CNF-FSS and its Applications

Paul Bunn[⋆1], Eyal Kushilevitz[⋆⋆2], and Rafail Ostrovsky[⋆⋆⋆3]

[1] Stealth Software Technologies, Inc., Los Angeles paul@stealthsoftwareinc.com
[2] Computer Science Department, Technion, Haifa, Israel eyalk@cs.technion.ac.il
[3] Department of Computer Science and Department of Mathematics, University of California, Los Angeles rafail@cs.ucla.edu

Abstract. Function Secret Sharing (FSS), introduced by Boyle, Gilboa and Ishai [BGI15], extends the classical notion of secret-sharing a *value* to secret sharing a *function*. Namely, for a secret function f (from a class $\mathcal{F}$), FSS provides a sharing of f whereby *succinct* shares ("keys") are distributed to a set of parties, so that later the parties can non-interactively compute an additive sharing of $f(x)$, for any input x in the domain of f. Previous work on FSS concentrated mostly on the two-party case, where highly efficient schemes are obtained for some simple, yet extremely useful, classes $\mathcal{F}$ (in particular, FSS for the class of point functions, a task referred to as DPF – Distributed Point Functions [GI14,BGI15]).

In this paper, we concentrate on the multi-party case, with $p \geq 3$ parties and t-security ($1 \leq t < p$). First, we introduce the notion of CNF-DPF (or, more generally, CNF-FSS), where the scheme uses the CNF version of secret sharing (rather than additive sharing) to share each value $f(x)$. We then demonstrate the utility of CNF-DPF by providing several applications. Our main result shows how CNF-DPF can be used to achieve substantial asymptotic improvement in communication complexity when using it as a building block for constructing *standard* (t, p)-DPF protocols that tolerate $t > 1$ (semi-honest) corruptions (of the p parties). For example, we build a 2-out-of-5 secure (standard) DPF scheme of communication complexity $O(N^{1/4})$, where N is the domain size of f (compared with the current best-known of $O(N^{1/2})$ for $(2, 5)$-DPF). More generally, with $p > dt$ parties, we give a (t, p)-DPF whose communication grows as $O(N^{1/2d})$ (rather than $O(\sqrt{N})$ that follows from the $(p-1, p)$-DPF scheme of [BGI15]).[4]

⋆ This work was supported by DARPA and NIWC Pacific under contract N66001-15-C-4065. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright notation thereon. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

⋆⋆ Supported by ISF grant 2774/20, BSF grant 2018393, and NSF-BSF grant 2015782.

⋆⋆⋆ Supported in part by DARPA under Cooperative Agreement HR0011-20-2-0025, by DARPA and NIWC Pacific under contract N66001-15-C-4065, NSF grant CNS-2001096, US-Israel BSF grant 2015782, Google Faculty Award, JP Morgan Faculty Award, IBM Faculty Research Award, Xerox Faculty Research Award, OKAWA Foundation Research Award, B. John Garrick Foundation Award, Teradata Research Award, Lockheed-Martin Research Award and Sunday Group. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright annotation therein. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

[4] We ignore here terms that depend on the number of parties, p, the security parameter, etc. See precise statements in the main body of the paper below.

We also present a 1-out-of-3 secure CNF-DPF scheme, in which each party holds two of the three keys, with poly-logarithmic communication complexity. These results have immediate implications to scenarios where (multi-server) DPF was shown to be applicable. For example, we show how to use such a scheme to obtain asymptotic improvement ($O(\log^2 N)$ versus $O(\sqrt{N})$) in communication complexity over the 3-party protocol of [BKKO20].

Keywords: Secret Sharing, Function Secret Sharing (FSS), Replicated Secret Sharing (CNF)

1 Introduction

Function Secret Sharing (FSS) [BGI15] provides a sharing of a secret function f, from a class of functions $\mathcal{F}$, between p parties, such that each party's share of f (also termed "key") is *succinct* (in terms of the size of the truth-table representing f) and such that the parties can locally compute (additive) shares of $f(x)$, for any input x, without further interaction. While efficient FSS schemes are currently known only for limited classes of functions (and impossibility results demonstrate other classes of functions for which no efficient FSS scheme can exist), FSS has found enormous utility in distributed and multi-party protocols, due to its low communication overhead. Indeed, the FSS paradigm has proven to be incredibly powerful even for the most basic of function classes: Point Functions, which output a non-zero value at only a single point in their domain. FSS for the class of point functions is known as DPF (Distributed Point Functions). Since DPF and FSS were introduced [GI14,BGI15], they have found applications in many areas of cryptography (see Section 1.3 below for more details). For the case of $p = 2$ parties, highly efficient (both theoretically and practically) DPF schemes, with poly-logarithmic communication (in N) are known [GI14,BGI15,BGI16a], based on the minimal assumption that one-way functions (OWF) exist. This clearly implies $(1, p)$-DPF schemes for any $p > 2$. Obtaining similar results for the multiparty case, with $t > 1$, is an open problem and, to the best of our knowledge, the only known result in the FSS setting, based on OWF alone, is a $(p-1, p)$-DPF scheme of communication proportional to $\sqrt{N}$ from [BGI15] (and a protocol with similar communication for the special case of $(2, 3)$-DPF in [BKKO20]).

CNF secret-sharing [ISN87] (also known as "replication-based secret-sharing" in [GI99]) has found great utility in a variety of applications, including: Verifiable Secret Sharing and MPC protocols [Mau02], PIR [BIK05] and others.[5] A (t, p)-CNF secret-sharing works by first additively breaking the secret value s to $\ell = \binom{p}{t}$ random shares $\{s_T\}_{T \in \binom{[p]}{t}}$, subject to their sum satisfying $\sum_{T \in \binom{[p]}{t}} s_T = s$, and then distributing each share s_T to all parties *not* in T. It satisfies t-secrecy

[5] In fact, CNF secret sharing is a special case of formula-based secret sharing [BL88]; similar generalizations are in principle possible also in the context of FSS.

since, for any set T of t parties, all parties in T miss the share s_T.[6] In the present work, we adapt the same approach in the context of FSS and introduce the notion of CNF-FSS (and the analogous notion of CNF-DPF, when the class of functions being shared are point functions) whereby, given an input x, the parties obtain a CNF-secret-sharing of $f(x)$.[7] We then explore the power of this new notion by constructing CNF-DPF schemes and by getting applications of these constructions. Specifically, in Section 1.1 we describe our main result – for the case of p parties and $1 < t < p$, we show how to use CNF-DPF schemes to obtain improved *standard* DPF schemes; then, in Section 1.2, we deal with the special case $p = 3, t = 1$, where CNF-sharing is useful in some applications.

1.1 Improved Multiparty DPF with $t > 1$ from CNF-DPF

As mentioned, [GI14] and subsequent works demonstrated highly efficient 1-out-of-2 DPF schemes (with logarithmic communication in the size of the DPF domain), but much less is known for the $p > 2$ and secrecy threshold $t > 1$ case. A trivial solution for $(p-1)$-out-of-p DPF is to additively share the truth-table of the point function $f_{x,v} : [N] \to \mathbb{F}$ as a string. However, this approach has communication complexity $O(N \cdot m)$, where N is the size of the domain of $f_{x,v}$ and m is its output length (note that this trivial solution is, in fact, information-theoretic). A more efficient $(p-1)$-out-of-p DPF solution is given by [BGI15] and has communication of essentially $O(\sqrt{N})$ (more precisely $O(\sqrt{N \cdot 2^p} \cdot (\lambda + m))$, where λ is the security parameter).[8] For the special case $p = 3$, a scheme with $O(\sqrt{N})$ communication was also pointed out in [BKKO20]. When making stronger assumptions than the existence of OWF, additional results are known: for example, using PK assumptions and operations, specifically seed-homomorphic PRG, [CBM15] also achieve a scheme with communication that depends on $\sqrt{N}$, but has better dependency on p and, under the LWE assumption, a $(p-1, p)$-FSS scheme for all functions can be constructed [DHRW16].

In this paper, we show how to get better communication complexity, when one can settle for smaller values of t. The high-level idea is as follows. First, we construct, as an intermediate tool, a (t, p)-CNF-DPF scheme. The key-generation algorithm Gen of our (t, p)-CNF-DPF scheme, produces $\ell = \binom{p}{t}$ keys $\{K_T\}_{T \in \binom{[p]}{t}}$ and gives each key K_T to each party not in T (i.e., to each party in $[p] \setminus T$). This is done by invoking the key-generation algorithm of [BGI15] for the $(\ell - 1, \ell)$-DPF case. Algorithm Eval of [BGI15] can be applied to any input y and any

[6] CNF sharing immediately implies additive sharing, by arbitrarily assigning each share s_T to one of the parties who hold it (i.e., a party not in T), and each party's share being the sum of all (at least one) shares assigned to it.

[7] In our constructions, this is often achieved by having each party receive multiple overlapping keys, in CNF form, that encode the DPF function; however, in general, this is not a requirement. For formal definitions, see Section 2 (including Remark 1).

[8] In [BGI15], the range may be a group, as they only need the additive structure. However, we require a Ring structure for the range, since we will also use multiplication. For concreteness, we can think of the field $GF[2^m]$, represented by m-bit strings.

key K_T, and together these ℓ values give an additive sharing of $f_{x,v}(y)$ with ℓ values. Our next idea is to view the domain $[N]$ of the point function as a d-dimensional cube, where each dimension is of size $M = N^{1/d}$. This allows to express the point function $f_{x,v}$ as the product of d point functions $f_1, \ldots, f_d$, on much smaller domain of size M and apply CNF-DPF sharing to each f_i. Finally, the property of CNF sharing is that with the right relations between p, t and d (specifically, when $p > td$) non-interactive multiplication is possible, since the replication guarantees that each term in the product is known to some party.

These results are presented in detail in Section 3. As an example, we get a (standard) $(2, 5)$-DPF scheme with communication $O(N^{1/4})$ (instead of $O(N^{1/2})$) and, more generally, with $p > dt$ parties, we get a (t, p)-DPF with communication $O(N^{1/2d} \cdot \sqrt{2^{p^t}} \cdot p^t \cdot d \cdot (\lambda + m))$. In fact, one can also obtain a scheme with information-theoretic security and communication complexity $O(N^{1/d} \cdot p^t \cdot d \cdot m)$ by just replacing the CNF-DPF above (based on [BGI15]) with a naive CNF sharing of the truth table of each f_i.

Our results may be useful in several cases where DPF was already shown to be relevant. For instance, consider Binary CPIR (i.e., Computational Private Information Retrieval schemes where servers' answers are a single bit) or, more generally, CPIR with constant answer length. Binary PIR schemes are useful in the context of retrieving long records, and have connections with locally decodable codes (LDC). The fact that DPF schemes imply Binary CPIR schemes with the same complexity was shown in [GI14,BGI15] and this connection holds also for the t-private version of DPF and CPIR schemes. Hence, our (t, p)-DPF schemes with communication $\approx O(N^{1/2d})$ (for $p = dt + 1$ parties) imply (t, p)-binary-CPIR with similar complexity. Before, to get (t, p)-binary-CPIR, one could use a number of servers p that is exponential in t, to get much better communication complexity. Specifically, one could get information-theoretic Binary PIR with communication $N^{o(1)}$ using $p = 3^t$ servers (by combining [BIW07] with [Efr09]), or Binary CPIR of poly-logarithmic communication using $p = 2^t$ servers (by combining [BIW07] with a 2-server DPF, as pointed out in [GI14]). Or, with a moderate number of servers p, one could use a binary (information theoretic) PIR with communication $\approx O(N^{1/d})$ (again, for $p = dt+1$) [DIO98,BIK05]. We essentially get a quadratic improvement in this regime of parameters. Similar improvements can be applied also to the PIR writing model [OS97].

1.2 1-out-of-3 CNF-DPF

Motivated by applications, we give a special treatment for the 3-party case. A $(1, 3)$ standard DPF scheme of poly-logarithmic communication complexity is easy to achieve just by using solutions for the $(1, 2)$-case and not utilizing the third party at all. However, in some settings, (2,3)-DPF may be required, for which only schemes of communication $O(\sqrt{N})$ are known. For example, in [BKKO20] a so-called *Distributed ORAM* (DORAM) scheme is presented that relies on $(2, 3)$-DPF. We observe that [BKKO20] does not need the full strength of (2,3)-security and, instead, can rely on a $(1, 3)$-DPF, provided an appropriate "CNF" replication of keys between the 3 parties (i.e., there are still 3 keys,

as in the $(2,3)$ case, but each of them is known to 2 of the parties, which can clearly only give 1-security). Note that this does not seem trivial to achieve: if we start from a (2,3)-DPF scheme then we can easily get a $(1,3)$-CNF-DPF but with much higher communication than what we aim for, and if we start from a $(1,3)$-DPF and give each key to 2 parties, then security is lost. Nevertheless, we show (in Section 4) how to construct a $(1,3)$-CNF-DPF scheme, while still maintaining poly-logarithmic communication. Hence, improving the asymptotic communication of the scheme from [BKKO20].[9]

While we focus on the case of 1-out-of-3 CNF-DPF, as our construction is similar in spirit to the 2-party DPF scheme of [BGI15], the same modifications that are proposed in [BGI15] to extend DPF to FSS for related function classes (e.g. "sparse" vectors or matrices, step functions, interval functions, etc.) are applicable for our 1-out-of-3 scheme as well; details and additional discussion will be provided in the full version.

Additionally, $(1,3)$-CNF-DPF schemes have features that may be useful in other applications. The most basic one is the ability to perform *multiplication*; that is, given two point functions f and g that are shared using a $(1,3)$-CNF-DPF scheme, and any evaluation points x and y, the parties can (non-interactively) generate additive shares of the product[10] $f(x) \cdot g(y)$. The reason is that $f(x)$ is the sum of 3 values, each of which is known to 2 parties, and similarly $g(y)$ is the sum of 3 values, each known to 2 parties, so their product contains 9 terms each known to (at least) one party. (A similar observation is what we use for the general (t,p)-case (see below), and what is used in other contexts where CNF secret sharing is used; see, e.g., [BIK05].) Similarly, we can multiply a point function by a (secret-shared) value a to get additive shares of $a \cdot f(x)$, as well as other generalizations. We note that the ability to perform non-interactive multiplication(s), in CNF-FSS schemes, can be used to extend the known function classes for which *standard* FSS is available. For example, FSS for functions that involve the product of two sparse matrices, or of a sparse vector times a (secret-shared) pseudo-random matrix, can be readily built using CNF-FSS. (See below for comparison with a related notion from [BGI16b].)

1.3 Related Work

Distributed point functions (DPF) were introduced by Gilboa and Ishai [GI14] who gave efficient constructions of (2-party) DPF schemes, based on the minimal assumption of OWF, together with a spectrum of useful applications, such

[9] In order to use our $(1,3)$-CNF-DPF scheme as a subprotocol of [BKKO20], it must be converted into a distributed (*dealerless*) protocol. While generic techniques exist to perform this conversion, using these would decrease overall performance of the resulting protocol. In the full version we show how our $(1,3)$-CNF-DPF can be converted into a distributed protocol in a black-box manner, while maintaining polylog communication (though this conversion does incur a hit in round-complexity over the protocol of [BKKO20]: log rounds versus constant-round).

[10] As mentioned, for the product to be defined we need the range of the functions to be a ring rather than just a group.

as improved schemes for 2-server (computational) PIR, "PIR writing" (PIW) and related problems. Boyle, Gilboa and Ishai [BGI15], generalized this notion to other classes of functions, obtaining the notion of *Function Secret Sharing* (FSS). They present various FSS schemes, for DPF and other classes and, of particular relevance to the present work, they presented the first non-trivial solution for *multi-party* DPF. Further extension and optimizations of FSS are given in [BGI16b]. In particular, this paper presents the notion of *FSS product operator*, that allows to combine FSS schemes for classes $\mathcal{F}_1, \mathcal{F}_2$ to an FSS for the class of their products. For a more detailed discussion and a comparison of this operator with our construction, see Section 3.2.

The related notion of *Homomorphic Secret Sharing* (HSS), which is "dual" to FSS (in the sense that it switches roles between functions and inputs), was introduced in [BGI16a] and further studied in [BGI$^+$18b]. It allows for sharing a value x between p parties, so that given a function $f \in \mathcal{F}$, each party may (non-interactively) apply Eval to its share of x (and a representation of f) so as to get a sharing of $f(x)$. In particular, [BGI16a] gives a 2-party FSS for a wide class of functions such as branching programs (though, under a stronger assumption, DDH, and with 1/poly error probability). This result yields 2-party secure computation protocols with communication sub-linear in the circuit size. Other applications of FSS include *silent OT extension* and *pseudorandom correlation generation* for simple correlations [BCG$^+$19], and many more.

