{poly}(k, m) \cdot \log\|\widetilde{\mathbb{H}}\|$. Although the resulting FSS construction does not reveal the clauses of the induced t-CNF, the FSS still reveals the number of new variables, which perhaps leaks the shapes of the clauses of the original CNF formula.

It is straightforward to transform a CNF/DNF formula to a branching program such that each level of the branching program has width 3 and the length km, where k is the variable number and m the clause number. The FSS for resulting branching programs does not reveals the shapes of each clause.

Remark G.7. *As shown in Table 3, the FSS constructions for width* 3 *branching programs directly from EOH-PRG instantiated from Ring-LWR have key size around one-sixth and running time improved by a factor of* 24 *comparing with the FSS for branching programs from HSS instantiated from Ring-LWE. For DCR assumptions, the direct FSS constructions for width* 3 *branching programs have key size reduced by a factor of* 2.625 *and running time reduced by a factor around* 11.45 *comparing with the FSS for width* 3 *branching programs from HSS.*

G.5 Other Applications

Multi-server PIR The counting query or retrieval matches or payload computing are the typical applications of FSS for 2-server PIR as in [BGI15, BGI16b, BKS19]. With the new FSS constructions for bit-fixing predicates, pattern matching with wildcards becomes more efficient comparing with existing schemes from HSS as indicated in Section H. The direct FSS constructions for branching programs avoid the overheads in the previous FSS for branching programs from HSS [BGI16a]. Thus, any queries expressed by branching programs can also be efficiently implemented by the new FSS for branching programs.

Polynomial number of conjunctions of intervals [BGI15] proposed an FSS scheme for the interval function, which is improved from one dimension interval to constant dimension intervals in [BGI16b]. The FSS for constant dimension intervals has many practical applications such as Splinter [WYG$^+$17], which provides private searching queries on Yelp clone of restaurant reviews, ticket search, etc. However, the key size for constant dimension intervals scales with n^d, where n is the length of the PRG seed and d the number of dimensions. Hence for super-constant dimension intervals the key size increases to super-polynomial. The tensor product from EOH-PRG works for arbitrary polynomial number of intervals via replacing the PRG by EOH-PRG and tensoring by EOH-PRG. The key size of tensor product FSS only scales linearly with the number of intervals. [BCG$^+$21] pointed out that the conjunctions of FSS from one-way function as a barrier to 1-round secure evaluation of multiply-then-truncate. As mentioned in introduction, the multiply-then-truncate operation can be implemented by a NC1 circuit. We leave this as future research to use EOH-PRG to concretely improve the efficiency of 1-round secure evaluation of multiply-then-truncate.

SQL query Splinter [WYG$^+$17] proposed using FSS to efficiently implement private SQL queries, especially focusing on hiding the *where* conditions as pointed out in Section G.1 and G.2. The FSS for bit-fixing predicates, for approximate matching functions and for DFAs from EOH-PRG greatly enriches the expressiveness of where conditions. Actually any where condition captured by NC^1 can be efficiently hidden by our FSS for branching programs. Moreover, the polynomial number of conjunctions of intervals can also be used to support polynomial number of range conditions.

Decision Tree Theorem D.2 gives an FSS for branching program without revealing the topology of the branching program via adding dummy nodes. Boyle et al. [BGI16b, Section 3.3] presented a FSS for decision tree from one-way function, which reveals the topology of the decision tree. The same method could also be used to implement an FSS for decision tree without leaking the topology of the decision tree at the cost of increased key size via adding dummy nodes.

H Comparisons

In this section, we present comparisons for FSS instantiated from concrete assumptions and the FSS constructed from HSS. We focus on the FSS for bit-fixing predicates and FSS for branching programs, respectively.

There are also FSS constructions from homomorphic secret sharing (HSS) [BKS19, OSY21]. HSS can be viewed as the dual of FSS. In particular, for HSS the input is hidden and the circuit is public, whereas for FSS, the circuit is private and the input is shared by all parties. [BKS19, OSY21] constructed HSS for RMS programs with super-polynomial plaintext space and negligible correctness error based on LWE or DCR assumptions. [BGI16a] proposed a general framework to construct an FSS for arbitrary branching program from HSS for branching programs. However, this makes the resulting FSS cumbersome. We compare our direct FSS constructions for branching programs with the FSS for branching programs from HSS. The FSS constructions in Section G.4 for t-CNF/t-DNF are based on the FSS for bit-fixing predicates from EOH-PRG

The comparison results in this section indicate our FSS for bit-fixing predicates, branching programs and DFAs from EOH-PRG are very efficient.