Another application that makes use of DPF for $p > 2$ parties is Distributed ORAM (DORAM), where read/write operations into memory are done obliviously (see, e.g., [LO13,ZWR$^+$16,DS17,GKW18,JW18,KM19,BKKO20,HV20]). Concretely, [BKKO20] use a $(2, 3)$-DPF scheme in order to construct efficient 3-party DORAM. They use overlaps between the keys of pairs of servers, to invoke PIR/PIW schemes that rely on replication of information. This serves as one motivation for the study of CNF-sharing in the present paper. Before the work of [BKKO20], Doerner and Shelat [DS17] used 2-party DPF to construct what can be viewed as a DORAM protocol in the two-party setting. In comparing [DS17] and [BKKO20] as multiparty DORAM protocols: the former has superior communication complexity (polylog N versus $\sqrt{N}$) but inferior round-complexity (logarithmic in N versus constant-round). Applying the results in the current paper to [BKKO20], the communication complexity improves from $\sqrt{N}$ to polylog (albeit with a cost of logarithmic round-complexity), thus matching the asymptotic communication complexity of [DS17][11].

In [CBM15], the authors describe a multi-server system called Riposte for anonymous broadcast messaging, with various features. They use the general notion of (t, p)-DPF, hence giving motivation for improving the communication complexity of such schemes. While concentrating on a 3-server system (using 2-party DPF), they also present a $(p-1, p)$-DPF scheme of $O(\sqrt{N})$ communication. (It differs from the scheme of [BGI15] by using also PK assumptions and operations, specifically seed-homomorphic PRGs, and also their scheme does not have the 2^p term for communication.) A follow-up paper describes the Ex-

[11] This is demonstrated in the full version.

press system [ECZB19], in a 2-server setting and using 2-party DPF. They also mention the need for improved multi-party DPFs. Finally, Blinder [APY20] is a scalable system for so-called *Anonymous Committed Broadcast*. As with the previous systems, Blinder also uses DPF as a building block but concentrates on the multi-server case.

As mentioned, the CNF version of secret sharing [ISN87] is useful in many applications, including VSS and MPC (e.g. [Mau02,IKKP15,AFL+16,FLNW17]), PIR [BIK05], etc. In the context of *share conversion*, it was shown in [CDI05] that shares from the CNF scheme can be locally converted to shares of the same secret from any other linear scheme realizing the same access structure (e.g., shares from the (t,p)-CNF scheme can be converted to shares for the t-out-of-p Shamir scheme).

1.4 Organization

We provide the requisite definitions and notation in Section 2. We then describe our improved t-out-of-p secure DPF schemes in Section 3, and our 1-out-of-3 secure CNF-DPF construction, with poly-logarithmic communication complexity, in Section 4.

2 Model and Definitions

Notation: We use $[a..b]$ to denote the integers in the (closed) interval from a to b, and $[b]$ to denote $[1..b]$. We further denote by $\binom{[b]}{t}$ the collection of all subsets of $[b]$ of size t.

Definition 1 ***FSS [BGI15,BGI16a].*** *A t-out-of-p Function Secret Sharing scheme ((t,p)-FSS, for short) for a class of functions $\mathcal{F} = \{f : D \to \mathbb{G}\}$, with input domain D and output domain an abelian group $(\mathbb{G},+)$, is a pair of PPT algorithms $FSS = (\mathsf{Gen}, \mathsf{Eval})$ with the following syntax:*

- $\mathsf{Gen}(1^\lambda, f)$*: On input the security parameter λ and a description of a function $f \in \mathcal{F}$, outputs p keys: $\{\kappa_1, \ldots, \kappa_p\}$;*
- $\mathsf{Eval}(i, \kappa_i, x)$*: On input an index $i \in [p]$, key κ_i, and input string $x \in D$, outputs a value ("share") $y_i \in \mathbb{G}$;*

satisfying the following correctness and secrecy requirements:

Correctness. *For all $f \in \mathcal{F}$, $x \in D$:*

$$Pr\left[\{\kappa_1, \ldots, \kappa_p\} \leftarrow_R \mathsf{Gen}(1^\lambda, f) \; : \; \sum_{i=1}^{p} \mathsf{Eval}(i, \kappa_i, x) = f(x)\right] \; = \; 1.$$

Security. *For any subset of indices $\mathcal{I} \subset [p]$ with size $|\mathcal{I}| \leq t$, there exists a PPT simulator Sim such that for any polynomial-size function sequence $f_\lambda \in \mathcal{F}$, the following distributions are computationally indistinguishable:*

$$\left\{\{\kappa_1, \ldots, \kappa_p\} \leftarrow_R \mathsf{Gen}(1^\lambda, f) \; : \; \{\kappa_i\}_{i \in \mathcal{I}}\right\} \approx_C \left\{\{\kappa_1, \ldots, \kappa_{|\mathcal{I}|}\} \leftarrow_R \mathsf{Sim}(1^\lambda, D, \mathbb{G})\right\}.$$

We now extend the original FSS definition to *Conjunctive Normal Form (CNF) FSS*. This is similar to Definition 1, except that the output of Eval, over all p parties, should be a legal (t,p)-CNF secret-sharing of $f(x)$ (rather than additive secret sharing). That is, let $\mathcal{S}_t$ denote the set of subsets of $[p]$ of size t (there are $\binom{p}{t}$ such subsets) and, for any $i \in [p]$, let $\mathcal{T}_t^{\mathcal{P}_i} \subset \mathcal{S}_t$ denote the subsets of $\mathcal{S}_t$ that do *not* contain index i (there are $\binom{p-1}{t}$ such subsets). Then the algorithm Eval of a (t,p)-CNF-FSS scheme produces, for each party $i \in [p]$, all the shares of $f(x)$ corresponding to $\mathcal{T}_t^{\mathcal{P}_i}$.

Definition 2 *A t-out-of-p CNF-FSS (also denoted (t,p)-CNF-FSS) scheme for a class of functions $\mathcal{F} = \{f : D \to \mathbb{G}\}$ with input domain D and output domain an abelian group $(\mathbb{G}, +)$ is a pair of PPT algorithms CNF-FSS $= (\mathsf{Gen}, \mathsf{Eval})$ with the following syntax:*

- $\mathsf{Gen}(1^\lambda, f)$*: On input the security parameter λ and a description of a function $f \in \mathcal{F}$, outputs p keys: $\{\kappa_1, \ldots, \kappa_p\}$;*
- $\mathsf{Eval}(i, \kappa_i, x)$*: On input an index $i \in [p]$, key κ_i, and input string $x \in D$, outputs a sequence of $a = \binom{p-1}{t}$ values $\mathcal{Y}_i := \{y_T\}_{T \in \mathcal{T}_t^{\mathcal{P}_i}}$ in $\mathbb{G}^a$;*

satisfying the following consistency, correctness and secrecy requirements:

Consistency. *For every function $f \in \mathcal{F}$, input $x \in D$, pair of distinct parties $i, i' \in [p]$, and set $T \in \mathcal{S}_t$ that does not contain i or i' (i.e. $T \in \mathcal{T}_t^{\mathcal{P}_i} \cap \mathcal{T}_t^{\mathcal{P}_{i'}}$), when producing keys $\{\kappa_1, \ldots, \kappa_p\} \leftarrow_R \mathsf{Gen}(1^\lambda, f)$ and getting $y_{i,T} \in \mathcal{Y}_i$ for this $T \in \mathcal{T}_t^{\mathcal{P}_i}$ from $\mathsf{Eval}(i, \kappa_i, x)$ (among other outputs) and, similarly, $y_{i',T} \in \mathcal{Y}_{i'}$ for this same $T \in \mathcal{T}_t^{\mathcal{P}_{i'}}$ from $\mathsf{Eval}(i', \kappa_{i'}, x)$ then, with probability 1, we have $y_{i,T} = y_{i',T}$. Denote by y_T this common share value held by all parties $i \notin T$.*

Correctness. *For all $f \in \mathcal{F}$, $x \in D$: let $\{\kappa_1, \ldots, \kappa_p\} \leftarrow_R \mathsf{Gen}(1^\lambda, f)$ and let y_T, for all $T \in \mathcal{S}_t$, as defined above. Then, with probability 1, we have: $\sum_{T \in \mathcal{S}_t} y_T = f(x)$.*

Security. *For any subset of indices $\mathcal{I} \subset [p]$ with size $|\mathcal{I}| \leq t$, there exists a PPT simulator Sim such that for any polynomial-size function sequence $f_\lambda \in \mathcal{F}$, the following distributions are computationally indistinguishable:*

$$\left\{\{\kappa_1, \ldots, \kappa_p\} \leftarrow_R \mathsf{Gen}(1^\lambda, f) \ : \ \{\kappa_i\}_{i \in \mathcal{I}}\right\} \approx_C \left\{\{\kappa_1, \ldots, \kappa_{|\mathcal{I}|}\} \leftarrow_R \mathsf{Sim}(1^\lambda, D, \mathbb{G})\right\}.$$

Remark 1. The above definition requires only that the outputs of Eval, i.e. $\mathcal{Y}_i = \{y_T\}_{T \in \mathcal{T}_t^{\mathcal{P}_i}}$, over all parties i, is a legal CNF secret sharing (of $f(x)$). A stricter requirement that some of our constructions satisfy is that: (1) the keys themselves are in a CNF form, i.e. that each party i receives keys $\mathcal{K}_i := \{\kappa_T\}_{T \in \mathcal{T}_t^{\mathcal{P}_i}}$; and (2) each share y_T is computed only from κ_T. Satisfying (1) and (2) immediately implies that the shares are consistent and are in CNF form. Our CNF-DPF schemes in Section 3 have this property; while the (1,3)-CNF-DPF scheme of Section 4 satisfies (1) but not (2). That is, for the (1,3)-CNF-DPF scheme of Section 4, the keys are in CNF format, but Eval needs to operate on *both* keys of each party in order to produce its two shares.

Additionally, we will require the definitions of several variants of standard DPF for our (1,3)-CNF-DPF scheme of Section 4, which for clarity are defined as they are needed in Section 4.2.

3 t-out-of-p DPF from CNF-DPF

In this section, we discuss *standard* (i.e. non-CNF) DPF for $p > 2$ parties, and security threshold $t > 1$. As mentioned in Section 1.1, in contrast to the case $t = 1$, where very efficient poly-logarithmic solutions are known, even with $p = 2$ parties, the communication complexity in the general case of (t, p)-DPF (and more generally (t, p)-FSS) is much less understood.

We begin with a fixed choice of parameters $t = 2$ and $p = 5$ and demonstrate in Section 3.1 below how CNF-DPF can be used to construct an improved (standard) $(2, 5)$-DPF scheme. We then generalize this approach in Section 3.2 to show how to construct (t, p)-DPF from CNF-DPF for a variety of parameters t and p.

3.1 Example: 2-out-of-5 DPF

To demonstrate our ideas, we start with a concrete example of $t = 2$ and $p = 5$, and present a $(2, 5)$-DPF of communication $O(N^{1/4})$. As a first step towards this goal, we construct a (2,5)-CNF-DPF scheme B with communication $O(\sqrt{N})$. For this, we use the (standard) $(q-1, q)$-DPF scheme of [BGI15], with $q = \binom{5}{2} = 10$. This gives 10 keys $K_1, \ldots, K_{10}$ so that any set of 9 keys gives no information about the point function f, and those keys allow for producing additive shares for the value $f(y)$, for any input y. Next, associate with each key K_i ($i \in [10]$) a distinct subset $T \in \binom{[5]}{2}$ and give K_i to the 3 parties outside the set T (or, equivalently, do not give K_i only to the 2 parties in T). In other words, the key κ_j of party j in our scheme B consists of all the keys K_i that correspond to sets T with $j \notin T$ (there are $6 = \binom{4}{2}$ such sets). Our $B.\mathsf{Eval}$ algorithm, on input κ_j, simply works by applying the Eval algorithm of [BGI15] to each K_i that κ_j contains, separately. This gives a $(2, 5)$-CNF scheme B as needed: 10 shares/keys $K_1, \ldots, K_{10}$, where each pair of parties misses exactly one of them. Therefore, the view of this pair of parties in B is identical to the view of a corresponding set of 9 parties in the [BGI15] scheme, which is 9-secure (for $q = 10$).

Next, assume for convenience, that $N = M^2$. In this case, we can view points in the domain $[N]$ as *pairs* of elements in $[M]$ (e.g., we can view the point $x \in [N]$ as $(x_1, x_2) \in [M] \times [M]$). Similarly, we can view the truth table of the function $f_{x,v} : [N] \to \mathbb{F}$ as an $M \times M$ matrix, with v in position (x_1, x_2) and 0's elsewhere. With this view, we can write the point function $f_{x,v}$ as the product of two point functions (on a smaller domain) $f_{x_1,v}, f_{x_2,1} : [M] \to \mathbb{F}$. That is, for every $y = (y_1, y_2)$, we have $f_{x,v}(y) = f_{x_1,v}(y_1) \cdot f_{x_2,1}(y_2)$ (because if $y = x$ then $y_1 = x_1$ and $y_2 = x_2$ so the product will be $v \cdot 1 = v$ and, otherwise if $y \neq x$, the product will be 0 as either the row satisfies $y_1 \neq x_1$ or the column satisfies $y_2 \neq x_2$). The Gen algorithm will apply the $B.\mathsf{Gen}$ algorithm twice to generate 10 keys $\{K_1, \ldots, K_{10}\}$ for $f_{x_1,v}$, and 10 keys $\{K'_1, \ldots, K'_{10}\}$ for $f_{x_2,1}$; and then, distribute each set of keys, $\{K_i\}$ and $\{K'_i\}$, to the 5 parties according to the CNF associations, as described above. The Eval algorithm, on input $y = (y_1, y_2)$, is applied with those keys to get additive sharing of $f_{x_1,v}(y_1)$ (into 10 shares that we denote $u_1, \ldots, u_{10}$); and similarly to get additive sharing of $f_{x_2,1}(y_2)$ (into 10

shares that we denote $v_1, \ldots, v_{10}$). That is, we have:

$$f_{x,v}(y) = f_{x_1,v}(y_1) \cdot f_{x_2,1}(y_2) = \sum_{i=1}^{10} u_i \cdot \sum_{j=1}^{10} v_j = \sum_{i,j \in [10]} u_i \cdot v_j .$$

Finally, we observe that because B is a CNF-DPF scheme, the CNF sharing guarantees that for each pair (i,j), there is (at least one) party that knows both values u_i, v_j (since u_i is not known only to 2 parties and v_j is not known only to 2 parties but we have $p = 5$ parties). Hence, we can allocate the product $u_i \cdot v_j$, for each pair (i,j), to one of the 5 parties, which will compute it and include it in its share. So letting a_k denote party k's sum of all the pairs (i,j) allocated to it, the desired output value $f_{x,v}(y)$ is additively shared across the 5 parties as: $a_1 + \ldots + a_5$, as desired.

Correctness of the above scheme follows by the description. 2-security follows since the information known to each pair of parties T is only what they got in two invocations of the CNF-DPF scheme B. Since B is 2-secure this keeps the functions $f_{x_1,v}, f_{x_2,1}$ secret. Other than that, everything else is local computations that each party does on its own while applying Eval. As for the communication complexity (i.e., key sizes), both invocations of B and our final scheme have communication of $O(\sqrt{M} \cdot (\lambda + m)) = O(N^{1/4} \cdot (\lambda + m))$ where, as above, m denotes the output length of the point function and λ is the security parameter.

3.2 Extending to General *t*-out-of-*p* DPF

Next, we generalize the above example. Suppose one wants to secret-share a point function with security threshold t, and has $p \geq dt + 1$ parties available for the sharing, for some d (e.g., in the example, $p = 5$ and $t = d = 2$). Then, our next result shows how this can be done with communication complexity $\approx O(N^{1/2d})$.

Theorem 3 *Let t, p, d be such that $p = dt + 1$. Then, assuming OWF exists, there is a (standard, computational) (t,p)-DPF scheme Π with communication $O(N^{1/2d} \cdot \sqrt{2^{p^t}} \cdot p^t \cdot d \cdot (\lambda + m))$.*

Note that we usually think of p (and hence also t and d) as being "small" and of N as being the main parameter, so the not-so-good dependency on p (which is inherited from [BGI15]) is secondary.