H.1 FSS for Bit-fixing Predicates from HSS vs. EOH-PRG

There is a novel idea to implement an FSS for bit-fixing predicates from HSS. In this section, we compare the FSS for bit-fixing predicates built directly from EOH-PRG with FSS from HSS.

We first present the idea to construct FSS for bit-fixing predicates from HSS, and then compare the two FSS schemes constructed from the two methods.

Lemma H.1. *Let* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{Eval})$ *be a secure HSS scheme. Let* $\mathbf{a} \in \{0, 1, *\}^\ell$ *be a bit-fixing predicate. Then there exists an FSS scheme for* $\mathbf{a}$ *and the FSS key consists of at least* 2ℓ *ciphertexts.*

Proof. We first create a matrix $B \in \{0, 1\}^{2\times\ell}$ corresponding to $\mathbf{a}$. The converting rule works as follows.

Case $\mathbf{a}[i] = *$: $B[0, i] = B[1, i] = 1$.

Case $\mathbf{a}[i] = 0$: $B[0, i] = 1, B[1, i] = 0$.

Case $\mathbf{a}[i] = 1$: $B[0,i] = 0, B[1,i] = 1$.

Next, we use Input to encrypt each entry of B times sk to obtain a matrix C of ciphertexts, i.e., $C[j,i] = \mathsf{HSS.Input}(B[j,i] \cdot \mathsf{sk})$ for $i \in [\ell], j \in \{0,1\}$. Assume $(\mathsf{ek}_0, \mathsf{ek}_1) := \mathsf{HSS.Setup}(1^\lambda)$. Then $(\mathsf{ek}_0, \mathsf{ek}_1, C)$ is the key of the FSS for $P^{\mathsf{BF}}_{\mathbf{a}}$.

Given an input $\mathbf{x}$, the FSS evaluation is performed by the multiplication specified by the input $\mathbf{x}$. After first multiplication, $\mathbf{x}[1] \cdot \mathsf{sk}$ is shared by the two-party as sk is additively shared in $\mathsf{HSS.Setup}$. Each party runs the multiplication

$$\mathsf{Mul}(C[\mathbf{x}[i+1], i+1], M_i \cdot \mathsf{sk})$$

for $i \in [\ell-1]$ to do the evaluation, where M_i is the memory multiplication value, i.e., $M_i = \prod_{j=1}^{i} B[\mathbf{x}[j], j]$.

In this construction, the memory value becomes 0 once the input $\mathbf{x}[i]$ is not matched. It is straightforward to prove that the resulting FSS exactly implements the bit-fixing predicate $P^{\mathsf{BF}}_{\mathbf{a}}$. The security follows from the security of HSS and the CPA security of the encryption scheme. Note that the FSS key for P_b consists of ek_b and C with 2ℓ ciphertexts. $\square$

As [BKS19, OSY21] instantiated HSS from LWE and DCR assumption, we compare the FSS for bit-fixing predicates from HSS and from EOH-PRG instantiated from LWE and DCR assumptions.

We only consider the HSS instantiated from Ring-LWE [BKS19, Figure 8] rather than from LWE because the HSS instantiated from LWE incurs larger ciphertext size. Recall the HSS instantiated from Ring-LWE, each ciphertext contains four $\mathcal{R}_q$ elements and the key is a sharing of one ring element. The most costing operation is the Mul operation. Each Mul costs exactly four multiplications over the ring $\mathcal{R}_q$ and each multiplication takes $n \log n$ multiplications over $\mathbb{Z}_q$ assume the parameters are chosen permitting the NTT optimizations. There are total ℓ Mul operations during FSS evaluation. Hence, the key size for bit-fixing predicates FSS from HSS instantiated from Ring-LWE is about $8\ell n \log q$ as each HSS ciphertext contains exactly four $\mathcal{R}_q$ elements. The key size of ek_b is omited here. The evaluation cost is $4\ell n \log n$ multiplications over $\mathbb{Z}_q$. The PRF key used to re-randomize the intermediate memory values is not taken into account here.