Remark 2. Our result uses the $(p-1)$-out-of-p DPF protocol of [BGI15] and, as such, the result is limited to DPF functions where the range is a group $\mathbb{G}$ of characteristic two. Concretely, they consider functions with range $\{0,1\}^m$ which we view as $\mathbb{F} = GF[2^m]$, as we require a ring structure. While it is not explicitly stated in the conference version of [BGI15], the full version will include a generalization of the $(p-1)$-out-of-p DPF protocol for more general groups $\mathbb{G}$ (private communication with authors of [BGI15]), and this generality is transferrable to our constructions. Specifically, for "small" q, a simple modification

allows generalization to $\mathbb{G} = \mathbb{Z}_q$, and this can be further generalized to larger groups $\mathbb{Z}_m$ for m that is a product of distinct primes, by utilizing the Chinese Remainder Theorem in the Eval algorithm. Note that these ranges are what is needed for most applications of DPFs in the literature.

Proof. Assume for concreteness that $N = M^d$, and view each input in the domain as a vector of d values, e.g. $x = (x_1, \ldots, x_d) \in [M]^d$. Then the truth table of the point function $f_{x,v}$ can be viewed as a d-dimensional cube with v at position $x = (x_1, \ldots, x_d)$, and 0's elsewhere. Next, view the point function $f_{x,v} : [N] \to \mathbb{F}$ as the product of d point functions $f_{x_i,v_i} : [M] \to \mathbb{F}$, where $v = \prod_{i=1}^{d} v_i$ (e.g., $v_1 = v, v_2 = \ldots = v_d = 1$). Hence, we have, for all input $y = (y_1, \ldots, y_d)$ in the domain of $f_{x,v}$:

$$f_{x,v}(y) = \prod_{i=1}^{d} f_{x_i,v_i}(y_i).$$

As in the example, the idea will be to share each of the d "smaller" point functions f_{x_i,v_i} separately, using a CNF-DPF scheme B_i, in such a way that during the evaluation stage we can combine the outcomes of the d evaluations to obtain a (standard) additive sharing of $f_{x,v}(y)$. To construct each B, we use as a building block the $(q-1, q)$-DPF scheme of [BGI15], with $q = \binom{p}{t}$. Invoking the [BGI15] scheme with these parameters generates q keys, which we denote $\{K_T\}_{T \in \binom{[p]}{t}}$, and each $B.\mathsf{Gen}$ distributes each K_T to all parties in $[p] \setminus T$. Meanwhile, each $B.\mathsf{Eval}$ works by applying the Eval algorithm of the [BGI15] scheme for each key K_T separately. By construction, this is indeed a CNF-sharing scheme. The t-security of B follows from the fact that each set T of t parties misses the share K_T and by the $(q-1)$-security of the [BGI15] scheme (assuming OWF). The size of the keys in the [BGI15] scheme (on domain of size M) is $O(\sqrt{M \cdot 2^q} \cdot (\lambda + m))$ and each of the p parties gets $\binom{p-1}{t} < p^t$ of them, and also $q < p^t$ so all together $O(\sqrt{M \cdot 2^{p^t}} \cdot p^t \cdot (\lambda + m))$. The key-generation algorithm of our scheme, $\Pi.\mathsf{Gen}$, works by invoking the algorithm $B.\mathsf{Gen}$ d times, once for each point function f_{x_i,v_i}. Denote the keys generated by the i-th invocation by $\{K_{i,T}\}_{i \in [d], T \in \binom{[p]}{t}}$.

The algorithm $\Pi.\mathsf{Eval}$, on input $y = (y_1, \ldots, y_d)$, works as follows: Denote by $S_{i,T}$ the share obtained by applying the Eval algorithm of [BGI15] on $K_{i,T}$ and y_i. By the correctness of the underlying [BGI15] scheme, we have that $f_{x_i,v_i}(y_i) = \sum_{T_i \in \binom{[p]}{t}} S_{i,T_i}$, and hence:

$$f_{x,v}(y) = \prod_{i=1}^{d} f_{x_i,v_i}(y_i) = \prod_{i=1}^{d} \left(\sum_{T_i \in \binom{[p]}{t}} S_{i,T_i} \right) = \sum_{T_1, \ldots, T_d \in \binom{[p]}{t}} \left(\prod_{i=1}^{d} S_{i,T_i} \right).$$

Consider any term of the form $\prod_{i=1}^{d} S_{i,T_i}$. Each of the shares S_{i,T_i} is not known only to the t parties in T_i, so all together at most $d \cdot t$ parties miss any of d shares of this term. Since $p > d \cdot t$, there is (at least) one party that knows all the shares of this term and can compute it. Assign each term to, say, the lexicographically first party (by index) that knows this term, and let a_j, for $j \in [p]$, be the sum of all terms assigned to the j-th party. We get that, as needed:

$$\sum_{j=1}^{p} a_j = \sum_{T_1,\ldots,T_d \in \binom{[p]}{t}} \left(\prod_{i=1}^{d} S_{i,T_i} \right) = f_{x,v}(y),$$

In terms of t-security: this follows since we invoke d times (independently) the t-secure scheme B. The size of keys is therefore d times larger than the size of keys in B, i.e. $O(\sqrt{M \cdot 2^{p^t}} \cdot p^t \cdot (\lambda + m) \cdot d) = O(N^{1/2d} \cdot \sqrt{2^{p^t}} \cdot p^t \cdot d \cdot (\lambda + m))$. □

Remark 3. We observe that setting e.g. $v_1 = v$, and then insisting that for all $i > 1$: $\{f_{x_i,v_i}\}$ has range $\{0,1\}$ instead of $\mathbb{F}$ (with $v_i := 1$ for all such i), removes the factor of m in the communication complexity of all of the $\{f_{x_i,v_i}\}$ (except for f_{x_1,v_1}), and consequently the overall complexity of Π in Theorems 3 and 4 can be reduced (by a factor of m for Theorem 4 below, and by a factor of m in one of the additive terms for Theorem 3, which is meaningful when $m >> \lambda$).

Note that the above scheme, as with most FSS/DPF schemes, provides computational security. However, it is possible to get *information-theoretic* security with only a relatively small loss (essentially replacing the $N^{1/2d}$ term in the communication of the above scheme with $N^{1/d}$) and, in fact, getting a slightly better dependency on p. More precisely:

Theorem 4 *Let t, p, d be such that $p = dt + 1$. Then, there is a (standard, **information-theoretically**-secure) (t,p)-DPF scheme Π with communication complexity $O(N^{1/d} \cdot p^t \cdot d \cdot m)$.*

Proof. The proof is very similar to the previous construction above, except that we replace all invocations of the (computational) scheme from [BGI15] for creating $q = \binom{p}{t}$ key shares, with the naive (but information-theoretically secure) scheme where the truth table is just CNF-shared as a string (with parameters (t,p)). When applied to point functions with domain size M and output length m, the key size of each of the p parties is at most $M \cdot m \cdot q = M \cdot m \cdot \binom{p}{t}$. The scheme then proceeds as above, by CNF-sharing the d point functions on domain of size $M = N^{1/d}$. The correctness and security arguments are similar to the above. The key size is $O(M \cdot p^t \cdot d \cdot m) = O(N^{1/d} \cdot p^t \cdot d \cdot m)$. □

Similarly, one can plug the $(p-1,p)$-DPF scheme of [CBM15] to the construction of CNF-DPF in Theorem 3. This scheme relies on the existence of seed-homomorphic PRG [BLMR13] (compared to the minimal assumption of OWF, as in standard DPF schemes), and has better dependency on p, i.e., communication of $O(\sqrt{N} \cdot poly(p) \cdot (\lambda + m))$. Hence, we can get a (t,p)-DPF scheme, like in Theorem 3, under the same assumption as [CBM15], with somewhat better communication of $O(N^{1/2d} \cdot poly(p^t) \cdot d \cdot (\lambda + m))$.

It is instructive to compare our technique with that of [BGI16b] (and its full version in [BGI18a]). Concretely, [BGI18a, Thm. 3.22] shows how to combine (t,p_1)-FSS for a class $\mathcal{F}_1$ and a (t,p_2)-FSS for a class $\mathcal{F}_2$ into a $(t,p_1 \cdot p_2)$-FSS for the class of products (they term this "FSS product operator"). The main

difference between our construction and their transformation is that the number of parties in their transformation grows very quickly. This means that, even if combined with our idea of decomposing the point function $f_{x,v} : [N] \to \mathbb{F}$ to a product of d point functions with smaller domains, $f_{x_i,v_i} : [M] \to \mathbb{F}$, the [BGI16b] product will require number of parties which is exponential in d, while we only use $p = dt + 1$ parties. For example, in the case $t = 2$ we get, in Section 3.1 above, (2,5)-DPF with communication $O(N^{1/4})$, while combining two $(2, 3)$-DPFs, using [BGI16b], will result in a (2,9)-scheme (with communication $O(N^{1/4})$, using our decomposition). Also, [BGI18a, Thm. 3.23] can be viewed as a special case of our construction for $t = 1$; note that we focus in this paper on $t \geq 2$, as for $t = 1$ highly efficient DPF constructions are already known.

Determining the exact communication complexity of multi-party DPF, as a function of t and p, remains an intriguing open problem. This holds even in concrete special cases, such as $p = 4, t = 2$, where we do not know of a $(2, 4)$-DPF scheme with complexity $o(\sqrt{N})$ (while, as mentioned in the Introduction, $(2, 4)$-Binary-*CPIR* has a very efficient solution by combining DPF with [BIW07]).

4 1-out-of-3 CNF-DPF

In this section we present a 1-out-of-3 secure CNF-DPF protocol that achieves poly-log communication (in the domain size $N := |D|$). Our construction combines ideas from the original 1-out-of-2 protocol of [BGI15] with the 2-out-of-3 protocol of [BKKO20], whereby we seek to get the communication efficiency of the former, but extended to the 3-party setting as is treated by the latter. Before giving the formal presentation of our construction, we provide some insight on the main ideas of how we convert the $O(\sqrt{N})$ protocol of [BKKO20] into the poly-log(N) protocol presented below.

4.1 Overview of Construction

In [BKKO20], the Gen algorithm partitions the domain size N into $\sqrt{N}$ "blocks" of size $\sqrt{N}$, and to each block $1 \leq j \leq \sqrt{N}$, each key will be assigned a pair of PRG seeds $\{x_j, y_j\}$. In the notation below, the superscript $\mathcal{P}_i$ for $i \in [3]$ denotes the key index,[12] and $\mathcal{P}_R$ (respectively $\mathcal{P}_L$) refers to a PRG seed associated with a key to the "right" (respectively, to the "left") of key $\mathcal{P}$, where we view the key indices as a cycle: $\mathcal{P}_1 \to \mathcal{P}_2 \to \mathcal{P}_3 \to \mathcal{P}_1$, e.g. for $\mathcal{P} = \mathcal{P}_1$, we have: $\mathcal{P}_R = \mathcal{P}_2$ and $\mathcal{P}_L = \mathcal{P}_3$. Also, for the DPF scheme of [BKKO20], the terminology "on-block" index $j \in [\sqrt{N}]$ refers to the index of the unique block that contains $\alpha \in D$ that defines the point function. Similarly, in a binary tree partitioning of $[N]$ used in our construction, a node ν in this binary tree is "on-path" if the leaf-node with index $\alpha \in [N]$ is a descendent of ν.

[12] We write the key index as a superscript (not a subscript) to avoid confusion with the node index ν (denoted as a subscript). The choice of $\mathcal{P}$ over a simpler index $i \in [3]$ is to avoid confusion with an exponent (since it is a superscript), and the specific choice of "P" is for "Party," as FSS typically associates each key κ with a party $\mathcal{P}$.

With this notation, the PRG seeds in [BKKO20] satisfy:[13]

For *On-Block* Indices j	For *Off-Block* Indices j
$x_j^{\mathcal{P}_1} \neq x_j^{\mathcal{P}_2} \neq x_j^{\mathcal{P}_3}$	$x_j^{\mathcal{P}} = y_j^{\mathcal{P}_L}$
$y_j^{\mathcal{P}_1} = y_j^{\mathcal{P}_2} = y_j^{\mathcal{P}_3}$	$y_j^{\mathcal{P}} = x_j^{\mathcal{P}_R}$

(1)

In this paper, to avoid the $\sqrt{N}$ cost of dealing the seeds as per (1), as motivated by the paradigm of [BGI15] we "partition" the domain D via a binary tree, where the leaf-level has $N = |D|$ nodes. Then, instead of dealing the PRG seeds for *every* node in the binary tree, we only deal seeds at the root, and then describe a process (which uses extra auxiliary information dealt for each level) to generate PRG seeds for the rest of the nodes in the binary tree. This process is described formally in Section 4.3 below, but we mention here the important invariant that is maintained at every node ν in the binary tree:

For *On-Path* Nodes ν	For *Off-Path* Nodes ν
$x_\nu^{\mathcal{P}} = z_\nu^{\mathcal{P}_L}$	$x_\nu^{\mathcal{P}} = y_\nu^{\mathcal{P}_L} = z_\nu^{\mathcal{P}_R}$
$y_\nu^{\mathcal{P}_1} = y_\nu^{\mathcal{P}_2} = y_\nu^{\mathcal{P}_3}$	$y_\nu^{\mathcal{P}} = z_\nu^{\mathcal{P}_L} = x_\nu^{\mathcal{P}_R}$
$z_\nu^{\mathcal{P}} = x_\nu^{\mathcal{P}_R}$	$z_\nu^{\mathcal{P}} = x_\nu^{\mathcal{P}_L} = y_\nu^{\mathcal{P}_R}$

(2)

In comparing (2) to (1), notice first that instead of each key consisting of two PRG seeds, they each have *three* PRG seeds now: $\{x_j, y_j, z_j\}$. To clarify the nature of this extra seed, it will be convenient to (temporarily) modify the notation slightly: for an off-path block, in (1) the first key has PRG seeds $\{a, b\}$, the second key has seeds $\{b, c\}$, and the third key has seeds $\{c, a\}$.[14] So there are a total of three distinct seeds $\{a, b, c\}$ across all keys, and each key is missing exactly one of these three seeds for (1). Then the extra seed in each key of (2) is simply the third "missing seed."

Meanwhile, for the on-path block, in (1) the first key has PRG seeds $\{a, d\}$, the second key has seeds $\{b, d\}$, and the third key has seeds $\{c, d\}$.[15] So there are a total of *four* distinct seeds $\{a, b, c, d\}$ across all keys, with seed d being common to all three keys, and each of the other three seeds appearing in exactly one key. Thus, unlike in off-block positions where each key was missing *one* of the *three* seeds, in the on-block position each key is missing *two* of the *four* seeds. Then, in (2), each key is given one of the two missing seeds, namely the missing seed of the key on their "right." In sticking with the present notation, we can view the extra seed z included with each key as: $z^{\mathcal{P}_1} = b$, $z^{\mathcal{P}_2} = c$, and $z^{\mathcal{P}_3} = a$.

The two main points here are:

[13] The on-block property that seeds $\{x_j^{\mathcal{P}}\}_{\mathcal{P}}$ are not equal to each other, as described in (1), is intended to capture intuition. More formally, the requirement is that the on-block seeds $\{x_j^{\mathcal{P}}\}_{\mathcal{P}}$ are independent and (pseudo-)randomly generated.

[14] The overlapping nature of the PRG seeds, in a CNF format, is the important point; formally, to link the notations, set $a = x^{\mathcal{P}_1} = y^{\mathcal{P}_3}$, $b = x^{\mathcal{P}_2} = y^{\mathcal{P}_1}$, and $c = x^{\mathcal{P}_3} = y^{\mathcal{P}_2}$.

[15] The fact that there is one common seed "d" across all three keys, and that the other seeds are all distinct, is the important point here; formally, to link the two notations, set $a = x^{\mathcal{P}_1}$, $b = x^{\mathcal{P}_2}$, $c = x^{\mathcal{P}_3}$, and $d = y^{\mathcal{P}_1} = y^{\mathcal{P}_2} = y^{\mathcal{P}_3}$.

(i) Including an extra seed as part of the keys is necessary in order to iteratively generate the seeds on lower nodes in the binary tree. In [BKKO20], there was no iterative (tree) structure, but rather everything was flat: The domain D was partitioned into $\sqrt{N}$ blocks of $\sqrt{N}$ elements. But in following the binary tree approach of [BGI15] in attempt to minimize communication of the Gen algorithm, we need an iterative procedure to generate seeds on lower nodes in the binary tree. As in [BGI15], the difficult step is when the procedure attempts to specify the seeds on the children nodes of an on-path parent: one child node remains on-path, while the other becomes off-path. Maintaining the proper invariant (that on-path seeds should look like the left column of (2), while off-path seeds should look like the right column of (2)) will require keys to have partial information about the two "missing" seeds, which is why our algorithm provides one of the missing seeds as part of each key.

(ii) On the other hand, including one of the "missing" seeds as part of each key is exactly why the 2-out-of-3 security of [BKKO20] is reduced to 1-out-of-3 security in our protocol: If any two parties collude, they can easily link their own extra/missing seed that they were dealt with the node for which their partner also has that seed, and thus the secret path is revealed. However, even though this restricts our protocol to 1-out-of-3 security, we observe that providing one of the two "missing seeds" as part of each key is exactly the property we require for CNF sharing of the Gen keys.

Expanding more on (i) above, we provide an overview of how the two child nodes of an on-path node have correct values (i.e. values satisfying the invariant of (2)). Fix an on-path node μ on level l, and denote as the three sets of values on μ (as would be obtained by invoking the Eval algorithm using each of the three keys):

$$\begin{aligned} \kappa^{\mathcal{P}_1} \text{ seeds for on-path parent node } \mu\text{:} &\quad \{a, d, b\} \\ \kappa^{\mathcal{P}_2} \text{ seeds for on-path parent node } \mu\text{:} &\quad \{b, d, c\} \\ \kappa^{\mathcal{P}_3} \text{ seeds for on-path parent node } \mu\text{:} &\quad \{c, d, a\} \end{aligned} \tag{3}$$

where we have used in (3) that invariant (2) applies on the on-path node μ. Then in generating values on the children nodes of μ, the on-path child has keys:[16]

$$\begin{aligned} \kappa^{\mathcal{P}_1} \text{ seeds for } \mu\text{'s on-path child:} &\quad \{q \oplus G_*(a),\ q \oplus G_*(d),\ q \oplus G_*(b)\} \\ \kappa^{\mathcal{P}_2} \text{ seeds for } \mu\text{'s on-path child:} &\quad \{q \oplus G_*(b),\ q \oplus G_*(d),\ q \oplus G_*(c)\} \\ \kappa^{\mathcal{P}_3} \text{ seeds for } \mu\text{'s on-path child:} &\quad \{q \oplus G_*(c),\ q \oplus G_*(d),\ q \oplus G_*(a)\} \end{aligned} \tag{4}$$

where q is a random length-λ bit string and $G_* \in \{G_L, G_R\}$ (which of these G_* equals depends on whether the on-path child of μ is the left or right child). Meanwhile, the off-path child will have keys:

$$\begin{aligned} \kappa^{\mathcal{P}_1} \text{ seeds for } \mu\text{'s off-path child:} &\quad \{q \oplus G_*(b),\ q \oplus G_*(c),\ q \oplus G_*(a)\} \\ \kappa^{\mathcal{P}_2} \text{ seeds for } \mu\text{'s off-path child:} &\quad \{q \oplus G_*(c),\ q \oplus G_*(a),\ q \oplus G_*(b)\} \\ \kappa^{\mathcal{P}_3} \text{ seeds for } \mu\text{'s off-path child:} &\quad \{q \oplus G_*(a),\ q \oplus G_*(b),\ q \oplus G_*(c)\} \end{aligned} \tag{5}$$

[16] The formulas used for (4) and (5) come from (16), where we assumed "sibling control bit" values $b^{\mathcal{P}_1} = b^{\mathcal{P}_2} = b^{\mathcal{P}_3} = 0$ for the on-path child of μ, and that $b^{\mathcal{P}_1} = b^{\mathcal{P}_2} = b^{\mathcal{P}_3} = 1$ for the off-path child of μ. The other cases for valid sibling control bits would produce different key values, but the intuition for how values match or not is similar.

Notice that both (4) and (5) satisfy the appropriate invariant in (2). Also notice that the values in (4) can be generated directly from the same key's corresponding values on parent node μ (from (3)), whereas the values in (5) cannot (e.g. each of the new y values require knowledge of the "missing" seed value on parent node μ). Namely, the ability for each key to generate the center ("y") seed values as in (5) will come from extra information that is provided by the "Correction Word" component of each Gen key (see (7), and notice the $x^{\mathcal{P}_L}$ term, which to emphasize is *not* $x^{\mathcal{P}}$ but rather is the x seed value from the "left" key $\kappa^{\mathcal{P}_L}$, and this exactly corresponds to the "missing" seed value for key $\kappa^{\mathcal{P}}$).

4.2 Variants of DPF

We introduce (somewhat informally) a few variants of DPF that will be used as building blocks for our protocol below. Formal definitions, as well as concrete instantiations of these, can be found in the full version.

Definition 5 ***(Informal)*** *A 1-out-of-p Matching-Share DPF (MS-DPF) is defined analogously as ordinary DPF, except that instead of the requirement that $\sum_i \mathsf{Eval}(i, \kappa_i, \beta) = 0$ for every $\beta \neq \alpha$ in the domain of the point function $f_{\alpha,v}$, we require: $\forall \beta \neq \alpha : \mathsf{Eval}(1, \kappa_1, \beta) = \mathsf{Eval}(2, \kappa_2, \beta) = \cdots = \mathsf{Eval}(p, \kappa_p, \beta)$, where p is the number of parties.*

Remark 4. Note that MS-DPF as defined above is strictly speaking *not* FSS for the class of point functions: because all Eval shares match on every input $\beta \neq \alpha$, the actual function that an MS-DPF protocol represents looks random. However, based on the close relation to point functions (indeed, for the two-party case ($p = 2$) with $(\mathbb{G}, +) = (\mathbb{Z}_2^m, XOR)$, MS-DPF is identical to ordinary DPF), we stick with the "DPF" terminology.

Definition 6 ***(Informal)*** *A t-out-of-p DPF^+ is defined analogously as ordinary DPF, except that instead of the requirement that $\sum_i \mathsf{Eval}(i, \kappa_i, \alpha) = v$, for the point function $f_{\alpha,v}$, we have a concrete specification of the exact value of $\mathsf{Eval}(i, \kappa_i, \alpha)$ for each i. Namely, DPF^+ allows specification of p values $\{v_1, \ldots, v_p\}$, such that $\forall i : \mathsf{Eval}(i, \kappa_i, \alpha) = v_i$.*

Finally, we combine the two definitions above, of MS-DPF and DPF^+, and get:

Definition 7 ***(Informal)*** *A 1-out-of-p MS-DPF^+ scheme has Correctness properties: $\forall \beta \neq \alpha : \mathsf{Eval}(1, \kappa_1, \beta) = \mathsf{Eval}(2, \kappa_2, \beta) = \cdots = \mathsf{Eval}(p, \kappa_p, \beta)$, and meanwhile at the special input point $\alpha \in D$: $\forall i : \mathsf{Eval}(i, \kappa_i, \alpha) = v_i$.*

We use MS-DPF^+ in our $(1, 3)$-CNF-DPF construction (constructions of each of the DPF variants is straightforward; see the full version).

Claim 8 *Assuming OWF, there exists a $(1, 3)$-MS-DPF^+ scheme with communication $O(\lambda \log(N))$.*

4.3 Detailed Construction of 1-out-of-3 CNF-DPF

For any point function $f_{\alpha,v} \in \mathcal{F}$ with domain D (of size $N := |D|$) and range a finite abelian group[17] $(\mathbb{G}, +)$ (of size $m := |\mathbb{G}|$), we demonstrate the following:

Theorem 9 *Assuming OWF, there is a* $(1,3)$*-CNF-DPF scheme with communication* $O(m + \lambda \log^2(N))$.

We prove Theorem 9 constructively: the Gen and Eval algorithms are presented in this section, and Appendix A details the proof that the resulting scheme enjoys the stated complexity and satisfies the consistency, correctness, and security requirements of Definition 2. Our construction assumes the existence of a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+4}$ for security parameter λ, and a pseudorandom "convert" function $\widehat{G} : \{0,1\}^\lambda \to \mathbb{G}$. To fix notation, when G is applied to a seed x on a node μ, it stretches that seed to two new seeds plus four more bits, with one new seed and two bits going to each child node of node μ. To emphasize this, we write $G(x_\mu) = \Big(G_L(x_\mu), H_L(x_\mu), \widehat{H}_L(x_\mu)\Big), \Big(G_R(x_\mu), H_R(x_\mu), \widehat{H}_R(x_\mu)\Big)$, where $G_L, G_R : \{0,1\}^\lambda \to \{0,1\}^\lambda$ are zero stretch PRGs; and $H_L, \widehat{H}_L, H_R, \widehat{H}_R : \{0,1\}^\lambda \to \{0,1\}$ all output a single bit. The L and R subscripts on each PRG emphasize how the outputs of the PRGs will be applied, namely when generating values on the *Left* and *Right* child nodes of a given parent node μ.

Gen **Algorithm**.

1. Values at Root (level 0). At the root node of the tree, for each index $\mathcal{P} \in [3]$, choose PRG seeds $\{x^\mathcal{P}, y^\mathcal{P}, z^\mathcal{P}\}$ which are random subject to the constraints of the left (On-Path) column of (2). Our algorithm will also require that, for each key $\mathcal{P}$, each node ν in the binary tree has "control bits" $\{c_\nu^\mathcal{P}\}$ associated with it. These "on-path" control bits should appear random, subject to the constraint that they sum to one if and only if node ν is on-path. We will also associate a second set of control bits $\{b_\nu^\mathcal{P}\}$ to each node; these will satisfy a similar property as the "on-path" control bits, except with the condition that they sum to one if and only if ν's *sibling* is on-path (for the root node ν, which has no sibling, we demand the "sibling" control bits sum to zero). Thus, at the root node, also choose sibling control bits $\{b^\mathcal{P}\}$ and on-path control bits $\{c^\mathcal{P}\}$ which are random subject to the constraints of (6). Each key will actually include four total control bits (as these are CNF-shared across keys) at the root: $\{b^\mathcal{P}, b^{\mathcal{P}_R}, c^\mathcal{P}, c^{\mathcal{P}_R}\}$.

"Sibling" Control Bit b	**"On-Path" Control Bit** c
$\bigoplus_\mathcal{P} b_\nu^\mathcal{P} = \begin{cases} 0 & \text{if } \nu\text{'s } \textit{sibling} \text{ is } \textit{off-path} \\ 1 & \text{if } \nu\text{'s } \textit{sibling} \text{ is } \textit{on-path} \end{cases}$	$\bigoplus_\mathcal{P} c_\nu^\mathcal{P} = \begin{cases} 0 & \text{if } \nu \text{ is } \textit{off-path} \\ 1 & \text{if } \nu \text{ is } \textit{on-path} \end{cases}$

(6)

[17] For most applications, $\mathbb{G} = \mathbb{Z}_2^B$, so addition (XOR over a bitstring) and multiplication are defined. While Section 4.3 focuses on characteristic two groups, which covers the majority of applications in the literature, extending to arbitrary (finite, abelian) groups is straightforward (only (17) and the definition of final correction word W require modification). A demonstration of this fact is presented in the full version.

2. Correction Words. For each level $1 \le l \le \log(N)$ and each $\mathcal{P} \in [3]$, let $\{\kappa_l^{\mathcal{P}}\}$ denote the keys to a MS-DPF$^+$ protocol for $f_l = f_{(\alpha)_{l-1},\{v_l^{\mathcal{P}_1},v_l^{\mathcal{P}_2},v_l^{\mathcal{P}_3}\}}$, and let $\{\widehat{\kappa}_l^{\mathcal{P}}\}$ denote the keys to a MS-DPF$^+$ protocol for $\widehat{f}_l = \widehat{f}_{(\alpha)_l,\{\widehat{v}_l^{\mathcal{P}_1},\widehat{v}_l^{\mathcal{P}_2},\widehat{v}_l^{\mathcal{P}_3}\}}$, for functions f_l and $\widehat{f}_l$ defined as follows.[18] First, for each level l, the Gen algorithm will generate uniformly random λ-bit strings $\{p_l, q_l\}$. Then, if $\nu = \nu_l$ denotes the unique on-path node at level l, and $\mu = \mu_l$ denotes its parent node, then f_l is the MS-DPF$^+$ function:

$$f_{(\alpha)_{l-1},\{v_l^{\mathcal{P}_1},v_l^{\mathcal{P}_2},v_l^{\mathcal{P}_3}\}} : \{0,1\}^{l-1} \to \{0,1\}^{2\lambda}, \quad \text{with } v_l^{\mathcal{P}} = (v_L^{\mathcal{P}}, v_R^{\mathcal{P}}), \text{ where:}$$
$$(v_L^{\mathcal{P}}, v_R^{\mathcal{P}}) \;=\; \big(G_L(x_\mu^{\mathcal{P}_L}) \oplus (1 \oplus \alpha_l)\cdot(q_l \oplus p_l),\; G_R(x_\mu^{\mathcal{P}_L}) \oplus \alpha_l \cdot (q_l \oplus p_l)\big) \quad (7)$$

where $x_\mu^{\mathcal{P}_L}$ is the first on-path seed of $\mathcal{P}_L$ (see Step 3 below), and α_l is the l^{th} bit of α. Meanwhile, $\widehat{f}_l$ is the MS-DPF$^+$ function:

$$\widehat{f}_{(\alpha)_l,\{\widehat{v}_l^{\mathcal{P}_1},\widehat{v}_l^{\mathcal{P}_2},\widehat{v}_l^{\mathcal{P}_3}\}} : \{0,1\}^{l} \to \{0,1\}^{\lambda}, \quad \text{with } \widehat{v}_l^{\mathcal{P}} = \begin{cases} p_l & \text{if } b_\nu^{\mathcal{P}_L} = 0 \\ q_l & \text{if } b_\nu^{\mathcal{P}_L} = 1 \end{cases} \quad (8)$$

The above MS-DPF$^+$ keys will serve as the "correction words" for the PRG seeds. Notice that we use MS-DPF+ (instead of just dealing correction words directly as a common term across all three keys) because we require each key use a slightly different correction word. Indeed, as motivated in Section 4.1, the first correction word (corresponding to f_l and keys $\{\kappa_l^{\mathcal{P}}\}$) encodes the "missing" seed information that allows each key to overlap (as per CNF sharing) with the values from the other key(s). Meanwhile, the second correction word (corresponding to $\widehat{f}_l$ and keys $\{\widehat{\kappa}_l^{\mathcal{P}}\}$) ensures that for the next on-path node at the next level l, each key is still missing information on exactly one of the four distinct seeds on that node.

In addition to the correction words, the Gen algorithm will produce "correction bits," which will ensure correct (i.e. respecting (6)) values for $\{b^{\mathcal{P}}\}$ and $\{c^{\mathcal{P}}\}$ on each level. To simplify notation in the definition of the correction bits, we define the following values:[19]

$$\begin{aligned} h_L &:= H_L(x_\mu^{\mathcal{P}_1}) \oplus H_L(x_\mu^{\mathcal{P}_2}) \oplus H_L(x_\mu^{\mathcal{P}_3}) \oplus H_L(y_\mu^{\mathcal{P}_1}) \\ h_R &:= H_R(x_\mu^{\mathcal{P}_1}) \oplus H_R(x_\mu^{\mathcal{P}_2}) \oplus H_R(x_\mu^{\mathcal{P}_3}) \oplus H_R(y_\mu^{\mathcal{P}_1}) \\ \widehat{h}_L &:= \widehat{H}_L(x_\mu^{\mathcal{P}_1}) \oplus \widehat{H}_L(x_\mu^{\mathcal{P}_2}) \oplus \widehat{H}_L(x_\mu^{\mathcal{P}_3}) \oplus \widehat{H}_L(y_\mu^{\mathcal{P}_1}) \\ \widehat{h}_R &:= \widehat{H}_R(x_\mu^{\mathcal{P}_1}) \oplus \widehat{H}_R(x_\mu^{\mathcal{P}_2}) \oplus \widehat{H}_R(x_\mu^{\mathcal{P}_3}) \oplus \widehat{H}_R(y_\mu^{\mathcal{P}_1}) \end{aligned} \quad (9)$$

where μ denotes the on-path node on level $l-1$. Now, for each level l, each key will include four "correction bits" $\{r_l, s_l, t_l, u_l\}$, defined as follows:

[18] Recall that MS-DPF+ functions f_l and $\widehat{f}_l$ are *not* technically point functions (see Definition 5 and the ensuing remark). Also, for notation, $(\alpha)_{l-1}$ (as the special point in the domain of $f_l = f_{(\alpha)_l,\{v_l^{\mathcal{P}_1},v_l^{\mathcal{P}_2},v_l^{\mathcal{P}_3}\}}$) and $(\alpha)_l$ (for $\widehat{f}_l = \widehat{f}_{(\alpha)_l,\{\widehat{v}_l^{\mathcal{P}_1},\widehat{v}_l^{\mathcal{P}_2},\widehat{v}_l^{\mathcal{P}_3}\}}$) denote the first $l-1$ bits (respectively l bits) of α; whereas α_l (as it appears in (7) and (8)) denotes the l^{th} bit of α.