As for the FSS for bit-fixing predicates from EOH-PRG instantiated from (Ring-)LWE, we consider instantiations from both of the LWE assumption and the Ring-LWE assumption. Recall that in Section 8, for EOH-PRG instantiated from LWE, each correction word has size $2(n+1)\log p$ and the key size for the first bit is still $2(n+1)\log p$. There are $\ell-1$ correction words. Thus the total key size is $2\ell(n+1)\log p$. The size for the PRG matrix is not taken into account because it can be instantiated from a PRG or a random oracle. During evaluation, the most expensive operation is the PRG evaluation. Each PRG evaluation takes $2n(n+1)$ multiplications over $\mathbb{Z}_q$. Hence, the total computation cost is $2(\ell-1)n(n+1)$ multiplications over $\mathbb{Z}_q$.

For EOH-PRG instantiated from Ring-LWR, the PRG becomes $\mathsf{PRG} : \mathcal{R}_q \to \mathcal{R}_p^2 \times \mathbb{Z}_p^2$. Thus each correction word consists of 2 elements of $\mathcal{R}_p$ and 2 elements of $\mathbb{Z}_p$. Hence the total key size is $2\ell(n+1)\log p$. During evaluation, the most costing operation is still the PRG evaluation. Each PRG evaluation takes 4 multiplications over $\mathcal{R}_q$ as the PRG evaluation takes multiplications over $\mathcal{R}_q$ before applies the $\psi(\cdot)$ function. Thus total number of multiplications of FSS evaluation over $\mathbb{Z}_q$ is bounded by $4\ell n \log n$. The comparison is shown in Table 2.

Next, we consider the instantiations from DCR assumption. Recall the HSS instantiated from DCR assumption [OSY21, Section 4.2], each ciphertext contains seven $\mathbb{Z}_{N^2}$ elements(six elements suffice for the KDM security and each HSS ciphertext contains exactly seven elements). Thus, the key size for bit-fixing FSS from HSS is around $14\ell \log N^2$. For the DCR assumption, the most costing is the exponentiation operation. Each Mul takes seven exponentiation operations. Thus, the FSS evaluation takes in total 7ℓ exponentiation operations. For the EOH-PRG instantiated from DCR variant assumption, recall that the PRG is $\mathsf{PRG} : \mathbb{Z}_N \to \mathbb{Z}_{N^2}^4$ and each correction word contains 4 $\mathbb{Z}_{N^2}$ elements and thus the key size is bounded by $4\ell \log N^2$. During evaluation, each PRG evaluation takes 4 exponentiation operations and thus the FSS evaluation totally takes at most 4ℓ exponentiation operations. The comparison is shown in Table 2.

In summary,

- the direct FSS from EOH-PRG instantiated from LWE provides a quarter of the key size of the FSS from HSS instantiated from Ring-LWE whereas the running time is amplified by a factor of $\frac{n}{\log n}$.
- the Ring-LWR instantiation of EOH-PRG provides a quarter of the key size of the HSS from Ring-LWE and comparable running time.

		Security Assumption	Key Size	No. of Mul. or Exp.
LWE	HSS [BKS19]	Ring-LWE	$8\ell n \log q$	$4\ell n \log n$
	EOH-PRG	LWE	$2\ell(n+1)\log p$	$2(\ell-1)n(n+1)$
	EOH-PRG	Ring-LWR	$2\ell(n+1)\log p$	$4\ell n \log n$
DCR	HSS [OSY21, RS21]	KDM security	$14\ell \log N^2$	7ℓ
	EOH-PRG	DCR variant	$4\ell \log N^2$	4ℓ

Table 2: The comparison of FSS for bit-fixing predicates constructed from EOH-PRG and from HSS. ℓ stands for bit-fixing predicate length. For LWE assumption, n stands for the secret length, q the modulus of the LWE assumption, and p the output modulus of the PRG. The number of multiplications is counted over $\mathbb{Z}_q$. For DCR assumption, N stands for RSA modulus. Here we assume six ciphertexts suffice to achieve KDM security for the HSS from DCR assumption as in [OSY21, Section 4.2]. The number of exponentiations is counted over $\mathbb{Z}_{N^2}$.

- the direct FSS from EOH-PRG instantiated from DCR variant assumption provides significant improvements over the HSS from DCR assumption in key size and the number of exponentiations. Concretely, the key size is reduced by a factor of 3.5 and the running time reduced by a factor a 1.75.

Remark H.2. *There is no encryption operation involved in the FSS construction from EOH-PRG. If the HSS is instantiated from the LWE assumption, there exists an HSS from symmetric encryption scheme. The decryption is just an linear operation. However, if HSS is instantiated from the DCR assumption, there is an* explicit *decryption procedure involved. In our FSS construction from EOH-PRG, there is definitely no decryption operation and even there is no secret key derived in the* Gen *procedure.*

H.2 FSS for Branching Programs from HSS vs. EOH-PRG

In this section, we only consider the branching programs with out-degree 2 as in Section D.1. Given a layered oblivious branching program P, [BGI16a, Theorem A.5] provided a general method to construct an FSS for P from HSS. The construction encodes P into $\hat{P}$ and a universal branching program UBP such that for any x, $\mathsf{UBP}(\hat{P}, x) = P(x)$, and runs the HSS for $\mathsf{UBP}(\hat{P}, x)$ to hide $\hat{P}$ to obtain an FSS for P.

We roughly recall the structure of UBP and $\hat{P}$ to show the key size and the efficiency of the FSS. The encoding $\hat{P}$ contains every pair of nodes in adjacent levels of P. If the pair of nodes corresponds to an edge in P, then it is 1 in $\hat{P}$. Otherwise, it is 0. Thus, the size of $\hat{P}$ is $\sum_{i\in[1,\ell-1]} w_i \cdot w_{i+1} \le \ell \cdot w^2$. The UBP contains $\ell+1$ blocks of levels $W_0, W_1 \dots W_\ell$. The number of levels of W_i is $2w_i \cdot w_{i+1}$ for $i < \ell$. In UBP evaluation, the input to level U_m for $1 \le m < w_i w_{i+1}$, where $m = jw_{i+1} + k, 1 \le j \le w_i, 1 \le k \le w_{i+1}$, is $y_{\alpha_0} \in \hat{P}$ for $\alpha_0 = (j,k)$ and the input to level U_m for $w_i w_{i+1} \le m < 2w_i w_{i+1}$, where $m = w_i w_{i+1} + jw_{i+1} + k, 1 \le j \le w_i, 1 \le k \le w_{i+1}$ is $y_{\alpha_1} \in \hat{P}$ for $\alpha_1 = (j,k)$. In summary, the input for each U_m for $1 \le m < 2w_i w_{i+1}$ is a value in $\hat{P}$ only depending on m. The input to level $U_{2w_i w_{i+1}}$ is x_i.

The FSS for P from HSS encrypts every element of $\hat{P}$. There are roughly $\ell \cdot w^2$ HSS ciphertexts. During the evaluation, for each block except for the last level, the input is in encryption form and thus for each block U_i there are $2w_i w_{i+1} - 1$ Mul operations of HSS. There are totally at most $2\ell w^2$ number of Mul operations for the FSS evaluation.

For HSS from Ring-LWE, the key contains $4\ell \cdot w^2$ $\mathcal{R}_q$ elements as each ciphertext consists of four $\mathcal{R}_q$ elements. Thus the key size is $4\ell w^2 n \log q$. There are $8\ell w^2$ multiplications over $\mathcal{R}_q$ and thus totally $8\ell w^2 n \log n$ multiplications over $\mathbb{Z}_q$.

For HSS from DCR assumption, the key contains around $7\ell \cdot w^2$ $\mathbb{Z}_{N^2}$ elements as each ciphertext contains seven $\mathbb{Z}_{N^2}$ elements. Thus the key size is $7\ell w^2 \log N^2$. There are total $14\ell w^2$ exponentiations operations during evaluation as each Mul takes 6 exponentiations.