[19] For clarity, we suppress the level l in the subscript in the notation of (9).

$$r_l = \begin{cases} h_L & \text{if } \alpha_l = 0 \\ 1 \oplus h_L & \text{if } \alpha_l = 1 \end{cases} \qquad s_l = \begin{cases} 1 \oplus h_R & \text{if } \alpha_l = 0 \\ h_R & \text{if } \alpha_l = 1 \end{cases}$$
$$t_l = \begin{cases} 1 \oplus \widehat{h}_L & \text{if } \alpha_l = 0 \\ \widehat{h}_L & \text{if } \alpha_l = 1 \end{cases} \qquad u_l = \begin{cases} \widehat{h}_R & \text{if } \alpha_l = 0 \\ 1 \oplus \widehat{h}_R & \text{if } \alpha_l = 1 \end{cases} \tag{10}$$

3. Compute On-Path Seed Values. For each level l, use the correction words and bits (from the previous step) to generate seeds for the following level, as per the formulas in (15) and (16) (see Step 1c of the Eval Algorithm).
4. Final Correction Word. Define the final correction word $W := v \oplus_{\mathbb{G}} Q \in \mathbb{G}$, where v is the non-zero value of the target point function $f_{\alpha,v}$, and Q is:
$$Q := \widehat{G}(x_{\widehat{\nu}}^{\mathcal{P}_1}) \oplus_{\mathbb{G}} \widehat{G}(x_{\widehat{\nu}}^{\mathcal{P}_2}) \oplus_{\mathbb{G}} \widehat{G}(x_{\widehat{\nu}}^{\mathcal{P}_3}) \ominus_{\mathbb{G}} 3 \cdot \widehat{G}(y_{\widehat{\nu}}^{\mathcal{P}_1}) \ \in \mathbb{G}, \tag{11}$$
where $\ominus_{\mathbb{G}}$ denotes the negation of the group operation in $\mathbb{G}$,[20] and $\widehat{G} : \{0,1\}^\lambda \to \mathbb{G}$ is a map that converts a random λ-bit string into a pseudorandom group element in $\mathbb{G}$.

As the final output, for each $\mathcal{P} \in [3]$, the Gen algorithm outputs keys:
$$\kappa^{\mathcal{P}} := \Big(\{x^{\mathcal{P}}, y^{\mathcal{P}}, z^{\mathcal{P}}\},\ \{b^{\mathcal{P}}, b^{\mathcal{P}_R}, c^{\mathcal{P}}, c^{\mathcal{P}_R}\},\ W \\ \forall\, 1 \le l \le \log(N) :\ \{\kappa_l^{\mathcal{P}}\},\ \{\widehat{\kappa}_l^{\mathcal{P}}\},\ \{r_l, s_l, t_l, u_l\} \Big) \tag{12}$$

Eval **Algorithm**.

The $\mathsf{Eval}(\kappa^{\mathcal{P}}, i, \beta)$ algorithm is an iterative procedure where we start at the root of the binary tree, and define a procedure for traversing the tree (along the path of input $\beta \in D$)[21] whereby, at each step, we use the current node's values (plus the Gen key) to compute the values of the next node on the path. Formally, for any current node μ on level l of the path of β, with seed values $\{x_\mu^{\mathcal{P}}, y_\mu^{\mathcal{P}}, z_\mu^{\mathcal{P}}\}$ and (CNF-shared) control bits $\{b_\mu^{\mathcal{P}}, b_\mu^{\mathcal{P}_R}, c_\mu^{\mathcal{P}}, c_\mu^{\mathcal{P}_R}\}$, we demonstrate how to generate corresponding values for the next node ν on the path of β, corresponding to node μ's left or right child (depending on whether β_l is zero or one).

1. Traverse Tree per $\beta \in D$. For each level $1 \le l \le \log(N)$, let ν denote the current node on the path[22] of β at level l, and let μ denote ν's parent node. The previous iteration[23] of this step output values on parent node μ: $\{x_\mu^{\mathcal{P}}, y_\mu^{\mathcal{P}}, z_\mu^{\mathcal{P}}\}$ and $\{b_\mu^{\mathcal{P}}, b_\mu^{\mathcal{P}_R}, c_\mu^{\mathcal{P}}, c_\mu^{\mathcal{P}_R}\}$. Also, recall from (12) that for the current level l the Gen algorithm output the MS-DPF$^+$ keys $\kappa_l^{\mathcal{P}}$ and $\widehat{\kappa}_l^{\mathcal{P}}$; as well as the correction bits $\{r_l, s_l, t_l, u_l\}$. Output the following corresponding values for node ν as follows:

[20] For characteristic two groups, $\ominus_{\mathbb{G}} = \oplus_{\mathbb{G}}$; but we use this notation in (11) so as to minimize changes when we extend to arbitrary finite abelian groups $\mathbb{G}$.

[21] The binary representation $\beta = \beta_1 \beta_2 \ldots \beta_{\log(N)}$ of input $\beta \in D$ naturally defines a path down a binary tree (of depth $\log(N)$) by interpreting $\beta_l = 0$ to indicate going to the *left* child of the current node at level l, and moving right at level l if $\beta_l = 1$.

[22] Formally, if we index (0-based) the nodes on any level l, then the (binary representation of the) index of ν is: $\beta_1 \beta_2 \ldots \beta_l$.

[23] If $l = 1$ then μ is the root node and the values on μ are directly from the Gen key.

(a) Generating CNF-sharing of sibling control bits: $\{b_\nu^{\mathcal{P}}, b_\nu^{\mathcal{P}_R}\}$.
Set $\{b_\nu^{\mathcal{P}}, b_\nu^{\mathcal{P}_R}\}$ as follows:

$$b_\nu^{\mathcal{P}} = \begin{cases} c_\mu^{\mathcal{P}} \cdot r_l \oplus H_L(x_\mu^{\mathcal{P}}) \oplus H_L(y_\mu^{\mathcal{P}}) & \text{if } \nu \text{ is } \textit{left} \text{ child of } \mu \\ c_\mu^{\mathcal{P}} \cdot s_l \oplus H_R(x_\mu^{\mathcal{P}}) \oplus H_R(y_\mu^{\mathcal{P}}) & \text{if } \nu \text{ is } \textit{right} \text{ child of } \mu \end{cases}$$
$$b_\nu^{\mathcal{P}_R} = \begin{cases} c_\mu^{\mathcal{P}_R} \cdot r_l \oplus H_L(y_\mu^{\mathcal{P}}) \oplus H_L(z_\mu^{\mathcal{P}}) & \text{if } \nu \text{ is } \textit{left} \text{ child of } \mu \\ c_\mu^{\mathcal{P}_R} \cdot s_l \oplus H_R(y_\mu^{\mathcal{P}}) \oplus H_R(z_\mu^{\mathcal{P}}) & \text{if } \nu \text{ is } \textit{right} \text{ child of } \mu \end{cases} \quad (13)$$

(b) Generating CNF-sharing of on-path control bits: $\{c_\nu^{\mathcal{P}}, c_\nu^{\mathcal{P}_R}\}$.
Set $\{c_\nu^{\mathcal{P}}, c_\nu^{\mathcal{P}_R}\}$ as follows:

$$c_\nu^{\mathcal{P}} = \begin{cases} c_\mu^{\mathcal{P}} \cdot t_l \oplus \widehat{H}_L(x_\mu^{\mathcal{P}}) \oplus \widehat{H}_L(y_\mu^{\mathcal{P}}) & \text{if } \nu \text{ is } \textit{left} \text{ child of } \mu \\ c_\mu^{\mathcal{P}} \cdot u_l \oplus \widehat{H}_R(x_\mu^{\mathcal{P}}) \oplus \widehat{H}_R(y_\mu^{\mathcal{P}}) & \text{if } \nu \text{ is } \textit{right} \text{ child of } \mu \end{cases}$$
$$c_\nu^{\mathcal{P}_R} = \begin{cases} c_\mu^{\mathcal{P}_R} \cdot t_l \oplus \widehat{H}_L(y_\mu^{\mathcal{P}}) \oplus \widehat{H}_L(z_\mu^{\mathcal{P}}) & \text{if } \nu \text{ is } \textit{left} \text{ child of } \mu \\ c_\mu^{\mathcal{P}_R} \cdot u_l \oplus \widehat{H}_R(y_\mu^{\mathcal{P}}) \oplus \widehat{H}_R(z_\mu^{\mathcal{P}}) & \text{if } \nu \text{ is } \textit{right} \text{ child of } \mu \end{cases} \quad (14)$$

(c) Generating Seeds $\{x_\nu^{\mathcal{P}}, y_\nu^{\mathcal{P}}, z_\nu^{\mathcal{P}}\}$. First, to set notation: Let $G_* = G_L$ (respectively $G_* = G_R$) if ν is the *left* (respectively *right*) child of μ. Also, let $w_\nu^{\mathcal{P}} := \mathsf{Eval}(\kappa_l^{\mathcal{P}}, \mu)$ and let $\widehat{w}_\nu^{\mathcal{P}} := \mathsf{Eval}(\widehat{\kappa}_l^{\mathcal{P}}, \nu)$.[24] Recall from (7) that $w_\nu^{\mathcal{P}} \in \{0,1\}^{2\lambda}$, so let $w_*^{\mathcal{P}}$ be the first (respectively the last) λ bits of $w_\nu^{\mathcal{P}}$ if ν is the left (respectively right) child of its parent. We condition on the $\{b_\nu^{\mathcal{P}}, b_\nu^{\mathcal{P}_R}\}$ values that were output in Step 1a above:

Case I: $b_\nu^{\mathcal{P}} \neq b_\nu^{\mathcal{P}_R}$. Then set $\{x_\nu^{\mathcal{P}}, y_\nu^{\mathcal{P}}, z_\nu^{\mathcal{P}}\}$ as follows:

$$\begin{aligned} x_\nu^{\mathcal{P}} &= G_*(y_\mu^{\mathcal{P}}) \oplus b_\nu^{\mathcal{P}} \cdot (G_*(y_\mu^{\mathcal{P}}) \oplus w_*^{\mathcal{P}}) \oplus \widehat{w}_\nu^{\mathcal{P}} \\ y_\nu^{\mathcal{P}} &= G_*(x_\mu^{\mathcal{P}}) \oplus b_\nu^{\mathcal{P}} \cdot (G_*(x_\mu^{\mathcal{P}}) \oplus G_*(z_\mu^{\mathcal{P}})) \oplus \widehat{w}_\nu^{\mathcal{P}} \\ z_\nu^{\mathcal{P}} &= w_*^{\mathcal{P}} \oplus b_\nu^{\mathcal{P}} \cdot (w_*^{\mathcal{P}} \oplus G_*(y_\mu^{\mathcal{P}})) \oplus \widehat{w}_\nu^{\mathcal{P}} \end{aligned} \quad (15)$$

Case II: $b_\nu^{\mathcal{P}} = b_\nu^{\mathcal{P}_R}$. Then set $\{x_\nu^{\mathcal{P}}, y_\nu^{\mathcal{P}}, z_\nu^{\mathcal{P}}\}$ as follows:

$$\begin{aligned} x_\nu^{\mathcal{P}} &= G_*(x_\mu^{\mathcal{P}}) \oplus b_\nu^{\mathcal{P}} \cdot (G_*(x_\mu^{\mathcal{P}}) \oplus G_*(z_\mu^{\mathcal{P}})) \oplus \widehat{w}_\nu^{\mathcal{P}} \\ y_\nu^{\mathcal{P}} &= G_*(y_\mu^{\mathcal{P}}) \oplus b_\nu^{\mathcal{P}} \cdot (G_*(y_\mu^{\mathcal{P}}) \oplus w_*^{\mathcal{P}}) \oplus \widehat{w}_\nu^{\mathcal{P}} \\ z_\nu^{\mathcal{P}} &= G_*(z_\mu^{\mathcal{P}}) \oplus b_\nu^{\mathcal{P}} \cdot (G_*(z_\mu^{\mathcal{P}}) \oplus G_*(x_\mu^{\mathcal{P}})) \oplus \widehat{w}_\nu^{\mathcal{P}} \end{aligned} \quad (16)$$

2. Apply Final Correction Word. After terminating the above step at the leaf node ν on level $l = \log(N)$, the above iterative procedure has output values on ν: $\{x_\nu^{\mathcal{P}}, y_\nu^{\mathcal{P}}, z_\nu^{\mathcal{P}}\}$ and $\{b_\nu^{\mathcal{P}}, b_\nu^{\mathcal{P}_R}, c_\nu^{\mathcal{P}}, c_\nu^{\mathcal{P}_R}\}$. Then, as per the definition of $(1,3)$-CNF-FSS, $\mathsf{Eval}(\mathcal{P}, \kappa^{\mathcal{P}}, \beta)$ outputs $\binom{3-1}{1} = 2$ values in $\mathbb{G}$, which are:

$$\mathsf{Eval}(\mathcal{P}, \kappa^{\mathcal{P}}, \beta) := \left(\widehat{G}(x_\nu^{\mathcal{P}}) \ominus_{\mathbb{G}} \widehat{G}(y_\nu^{\mathcal{P}}) \oplus_{\mathbb{G}} c_\nu^{\mathcal{P}} \cdot W,\ \widehat{G}(z_\nu^{\mathcal{P}}) \ominus_{\mathbb{G}} \widehat{G}(y_\nu^{\mathcal{P}}) \oplus_{\mathbb{G}} c_\nu^{\mathcal{P}_R} \cdot W\right) \quad (17)$$

[24] Recall that $\{\kappa_l^{\mathcal{P}}, \widehat{\kappa}_l^{\mathcal{P}}\}$ were output as part of the Gen key, and they correspond to the MS-DPF$^+$ protocols described by (7) - (8). Also, notice that $w_\nu^{\mathcal{P}}$ comes from evaluating MS-DPF$^+$ key $\kappa_l^{\mathcal{P}}$ at point μ (the location of the *parent* node), whereas $\widehat{w}_\nu^{\mathcal{P}}$ comes from evaluating MS-DPF$^+$ key $\widehat{\kappa}_l^{\mathcal{P}}$ at point ν; this is why the domains of the two MS-DPF$^+$ functions $\{f_l, \widehat{f}_l\}$ differ by a factor of two (one extra bit for $\widehat{f}_l$).

References

[AFL+16] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *CCS*, pages 805–817. ACM Press, 2016.

[APY20] I. Abraham, B. Pinkas, and A. Yanai. Blinder: MPC based scalable and robust anonymous committed broadcast. In *CCS*. ACM Press, 2020.

[BCG+19] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO(3)*, pages 489–518. Springer, 2019.

[BGI15] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. In *EUROCRYPT*, pages 337–367. Springer, 2015.

[BGI16a] E. Boyle, N. Gilboa, and Y. Ishai. Breaking the circuit size barrier for secure computation under DDH. In *CRYPTO(1)*, pages 509–539. Springer, 2016.

[BGI16b] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing: Improvements and extensions. In *CCS*, pages 1292–1303. ACM Press, 2016.

[BGI18a] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing: Improvements and extensions. https://eprint.iacr.org/2018/707.pdf, 2018.

[BGI+18b] E. Boyle, N. Gilboa, Y. Ishai, H. Lin, and S. Tessaro. Foundations of homomorphic secret sharing. In *ITCS*, pages 21:1–21:21, 2018.

[BIK05] A. Beimel, Y. Ishai, and E. Kushilevitz. General constructions for information-theoretic private information retrieval. *J. Comput. Syst. Sci.*, 71(2):213–247, 2005.