Now we show the key size and runtime of the FSS from EOH-PRG. For EOH-PRG instantiated from LWE, the PRG is $\mathsf{PRG} : \mathbb{Z}_q^n \to \mathbb{Z}_p^{2(n+w)}$. The size of each level correction word is $2w(n+w)\log p$ and as each level has at most w correction words. Thus the total key size is bounded by $2\ell w(n+w)\log p$. During PRG evaluation, there are $2n(n+w)$ multiplications over $\mathbb{Z}_q$ in each level. Thus, there are totally $2\ell n(n+w)$ multiplications.

For EOH-PRG instantiated from Ring-LWE, the PRG becomes $\mathsf{PRG} : \mathcal{R}_q \to \mathcal{R}_p^{2+\lfloor \frac{2w}{n} \rfloor} \times \mathbb{Z}_p^{2w-n\lfloor \frac{2w}{n} \rfloor}$ as multiple tag bits are compressed to ring elements. The size of each level correction word is $2w(n+w)\log p$

and thus the total key size is bounded by $2\ell w(n+w)\log p$. During PRG evaluation, there are $2+\lceil\frac{2w}{n}\rceil$ multiplications over $\mathcal{R}_q$ each level. Hence, there are totally $\ell(2+\lceil\frac{2w}{n}\rceil)n\log n$ multiplications over $\mathbb{Z}_q$.

For EOH-PRG instantiated from DCR variant assumption, the PRG is $\mathsf{PRG} : \mathbb{Z}_N \to \mathbb{Z}_{N^2}^{2+2w}$. The size of each level correction word is $2w(w+1)\log N^2$ and thus the total key size is $2\ell w(w+1)\log N^2$. During PRG evaluation, there are $3w+2$ exponentiations in each level as each tag bit is used to compute the exponentiation of corresponding correction word. Thus, there are totally $\ell(3w+2)$ exponentiation operations.

The key size and efficiency is summarized in Table 3.

		Security Assumption	Key Size	No. of Mul. or Exp.
LWE	HSS	Ring-LWE	$4\ell w^2 n\log q$	$8\ell w^2 n\log n$
	EOH-PRG	LWE	$2\ell w(n+w)\log p$	$2\ell n(n+w)$
	EOH-PRG	Ring-LWR	$2\ell w(n+w)\log p$	$\ell(2+\lceil\frac{2w}{n}\rceil)n\log n$
DCR	HSS	KDM security	$7\ell w^2\log N^2$	$14\ell w^2$
	EOH-PRG	DCR variant	$2\ell w(w+1)\log N^2$	$\ell(3w+2)$

Table 3: The comparison of FSS for branching programs constructed from EOH-PRG and from HSS. w stands for the branching program width and ℓ for the branching program length. For branching programs we assume fixed out-degree $d=2$. For LWE assumption, n stands for the secret length, q the modulus of the LWE assumption, and p the output modulus of the PRG. The number of multiplications is counted over $\mathbb{Z}_q$. For DCR assumption, N stands for RSA modulus. Here we assume six ciphertexts suffice to achieve KDM security for the HSS from DCR assumption as in [OSY21, Section 4.2]. The number of exponentiations is counted over $\mathbb{Z}_{N^2}$.

In summary,

- for LWE related assumption, the key size of the FSS from EOH-PRG instantiated from LWE and Ring-LWR are equal. The key size is improved from a factor of $4wn$ to $2(n+w)$ comparing to the FSS for branching programs from HSS Ring-LWE. As for the running time, the the FSS from EOH-PRG from LWE is very inefficient. The running time of the FSS from EOH-PRG from Ring-LWR is improved from a factor $8w$ to $2+\lceil\frac{2w}{n}\rceil$.
- for DCR related assumption, the key size of FSS from EOH-PRG is improved by a factor around 3 of the the key size of the FSS from HSS and the number of exponentiation operations of FSS from EOH-PRG is a w of magnitude less than the FSS from HSS.

Remark H.3. *It is worth to mention that our FSS for branching programs from EOH-PRG naturally generalize to branching programs with* multi-edge *as in Section E.1 and with polynomial out-degree as in Section E.2. However, for the FSS from HSS, the universal branching program needs to split the multi-edge at the cost of increasing edges because in the encoding $\hat{P}$ of P each two nodes and the edge value form a point in $\hat{P}$ and the polynomial out-degree of P causes worse blow-up for the universal branching programs for the number of levels of each block, which leads to worse running time.*