[BIW07] O. Barkol, Y. Ishai, and E. Weinreb. On locally decodable codes, self-correctable codes, and t-private pir. In *APPROX-RANDOM*, pages 311–325. Springer, 2007.

[BKKO20] P. Bunn, J. Katz, E. Kushilevitz, and R. Ostrovsky. Efficient 3-party distributed ORAM. In *12th SCN*, pages 215–232. Springer, 2020.

[BL88] J.Cohen Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *CRYPTO*, pages 27–35. Springer, 1988.

[BLMR13] D. Boneh, K. Lewi, H.W. Montgomery, and A. Raghunathan. Key homomorphic PRFs and their applications. In *CRYPTO(1)*, pages 410–428. Springer, 2013.

[CBM15] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An anonymous messaging system handling millions of users. In *IEEE SP*, pages 321–338. IEEE Computer Society, 2015.

[CDI05] R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *TCC*, pages 342–362. Springer, 2005.

[DHRW16] Y. Dodis, S. Halevi, R. D. Rothblum, and D. Wichs. Spooky encryption and its applications. In *CRYPTO(3)*, pages 93–122. Springer, 2016.

[DIO98] G. Di-Crescenzo, Y. Ishai, and R. Ostrovsky. Universal service-providers for database private information retrieval. In *PODC*, pages 91–100. ACM Press, 1998.

[DS17] J. Doerner and A. Shelat. Scaling ORAM for secure computation. In *CCS*, pages 523–535. ACM Press, 2017.

[ECZB19] S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh. Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. *CoRR*, abs/1911.09215(v1), 2019.

[Efr09] E. Efremenko. 3-query locally decodable codes of subexponential length. In *STOC*, pages 39–44. ACM Press, 2009.

[FLNW17] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *EUROCRYPT(2)*, pages 225–255. Springer, 2017.

[FLO19] B. H. Falk, S. Lu, and R. Ostrovsky. Durasift: A robust, decentralized, encrypted database supporting private searches with complex policy controls. In *WPES-CCS*, pages 26–36, 2019.

[GI99] N. Gilboa and Y. Ishai. Compressing cryptographic resources. In *CRYPTO*, pages 591–608. Springer, 1999.

[GI14] N. Gilboa and Y. Ishai. Distributed point functions and their applications. In *EUROCRYPT*, pages 640–658. Springer, 2014.

[GKW18] S. D. Gordon, J. Katz, and X. Wang. Simple and efficient two-server ORAM. In *ASIACRYPT*, pages 141–157. Springer, 2018.

[Gol09] O. Goldreich. Foundations of cryptography: volume 2, basic applications. In *Cambridge university press*, 2009.

[HV20] A. Hamlin and M. Varia. Two-server distributed ORAM with sublinear computation and constant rounds. https://eprint.iacr.org/2020/1547, 2020.

[IKKP15] Y. Ishai, R. Kumaresan, E. Kushilevitz, and A. Paskin-Cherniavsky. Secure computation with minimal interaction, revisited. In *CRYPTO(2)*, pages 359–378. Springer, 2015.

[IKLO16] Y. Ishai, E. Kushilevitz, S. Lu, and R. Ostrovsky. Private large-scale databases with distributed searchable symmetric encryption. In *CT-RSA*, pages 90–107, 2016.

[ISN87] M. Ito, A. Saito, and T. Nishizeki. Secret sharing schemes realizing general access structure. In *Globecom*, pages 99–102. IEEE, 1987.

[JW18] S. Jarecki and B. Wei. 3pc ORAM with low latency, low bandwidth, and fast batch retrieval. https://eprint.iacr.org/2018/347.pdf, 2018.

[KM19] E. Kushilevitz and T. Mour. Sub-logarithmic distributed oblivious RAM with small block size. In *PKC*, pages 3–33, 2019.

[LO13] S. Lu and R. Ostrovsky. Distributed oblivious RAM for secure two-party computation. In *TCC*, volume 7785, pages 377–396. Springer, 2013.

[Mau02] U. Maurer. Secure multi-party computation made simple. In *SCN*, pages 14–28, 2002.

[OS97] R. Ostrovsky and V. Shoup. Private information storage. In *STOC*, pages 294–303. ACM Press, 1997.

[ZWR+16] S. Zahur, X. S. Wang, M. Raykova, A. Gascón, J. Doerner, D. Evans, and J. Katz. Revisiting square-root ORAM: Efficient random access in multi-party computation. In *IEEE Symposium on Security and Privacy*, pages 218–234. IEEE, 2016.

A Proof of Theorem 9

We argue how the scheme described in Section 4.3 enjoys the stated communication complexity and satisfies each of the requisite properties of CNF-DPF (see Definition 2).

Communication. The size of each Gen key is $O(m + \lambda \log^2(N))$:

– $O(\lambda)$ for each of the original PRG seeds $\{x^{\mathcal{P}}, y^{\mathcal{P}}, z^{\mathcal{P}}\}$.
– $O(1)$ for the four control bits on the root node $\{b^{\mathcal{P}}, b^{\mathcal{P}_R}, c^{\mathcal{P}}, c^{\mathcal{P}_R}\}$.
– $O(m)$ for the $W \in \mathbb{G}$ (recall $m = \log(|\mathbb{G}|)$).
– For each $1 \leq l \leq \log(N)$: $O(\lambda \log(N))$ for the collection of MS-DPF$^+$ keys $\{\kappa_l^{\mathcal{P}}, \widehat{\kappa}_l^{\mathcal{P}}\}$ (see Claim 8). Adding these costs for each level l yields total cost of these keys: $O(\lambda \log^2(N))$.

Consistency. That the protocol of Section 4.3 satisfies the Consistency property of CNF-FSS (see Definition 2) requires showing, among other things, that for each $\mathcal{P} \in [3]$ and for each $\widehat{P} := \mathcal{P}_R$, that the control bits observe CNF-sharing:

$$b_\nu^{\mathcal{P}_R} = b_\nu^{\widehat{\mathcal{P}}} \quad \text{and} \quad c_\nu^{\mathcal{P}_R} = c_\nu^{\widehat{\mathcal{P}}} \tag{18}$$

In other words, (18) is emphasizing that the formulas for $b_\nu^{\mathcal{P}_R}$ and $c_\nu^{\mathcal{P}_R}$ in (the bottom equations of) (13) and (14) generate the same bits as (the top equations of) the corresponding formulas for $b_\nu^{\widehat{\mathcal{P}}}$ and $c_\nu^{\widehat{\mathcal{P}}}$ in (13) and (14), for $\widehat{\mathcal{P}} = \mathcal{P}_R$. For example, when computing the bottom formulas of (13) and (14) for $\mathcal{P} = \mathcal{P}_1$, the values output there (which are for $\mathcal{P}_R = \mathcal{P}_2$) match the values that are output for key $\mathcal{P}_2$ in (the top part of) the equations (13) and (14).

We make an inductive argument to demonstrate CNF-sharing of the control bits (as per (18)) holds for all nodes ν. At the root, (18) is true by construction of values $\{b_\nu^{\mathcal{P}}\}$ and $\{c_\nu^{\mathcal{P}}\}$ in Step 1 of the Gen algorithm. Now for any non-root node ν, let μ denote its parent, and assume that (18); we use the formulas in (13) and (14) to demonstrate that (18) also holds for ν. To fix notation, fix $\mathcal{P} \in [3]$, and let $\widehat{\mathcal{P}} = \mathcal{P}_R$ denote the *right* key of $\mathcal{P}$.

Case 1: μ is *off-path*. In the Correctness argument above, we demonstrated that (2) is satisfied for the seeds on every node. Since μ is off-path: $x_\mu^{\mathcal{P}} = z_\mu^{\widehat{\mathcal{P}}}$, $y_\mu^{\mathcal{P}} = x_\mu^{\widehat{\mathcal{P}}}$, and $z_\mu^{\mathcal{P}} = y_\mu^{\widehat{\mathcal{P}}}$. Plugging in these relations into (13) for $b_\nu^{\mathcal{P}_R}$:

$$\begin{aligned}
\text{If } \nu \text{ is } \textit{left} \text{ child of } \mu\text{:} \quad b_\nu^{\mathcal{P}_R} &= c_\mu^{\mathcal{P}_R} \cdot r_l \oplus H_L(y_\mu^{\mathcal{P}}) \oplus H_L(z_\mu^{\mathcal{P}}) \\
&= c_\mu^{\widehat{\mathcal{P}}} \cdot r_l \oplus H_L(x_\mu^{\widehat{\mathcal{P}}}) \oplus H_L(y_\mu^{\widehat{\mathcal{P}}}) = b_\nu^{\widehat{\mathcal{P}}} \\
\text{If } \nu \text{ is } \textit{right} \text{ child of } \mu\text{:} \quad b_\nu^{\mathcal{P}_R} &= c_\mu^{\mathcal{P}_R} \cdot s_l \oplus H_R(y_\mu^{\mathcal{P}}) \oplus H_R(z_\mu^{\mathcal{P}}) \\
&= c_\mu^{\widehat{\mathcal{P}}} \cdot s_l \oplus H_R(x_\mu^{\widehat{\mathcal{P}}}) \oplus H_R(y_\mu^{\widehat{\mathcal{P}}}) = b_\nu^{\widehat{\mathcal{P}}}
\end{aligned}$$

where we have applied the inductive argument that $c_\mu^{\mathcal{P}_R} = c_\mu^{\widehat{\mathcal{P}}}$ for parent node μ for the center equality of each case above.

Case 2: μ is *on-path*. Since μ is on-path: $z_\mu^{\mathcal{P}} = x_\mu^{\widehat{\mathcal{P}}}$ and $y_\mu^{\mathcal{P}} = y_\mu^{\widehat{\mathcal{P}}}$. Plugging in these relations into (13) for $b_\nu^{\mathcal{P}_R}$:

$$\begin{aligned}
\text{If } \nu \text{ is } \textit{left} \text{ child of } \mu\text{:} \quad b_\nu^{\mathcal{P}_R} &= c_\mu^{\mathcal{P}_R} \cdot r_l \oplus H_L(y_\mu^{\mathcal{P}}) \oplus H_L(z_\mu^{\mathcal{P}}) \\
&= c_\mu^{\widehat{\mathcal{P}}} \cdot r_l \oplus H_L(y_\mu^{\widehat{\mathcal{P}}}) \oplus H_L(x_\mu^{\widehat{\mathcal{P}}}) = b_\nu^{\widehat{\mathcal{P}}} \\
\text{If } \nu \text{ is } \textit{right} \text{ child of } \mu\text{:} \quad b_\nu^{\mathcal{P}_R} &= c_\mu^{\mathcal{P}_R} \cdot s_l \oplus H_R(y_\mu^{\mathcal{P}}) \oplus H_R(z_\mu^{\mathcal{P}}) \\
&= c_\mu^{\widehat{\mathcal{P}}} \cdot s_l \oplus H_R(y_\mu^{\widehat{\mathcal{P}}}) \oplus H_R(x_\mu^{\widehat{\mathcal{P}}}) = b_\nu^{\widehat{\mathcal{P}}}
\end{aligned}$$

The argument that $c_\nu^{\mathcal{P}_R} = c_\nu^{\widehat{\mathcal{P}}}$ is similar, using t_l fori r_l, u_l for s_l, and $\widehat{H}$ for H.

With (18) verified, Consistency follows immediately from the invariants of (2), both for the case ν is on-path (i.e. $\beta = \alpha$) and off-path (i.e. $\beta \neq \alpha$); see (17).

Security. We provide a sketch of the proof here, which captures the intuition of the argument; the full proof is relegated to the extended version.

We argue that the components of any Gen key $\kappa^{\mathcal{P}}$(see (12)) are independent from each other and either truly random or masked with pseudorandom values whose seeds are known only to other parties (and not to party $\mathcal{P}$). In fact, the information of $\kappa^{\mathcal{P}}$ related to the root node is randomly chosen, and the information related to the other levels of the tree is masked using pseudorandom values not known to $\mathcal{P}$. Based on this, a simulator that simply outputs random values according to the key structure will satisfy Definition 2, which we recall here (updated for our case of security threshold $t = 1$):

$$\left\{\{\kappa_1, \ldots, \kappa_p\} \leftarrow_R \mathsf{Gen}(1^\lambda, f_{\alpha,v}) \ : \ \kappa_i\right\} \approx_C \left\{\kappa \leftarrow_R \mathsf{Sim}(1^\lambda, D, \mathbb{G})\right\}. \tag{19}$$

The proof follows an inductive argument (on the depth of the binary tree), and argues that assuming a simulator that outputs random values satisfies (19) for depth $l-1$, the extra values output by Gen in (12) for level l do not threaten the validity of the same simulator (i.e. one that is simply outputting random values) for the extra layer of the tree. More concretely, we will demonstrate the existence of a related simulator[25]:

$$\begin{aligned}\forall 1 \leq l \leq \log N: \quad &\left\{\{\kappa^{\mathcal{P}_1}, \kappa^{\mathcal{P}_2}, \kappa^{\mathcal{P}_3}\} \leftarrow_R \mathsf{Gen}(1^\lambda, f_{\alpha,v}) \ : \ ((\kappa^{\mathcal{P}})_l, \ x_{\nu_l}^{\mathcal{P}_L})\right\} \approx_C \\ &\left\{((\kappa)_l \leftarrow_R \mathsf{Sim}(1^\lambda, D, \mathbb{G}), \ x \leftarrow_R \{0,1\}^\lambda \ : \ ((\kappa)_l, \ x)\right\},\end{aligned} \tag{20}$$

where ν_l refers to the on-path node at level l, $(\kappa^{\mathcal{P}})_l$ refers to the components of key $\kappa^{\mathcal{P}}$ from Gen steps 1-3 through level l (i.e. everything from (12) *except* the final correction word W and the per-level values for levels in $[l+1..\log N]$), and $x_{\nu_l}^{\mathcal{P}_L}$ refer to the seed values x on node ν_l that are associated with the key $\kappa^{\mathcal{P}_L}$ to the *left* of the provided key $\kappa^{\mathcal{P}}$.[26] The reason that the existence of a simulator as per (20) (and more specifically, where this simulator simply outputs random values as per the structure of $(\kappa^{\mathcal{P}})_l$) implies the existence of a

[25] The existence of a simulator as in (20) is actually *stronger* than what we need to argue (19). Technically, it would be sufficient to argue the existence of a simulator:

$$\begin{aligned}\forall 1 \leq l \leq \log N: \quad &\left\{\{\kappa^{\mathcal{P}_1}, \kappa^{\mathcal{P}_2}, \kappa^{\mathcal{P}_3}\} \leftarrow_R \mathsf{Gen}(1^\lambda, f_{\alpha,v}) \ : \ ((\kappa^{\mathcal{P}})_l, \ x_{\nu_l}^{\mathcal{P}_L})\right\} \approx_C \\ &\left\{((\kappa)_l, \ x) \leftarrow_R \mathsf{Sim}(1^\lambda, D, \mathbb{G})\right\}\end{aligned}$$

and then to prove that $x_{\nu_l}^{\mathcal{P}_L}$ is (computationally) independent of $(\kappa^{\mathcal{P}})_l$. While this is possible, our proof demonstrates the existence of the stronger simulator of (20).

[26] Note that (20) is motivated by the CNF-sharing of the keys (or more precisely, the seeds), whereby each key $\kappa^{\mathcal{P}}$ has overlapping information from one of the other keys (in this case $\kappa^{\mathcal{P}_R}$), but is missing information from the third key (in this case $\kappa^{\mathcal{P}_L}$). In particular, this is why it is the seed of the *left* key $x^{\mathcal{P}_L}$ that is referenced in (20), as well as in (7) and (8).

simulator as per (19) is based on the formulas dictating how the Gen algorithm computes the extra seed values on level l: $\{\kappa_l^{\mathcal{P}}\}$, $\{\widehat{\kappa}_l^{\mathcal{P}}\}$, $\{r_l, s_l, t_l, u_l\}$. Namely, investigating the formulas for these extra values on level l ((7), (8), (9), and (10)), each formula has a term involving $x_\mu^{\mathcal{P}_L}$ for the value of $x^{\mathcal{P}_L}$ on node μ on level $l-1$, and consequently as long as the value of $x_\mu^{\mathcal{P}_L}$ on parent level $l-1$ cannot be distinguished from uniform, the new Gen key values on level l: $\{\kappa_l^{\mathcal{P}}\}$, $\{\widehat{\kappa}_l^{\mathcal{P}}\}$, $\{r_l, s_l, t_l, u_l\}$ will also be indistinguishable from uniform. Notice that for each $1 \leq l \leq \log N$, (20) explicitly excludes the final correction word W from both sides. However, the last step of the argument has the same spirit, whereby the existence of the $x_\nu^{\mathcal{P}_L}$ term (for on-path leaf node ν) in W implies that W is indistinguishible from uniform.

We proceed with an inductive argument, demonstrating that all the values output by the Gen algorithm respect the security invariant, and then demonstrate how the security invariant implies that all values output by the Gen algorithm appear uniformly random (and independent of one another).

- Step 1: Values at Root. The seeds $\{x^{\mathcal{P}}, y^{\mathcal{P}}, z^{\mathcal{P}}\}$ are chosen uniformly at random (subject to the constraint in (2), i.e. that there is a single common seed $y^{\mathcal{P}}$ that is common across all three keys, and that the other two seeds of each key overlap with exactly one of the seeds from each of the other two keys) and, in particular, the seeds are chosen independently from the point function $f_{\alpha,v}$ parameters, α and v. Similarly, the sibling control bits $\{b^{\mathcal{P}}, b^{\mathcal{P}_R}\}$ and on-path control bits $\{c^{\mathcal{P}}, c^{\mathcal{P}_R}\}$ are also chosen uniformly at random (subject to the constraint in (6)) and independently from the parameters α and v.
- Step 2.i: For each $1 \leq l \leq \log(N)$: MS-DPF$^+$ keys $\{\kappa_l^{\mathcal{P}}, \widehat{\kappa}_l^{\mathcal{P}}\}$.
  Note that the security of the underlying MS-DPF$^+$ schemes for f_l and $\widehat{f_l}$ ensure that $\{v_l^{\mathcal{P}_L}, v_l^{\mathcal{P}_R}\}$ and $\{\widehat{v}_l^{\mathcal{P}_L}, \widehat{v}_l^{\mathcal{P}_R}\}$ cannot be distinguished from random even for someone holding $(\kappa^{\mathcal{P}})_l$ (and thus holding $\kappa_l^{\mathcal{P}}$ and $\widehat{\kappa}_l^{\mathcal{P}}$, which in particular reveals $v_l^{\mathcal{P}} = (v_L^{\mathcal{P}}, v_R^{\mathcal{P}})$ and $\widehat{v}_l^{\mathcal{P}}$; see (7) - (8)). That $\widehat{v}_l^{\mathcal{P}}$ do not leak information about parameters α or v follows from the fact that (8) indicates that $\widehat{v}_l^{\mathcal{P}}$ is uniformly random. Meanwhile, that $v_l^{\mathcal{P}} = (v_L^{\mathcal{P}}, v_R^{\mathcal{P}})$ does not leak information about parameters α or v is argued as follows: For the base case ($l = 1$), the formula for $v^{\mathcal{P}_L}$ indicates dependence on $G_L(x_\mu^{\mathcal{P}_L})$ (respectively $v^{\mathcal{P}_R}$ depends on $G_R(x_\mu^{\mathcal{P}_L})$), where μ is the on-path node on the parent level, i.e. μ is the root node if $l = 1$. Since (as mentioned in Step 1 above) $x_\mu^{\mathcal{P}_L}$ cannot be distinguished from uniform by information in $\kappa^{\mathcal{P}}$, it follows that $v_l^{\mathcal{P}}$ also cannot be distinguished from uniform (also, pseudorandomness of $G = (G_L, G_R)$ implies there is no dependence on the two components $(v_L^{\mathcal{P}}, v_R^{\mathcal{P}})$ of $v_l^{\mathcal{P}}$). For the inductive case ($1 < l \leq \log N$), we follow the same argument, except now we use the Security Invariant (20) (plus pseudorandomness of the PRG G) inductively to argue that $x_\mu^{\mathcal{P}_L}$ from the parent level $l-1$ cannot be distinguished from uniform, and therefore $v_l^{\mathcal{P}}$ also appears uniformly random.
- Step 2.ii: For each $1 \leq l \leq \log(N)$: Correction Bits $\{r_l, s_l, t_l, u_l\}$.
  As can be seen in (10), each correction bit depends on one of the values $\{h_L, h_R, \widehat{h}_L, \widehat{h}_R\}$, and, as per (9), each of these values in turn appears uni-

formly random due to its dependence on $x_{\mu}^{\mathcal{P}_L}$ for parent node μ (as was argued above in Step 2.i). Furthermore, pseudorandomness of $H, \widehat{H}$ implies that there is no dependency between the correction bit values and any other values dealt as part of the Gen key $\kappa^{\mathcal{P}}$.

– Step 3: Final Correction Word W.
While $W = v \oplus \widehat{G}(x_{\widehat{\nu}}^{\mathcal{P}_1}) \oplus \widehat{G}(x_{\widehat{\nu}}^{\mathcal{P}_2}) \oplus \widehat{G}(x_{\widehat{\nu}}^{\mathcal{P}_3}) \oplus \widehat{G}(y_{\widehat{\nu}}^{\mathcal{P}_1})$ involves the secret parameter v, the Security Invariant applied to on-path leaf node $\widehat{\nu}$ implies that W contains a term $(x_{\widehat{\nu}}^{\mathcal{P}_L})$ that cannot be distinguished from random by $\mathcal{P}$, and therefore v remains completely hidden. Furthermore, pseudorandomness of $\widehat{G}$ implies that there is no dependency between W and any other values dealt as part of the Gen key $\kappa^{\mathcal{P}}$.

Correctness. We demonstrate for any input $\beta \in D$ and for each $\mathcal{P} \in [3]$:

$$\sum_{\mathcal{P}} \mathsf{Eval}(\mathcal{P}, \kappa^{\mathcal{P}}, \beta) = \begin{cases} (0_{\mathbb{G}}, 0_{\mathbb{G}}) & \text{if } \beta \neq \alpha \\ (v, v) & \text{if } \beta = \alpha \end{cases} \tag{21}$$

(Recall that in a $(1,3)$-CNF scheme, Eval outputs for each party a pair of values, one per key, and the sum of all left values and the sum of all right values should both equal $f(\beta)$, which for DPF is either $0_{\mathbb{G}}$ or v, depending on whether input β equals α.) To show (21) holds, we first show that at every iteration of Step 1 of the Eval procedure, that the values $\{x_\nu^{\mathcal{P}}, y_\nu^{\mathcal{P}}, z_\nu^{\mathcal{P}}\}$ and $\{b_\nu^{\mathcal{P}}, b_\nu^{\mathcal{P}_R}, c_\nu^{\mathcal{P}}, c_\nu^{\mathcal{P}_R}\}$ respect the invariants listed in tables (2) and (6), respectively. Then, once this is shown, (21) follows immediately since:

First coordinate of $\bigoplus_{\mathcal{P}} \mathbb{G}\, \mathsf{Eval}(\mathcal{P}, \kappa^{\mathcal{P}}, \beta)$:

$$\begin{aligned}
&= \bigoplus_{\mathcal{P}} \mathbb{G} \left(\widehat{G}(x_\nu^{\mathcal{P}}) \ominus_{\mathbb{G}} \widehat{G}(y_\nu^{\mathcal{P}}) \oplus_{\mathbb{G}} c_\nu^{\mathcal{P}} \cdot W \right) \\
&= \left(\left(\widehat{G}(x_\nu^{\mathcal{P}_1}) \ominus_{\mathbb{G}} \widehat{G}(y_\nu^{\mathcal{P}_1}) \right) \oplus_{\mathbb{G}} \left(\widehat{G}(x_\nu^{\mathcal{P}_2}) \ominus_{\mathbb{G}} \widehat{G}(y_\nu^{\mathcal{P}_2}) \right) \oplus_{\mathbb{G}} \left(\widehat{G}(x_\nu^{\mathcal{P}_3}) \ominus_{\mathbb{G}} \widehat{G}(y_\nu^{\mathcal{P}_3}) \right) \right) \oplus_{\mathbb{G}} \\
&\qquad W \cdot \bigoplus_{\mathcal{P}} \mathbb{G}\, c_\nu^{\mathcal{P}} \\
&= \left(\left(\widehat{G}(x_\nu^{\mathcal{P}_1}) \ominus_{\mathbb{G}} \widehat{G}(y_\nu^{\mathcal{P}_1}) \right) \oplus_{\mathbb{G}} \left(\widehat{G}(x_\nu^{\mathcal{P}_2}) \ominus_{\mathbb{G}} \widehat{G}(y_\nu^{\mathcal{P}_2}) \right) \oplus_{\mathbb{G}} \left(\widehat{G}(x_\nu^{\mathcal{P}_3}) \ominus_{\mathbb{G}} \widehat{G}(y_\nu^{\mathcal{P}_3}) \right) \right) \oplus_{\mathbb{G}} \\
&\qquad (v \oplus_{\mathbb{G}} Q) \cdot \bigoplus_{\mathcal{P}} \mathbb{G}\, c_\nu^{\mathcal{P}}
\end{aligned} \tag{22}$$

Notice from (2) that:

$$\begin{aligned}
&\left(\widehat{G}(x_\nu^{\mathcal{P}_1}) \ominus_{\mathbb{G}} \widehat{G}(y_\nu^{\mathcal{P}_1}) \right) \oplus_{\mathbb{G}} \left(\widehat{G}(x_\nu^{\mathcal{P}_2}) \ominus_{\mathbb{G}} \widehat{G}(y_\nu^{\mathcal{P}_2}) \right) \oplus_{\mathbb{G}} \left(\widehat{G}(x_\nu^{\mathcal{P}_3}) \ominus_{\mathbb{G}} \widehat{G}(y_\nu^{\mathcal{P}_3}) \right) \\
&\quad = \begin{cases} \widehat{G}(x_\nu^{\mathcal{P}_1})) \oplus_{\mathbb{G}} \widehat{G}(x_\nu^{\mathcal{P}_2})) \oplus_{\mathbb{G}} \widehat{G}(x_\nu^{\mathcal{P}_3})) \ominus_{\mathbb{G}} 3 \cdot \widehat{G}(y_\nu^{\mathcal{P}_1})) = Q & \text{if } \beta = \alpha \\ 0_{\mathbb{G}} & \text{if } \beta \neq \alpha \end{cases}
\end{aligned}$$

Also, notice that (6) implies that[27]:

$$\bigoplus_{\mathcal{P}} \mathbb{G}\, c_\nu^{\mathcal{P}} = \begin{cases} 1 & \text{if } \widehat{\nu} = \nu \text{ is } \textit{on-path} \;\Leftrightarrow\; \beta = \alpha \\ 0 & \text{if } \widehat{\nu} \neq \nu \text{ is } \textit{off-path} \;\Leftrightarrow\; \beta \neq \alpha \end{cases} \tag{23}$$

[27] (23) assumes $\mathbb{G}$ has characteristic two, so that (6), which is for XOR, is valid for $\oplus_{\mathbb{G}}$.

Thus (22) becomes:

$$\text{First coordinate of } \bigoplus_{\mathcal{P}}{}_{\mathbb{G}} \mathsf{Eval}(\mathcal{P}, \kappa^{\mathcal{P}}, \beta) :$$
$$= \begin{cases} Q \oplus_{\mathbb{G}} (v \oplus_{\mathbb{G}} Q) \cdot 1 \;=\; v & \text{if } \beta = \alpha \\ 0_{\mathbb{G}} \oplus_{\mathbb{G}} (v \oplus_{\mathbb{G}} Q) \cdot 0 \;=\; 0_{\mathbb{G}} & \text{if } \beta \neq \alpha \end{cases} \tag{24}$$

Meanwhile, the case for the second coordinate of $\sum_{\mathcal{P}} \mathsf{Eval}(\mathcal{P}, \kappa^{\mathcal{P}}, \beta)$ is similar, since the $\{c_\nu^{\mathcal{P}_R}\}$ obey (6) in the same way that $\{c_\nu^{\mathcal{P}}\}$ do, and the symmetry (in terms of (2)) of each key's first two PRG seeds $\{x_\nu^{\mathcal{P}}, y_\nu^{\mathcal{P}}\}$ and each key's second two PRG seeds $\{y_\nu^{\mathcal{P}}, z_\nu^{\mathcal{P}}\}$.

Thus, it remains to show that the invariants of (2) and (6) apply at every node in the binary tree. We argue this fact recursively, by demonstrating that as long as the invariants (2) and (6) hold on a parent node μ, then these invariants will continue to hold for both of μ's children. We kick off the recursive argument by noting that the root note (which is necessarily on-path) satisfies (2) and (6) by construction (see Step 1 of the Gen algorithm). For the inductive step, consider an arbitrary node ν on level $1 \leq l \leq \log(N)$, and let μ denote ν's parent. We do a case analysis based on whether ν is the left or right child of μ:

Case 1: ν is the left child of μ.

Sibling Control Bits $\{b_\nu^{\mathcal{P}}\}$.
Looking at formula (13) for generating the sibling control bits $\{b_\nu^{\mathcal{P}}, b_\nu^{\mathcal{P}_R}\}$ on ν:

$$\begin{aligned} \sum_{\mathcal{P}} b_\nu^{\mathcal{P}} &= \sum_{\mathcal{P}} \left(c_\mu^{\mathcal{P}} \cdot r_l \oplus H_L(x_\mu^{\mathcal{P}}) \oplus H_L(y_\mu^{\mathcal{P}})\right) \\ &= r_l \cdot \sum_{\mathcal{P}} c_\mu^{\mathcal{P}} \quad \oplus \\ &\quad \left(\left(H_L(x_\mu^{\mathcal{P}_1}) \oplus H_L(y_\mu^{\mathcal{P}_1})\right) \oplus \left(H_L(x_\mu^{\mathcal{P}_2}) \oplus H_L(y_\mu^{\mathcal{P}_2})\right) \oplus \left(H_L(x_\mu^{\mathcal{P}_3}) \oplus H_L(y_\mu^{\mathcal{P}_3})\right)\right) \\ &= \begin{cases} r_l \oplus h_L \text{ if } \mu \text{ is } \textit{on-path} \\ 0 \qquad\quad \text{if } \mu \text{ is } \textit{off-path} \end{cases} \end{aligned} \tag{25}$$

where we have used in (25) that $\sum_{\mathcal{P}} c_\mu^{\mathcal{P}} = 1$ if parent node μ is *on-path* and otherwise the sum equals zero (as per (6)); and from (2) that:

$$\begin{aligned} &\left(H_L(x_\mu^{\mathcal{P}_1}) \oplus H_L(y_\mu^{\mathcal{P}_1})\right) \oplus \left(H_L(x_\mu^{\mathcal{P}_2}) \oplus H_L(y_\mu^{\mathcal{P}_2})\right) \oplus \left(H_L(x_\mu^{\mathcal{P}_3}) \oplus H_L(y_\mu^{\mathcal{P}_3})\right) \\ &\quad = \begin{cases} H_L(x_\mu^{\mathcal{P}_1})) \oplus H_L(x_\mu^{\mathcal{P}_2})) \oplus H_L(x_\mu^{\mathcal{P}_3})) \oplus H_L(y_\mu^{\mathcal{P}_1})) = h_L & \text{if } \mu \text{ is } \textit{on-path} \\ 0 & \text{if } \mu \text{ is } \textit{off-path} \end{cases} \end{aligned}$$

Thus, if μ is off-path, then both ν and its sibling are also off-path, and $\{b_\nu^{\mathcal{P}}\}$ satisfies the requisite property of (6). Meanwhile, if μ is on-path, then exactly one of ν or its sibling is on-path. Since we are in the case that ν is the *left* child of μ, then ν is on-path if and only if $\alpha_l = 0$. In particular if μ is on-path:

$$\sum_{\mathcal{P}} b_\nu^{\mathcal{P}} = r_l \oplus h_L = \begin{cases} h_L \oplus h_L = 0 & \text{if } \alpha_l = 0 \;\Leftrightarrow \nu\text{'s } \textit{sibling} \text{ is off-path} \\ 1 \oplus h_L \oplus h_L = 1 & \text{if } \alpha_l = 1 \;\Leftrightarrow \nu\text{'s } \textit{sibling} \text{ is on-path} \end{cases}$$

where we used (10) to replace r_l conditioned on whether α_l is 0 or 1. The argument for the "right" sibling control bits $\{b_\nu^{\mathcal{P}_R}\}$ mirrors the above argument, since $\sum_{\mathcal{P}} c_\mu^{\mathcal{P}} = \sum_{\mathcal{P}} c_\mu^{\mathcal{P}_R}$ (per (18)) and $\{(x_\mu^{\mathcal{P}}, y_\mu^{\mathcal{P}})\}_{\mathcal{P}} = \{(y_\mu^{\mathcal{P}}, z_\mu^{\mathcal{P}})\}_{\mathcal{P}}$ (per (2)).

On-Path Control Bits $\{c_\nu^{\mathcal{P}}\}$.
Looking at formula (14) for generating the on-path control bits $\{c_\nu^{\mathcal{P}}, c_\nu^{\mathcal{P}_R}\}$ on ν:

$$\begin{aligned}\sum_{\mathcal{P}} c_\nu^{\mathcal{P}} &= \sum_{\mathcal{P}} \left(c_\mu^{\mathcal{P}} \cdot t_l \oplus \widehat{H}_L(x_\mu^{\mathcal{P}}) \oplus \widehat{H}_L(y_\mu^{\mathcal{P}}) \right) \\ &= t_l \cdot \sum_{\mathcal{P}} c_\mu^{\mathcal{P}} \quad \oplus \\ &\quad \left(\left(\widehat{H}_L(x_\mu^{\mathcal{P}_1}) \oplus \widehat{H}_L(y_\mu^{\mathcal{P}_1}) \right) \oplus \left(\widehat{H}_L(x_\mu^{\mathcal{P}_2}) \oplus \widehat{H}_L(y_\mu^{\mathcal{P}_2}) \right) \oplus \left(\widehat{H}_L(x_\mu^{\mathcal{P}_3}) \oplus \widehat{H}_L(y_\mu^{\mathcal{P}_3}) \right) \right) \\ &= \begin{cases} t_l \oplus \widehat{h}_L & \text{if } \mu \text{ is } \textit{on-path} \\ 0 & \text{if } \mu \text{ is } \textit{off-path} \end{cases} \end{aligned} \tag{26}$$

where we have used in (26) that $\sum_{\mathcal{P}} c_\mu^{\mathcal{P}} = 1$ if parent node μ is *on-path* and otherwise the sum equals zero (per (6)); and from (2) that:

$$\begin{aligned}&\left(\widehat{H}_L(x_\mu^{\mathcal{P}_1}) \oplus \widehat{H}_L(y_\mu^{\mathcal{P}_1}) \right) \oplus \left(\widehat{H}_L(x_\mu^{\mathcal{P}_2}) \oplus \widehat{H}_L(y_\mu^{\mathcal{P}_2}) \right) \oplus \left(\widehat{H}_L(x_\mu^{\mathcal{P}_3}) \oplus \widehat{H}_L(y_\mu^{\mathcal{P}_3}) \right) \\ &= \begin{cases} \widehat{H}_L(x_\mu^{\mathcal{P}_1})) \oplus \widehat{H}_L(x_\mu^{\mathcal{P}_2})) \oplus \widehat{H}_L(x_\mu^{\mathcal{P}_3})) \oplus \widehat{H}_L(y_\mu^{\mathcal{P}_1})) = \widehat{h}_L & \text{if } \mu \text{ is } \textit{on-path} \\ 0 & \text{if } \mu \text{ is } \textit{off-path} \end{cases} \end{aligned}$$

Thus, if μ is off-path, then both ν and its sibling are also off-path, and $\{c_\nu^{\mathcal{P}}\}$ satisfies the requisite property of (6). Meanwhile, if μ is on-path, then exactly one of ν or its sibling is on-path. Since we are in the case that ν is the *left* child of μ, then ν is on-path if and only if $\alpha_l = 0$. In particular if μ is on-path:

$$\sum_{\mathcal{P}} c_\nu^{\mathcal{P}} = t_l \oplus \widehat{h}_L = \begin{cases} 1 \oplus \widehat{h}_L \oplus \widehat{h}_L = 1 & \text{if } \alpha_l = 0 \ \Leftrightarrow \nu \text{ is on-path} \\ \widehat{h}_L \oplus \widehat{h}_L = 0 & \text{if } \alpha_l = 1 \ \Leftrightarrow \nu \text{ is off-path} \end{cases}$$

where we used (10) to replace t_l conditioned on whether $\alpha_l = 0$ or $\alpha_l = 1$. The argument for the "right" sibling control bits $\{c_\nu^{\mathcal{P}_R}\}$ mirrors the above argument, since $\sum_{\mathcal{P}} c_\mu^{\mathcal{P}} = \sum_{\mathcal{P}} c_\mu^{\mathcal{P}_R}$ (per (18)) and $\{(x_\mu^{\mathcal{P}}, y_\mu^{\mathcal{P}})\}_{\mathcal{P}} = \{(y_\mu^{\mathcal{P}}, z_\mu^{\mathcal{P}})\}_{\mathcal{P}}$ (per (2)).

Seeds $\{x_\nu^{\mathcal{P}}, y_\nu^{\mathcal{P}}, z_\nu^{\mathcal{P}}\}$.
Demonstrating that the formulas for the next-level seeds in (15) - (16) maintain the seed invariants of (2) is straightforward, but requires a case analysis based on whether the current node is on-path or off-path.

Case Analysis of Correctness for 1-out-of-3 CNF-DPF.
We prove the new seed values on ν, computed as per (15)-(16), obey (2) by doing a case analysis, broken down by ν's location (on-path, sibling is on-path, both self and sibling are off-path), as well as on the $\{b_\nu^{\mathcal{P}}, b_\nu^{\mathcal{P}_R}\}$ values on ν. Before proceeding, recall the notation for $w_*^{\mathcal{P}}$ (see Step (1c) of the Eval algorithm): the first (respectively last) λ bits of $\mathsf{Eval}(\kappa_l^{\mathcal{P}}, \nu)$ if ν is the left (respectively right)

child of its parent, where $\kappa_l^{\mathcal{P}}$ denotes the MS-DPF$^+$ key for level l (see (7) in Step 2 of the Gen algorithm); and also the notation for $\widehat{w}_\nu^{\mathcal{P}} = \mathsf{Eval}(\widehat{\kappa}_l^{\mathcal{P}}, \nu)$, and for $G_* = G_L$ (resp. G_R) if ν is the *left* (resp. *right*) child of its parent.

For each case below, we present a table which shows what each key's new seed values on node ν will be, given ν's position (on/off path) and the seed values that were present on ν's parent node μ. The tables indicate, for each key, which seed formula ((15) vs. (16)) are used to derive the new seed values on ν.

Case A: Parent μ is off-path. Because parent node μ is off-path, its position (at depth $l-1$) does *not* correspond to the DPF index $(\alpha)_{l-1}$ of MS-DPF$^+$ function f_l; and similarly, neither of its children nodes are at position $(\alpha)_l$, and therefore they do not correspond to the DPF index of $\widehat{f}_l$. Therefore, $w_*^{\mathcal{P}_1} = w_*^{\mathcal{P}_2} = w_*^{\mathcal{P}_3}$ and $\widehat{w}_\nu^{\mathcal{P}_1} = \widehat{w}_\nu^{\mathcal{P}_2} = \widehat{w}_\nu^{\mathcal{P}_3}$ (by definition of f_l and $\widehat{f}_l$; see Step 2 of the Gen algorithm), and so we suppress player superscripts and write simply w_* and $\widehat{w}_\nu$. Also, since μ is off-path, the seeds on μ satisfy invariant (2), and for convenience we will denote the three keys' seeds on off-path parent node μ as: $\kappa^{\mathcal{P}_1} = \{a, b, c\}$, $\kappa^{\mathcal{P}_2} = \{b, c, a\}$, $\kappa^{\mathcal{P}_3} = \{c, a, b\}$. Finally, since μ is off-path, so is ν and its sibling, and thus by the invariant of (6), we have that $\bigoplus_{\mathcal{P}} b_\nu^{\mathcal{P}} = 0$. Thus, there are four possibilities for the values of $(b_\nu^{\mathcal{P}_1}, b_\nu^{\mathcal{P}_2}, b_\nu^{\mathcal{P}_3})$: $(0,0,0)$, $(0,1,1)$, $(1,0,1)$, or $(1,1,0)$. We do a case-analysis just of the first two; the latter two are similar to the second:

Case A.1: $\{b_\nu^{\mathcal{P}}\} = (0,0,0)$:

	$\kappa^{\mathcal{P}_1}$ (via (16))	$\kappa^{\mathcal{P}_2}$ (via (16))	$\kappa^{\mathcal{P}_3}$ (via (16))
$x_\nu^{\mathcal{P}}$	$G_*(a) \oplus \widehat{w}_\nu$	$G_*(b) \oplus \widehat{w}_\nu$	$G_*(c) \oplus \widehat{w}_\nu$
$y_\nu^{\mathcal{P}}$	$G_*(b) \oplus \widehat{w}_\nu$	$G_*(c) \oplus \widehat{w}_\nu$	$G_*(a) \oplus \widehat{w}_\nu$
$z_\nu^{\mathcal{P}}$	$G_*(c) \oplus \widehat{w}_\nu$	$G_*(a) \oplus \widehat{w}_\nu$	$G_*(b) \oplus \widehat{w}_\nu$

Case A.2: $\{b_\nu^{\mathcal{P}}\} = (0,1,1)$:

	$\kappa^{\mathcal{P}_1}$ (via (15))	$\kappa^{\mathcal{P}_2}$ (via (16))	$\kappa^{\mathcal{P}_3}$ (via (15))
$x_\nu^{\mathcal{P}}$	$G_*(b) \oplus \widehat{w}_\nu$	$G_*(a) \oplus \widehat{w}_\nu$	$w_* \oplus \widehat{w}_\nu$
$y_\nu^{\mathcal{P}}$	$G_*(a) \oplus \widehat{w}_\nu$	$w_* \oplus \widehat{w}_\nu$	$G_*(b) \oplus \widehat{w}_\nu$
$z_\nu^{\mathcal{P}}$	$w_* \oplus \widehat{w}_\nu$	$G_*(b) \oplus \widehat{w}_\nu$	$G_*(a) \oplus \widehat{w}_\nu$

Case B: Parent μ is on-path; ν is on-path. Because parent node μ is on-path, its position (at depth $l-1$) corresponds to the DPF index $(\alpha)_{l-1}$ of MS-DPF$^+$ function f_l; and similarly ν on-path means that its position is $(\alpha)_l$ which corresponds to the DPF index of $\widehat{f}_l$. Therefore, $w_*^{\mathcal{P}}$ follows (7) and $\widehat{w}_\nu^{\mathcal{P}}$ follows (8):

$$w_*^{\mathcal{P}} = \begin{cases} v_L^{\mathcal{P}} = G_L(x_\mu^{\mathcal{P}_L}) \oplus q_l \oplus p_l & \text{if } \nu \text{ is the } \textit{left} \text{ child} \\ v_R^{\mathcal{P}} = G_R(x_\mu^{\mathcal{P}_L}) \oplus q_l \oplus p_l & \text{if } \nu \text{ is the } \textit{right} \text{ child} \end{cases} \tag{27}$$

$$\widehat{w}_\nu^{\mathcal{P}} = \widehat{v}_l^{\mathcal{P}} = \begin{cases} p_l & \text{if } b_\nu^{\mathcal{P}} = 0 \\ q_l & \text{if } b_\nu^{\mathcal{P}} = 1 \end{cases} \tag{28}$$

where $\{p_l, q_l\}$ are uniform random values chosen for each level $1 \le l \le \log(N)$, and we have used that, since ν is on-path, then $\alpha_l = 1$ (respectively $\alpha_l = 0$) when ν is the *left* child (respectively *right* child) of μ. Also, since μ is on-path, the seeds on μ satisfy invariant (2), and for convenience we will denote the three keys' seeds on on-path parent node μ as: $\kappa^{\mathcal{P}_1} = \{a, d, b\}$, $\kappa^{\mathcal{P}_2} = \{b, d, c\}$,

$\kappa^{\mathcal{P}_3} = \{c, d, a\}$. Finally, since ν is on-path, its sibling is off-path, and thus by the invariant of (6), we have that $\bigoplus_{\mathcal{P}} b_\nu^{\mathcal{P}} = 0$. Thus, there are four possibilities for the values of $(b_\nu^{\mathcal{P}_1}, b_\nu^{\mathcal{P}_2}, b_\nu^{\mathcal{P}_3})$: $(0,0,0)$, $(0,1,1)$, $(1,0,1)$, or $(1,1,0)$. We do a case-analysis just of the first two; the latter two are similar to the second:

Case B.1: $\{b_\nu^{\mathcal{P}}\} = (0,0,0)$:

	$\kappa^{\mathcal{P}_1}$ (via (16))	$\kappa^{\mathcal{P}_2}$ (via (16))	$\kappa^{\mathcal{P}_3}$ (via (16))
$x_\nu^{\mathcal{P}}$	$G_*(a) \oplus p_l$	$G_*(b) \oplus p_l$	$G_*(c) \oplus p_l$
$y_\nu^{\mathcal{P}}$	$G_*(d) \oplus p_l$	$G_*(d) \oplus p_l$	$G_*(d) \oplus p_l$
$z_\nu^{\mathcal{P}}$	$G_*(b) \oplus p_l$	$G_*(c) \oplus p_l$	$G_*(a) \oplus p_l$

Case B.2: $\{b_\nu^{\mathcal{P}}\} = (0,1,1)$:

	$\kappa^{\mathcal{P}_1}$ (via (15))	$\kappa^{\mathcal{P}_2}$ (via (16))	$\kappa^{\mathcal{P}_3}$ (via (15))
$x_\nu^{\mathcal{P}}$	$G_*(d) \oplus q_l$	$G_*(c) \oplus p_l$	$G_*(b) \oplus p_l$
$y_\nu^{\mathcal{P}}$	$G_*(a) \oplus q_l$	$G_*(a) \oplus q_l$	$G_*(a) \oplus q_l$
$z_\nu^{\mathcal{P}}$	$G_*(c) \oplus p_l$	$G_*(b) \oplus p_l$	$G_*(d) \oplus q_l$

Case C: μ is on-path, ν is off-path. Because parent node μ is on-path, its position (at depth $l-1$) corresponds to the DPF index $(\alpha)_{l-1}$ of MS-DPF$^+$ function f_l; and similarly ν off-path means that its position is does *not* correspond to $(\alpha)_l$, the DPF index of $\widehat{f}_l$. Therefore, $\widehat{w}_\nu^{\mathcal{P}_1} = \widehat{w}_\nu^{\mathcal{P}_2} = \widehat{w}_\nu^{\mathcal{P}_3}$ (by definition of $\widehat{f}_l$; see Step 2 of the Gen algorithm), and so we suppress player superscripts and write simply $\widehat{w}_\nu$. Meanwhile, per (7) we have that $w_*^{\mathcal{P}} = v_L^{\mathcal{P}} = G_L(x_\mu^{\mathcal{P}_L})$ if ν is the *left* child, and otherwise $w_*^{\mathcal{P}} = v_R^{\mathcal{P}} = G_R(x_\mu^{\mathcal{P}_L})$, since ν is off-path and parent μ is on-path, then $\alpha_l = 1$ (respectively $\alpha_l = 0$) when ν is the *left* child (respectively *right* child) of μ. Also, since μ is on-path, the seeds on μ satisfy invariant (2), and for convenience we will denote the three keys' seeds as above. Finally, since ν is off-path but parent node μ is on-path, the sibling of ν must be on-path, and thus by the invariant of (6), we have that $\bigoplus_{\mathcal{P}} b_\nu^{\mathcal{P}} = 1$. Thus, there are four possibilities for the values of $(b_\nu^{\mathcal{P}_1}, b_\nu^{\mathcal{P}_2}, b_\nu^{\mathcal{P}_3})$: $(1,1,1)$, $(0,0,1)$, $(0,1,0)$, or $(1,0,0)$. We do a case-analysis just of the first two; the latter two are similar to the second:

Case C.1: $\{b_\nu^{\mathcal{P}}\} = (1,1,1)$:

	$\kappa^{\mathcal{P}_1}$ (via (16))	$\kappa^{\mathcal{P}_2}$ (via (16))	$\kappa^{\mathcal{P}_3}$ (via (16))
$x_\nu^{\mathcal{P}}$	$G_*(b) \oplus \widehat{w}_\nu$	$G_*(c) \oplus \widehat{w}_\nu$	$G_*(a) \oplus \widehat{w}_\nu$
$y_\nu^{\mathcal{P}}$	$G_*(c) \oplus \widehat{w}_\nu$	$G_*(a) \oplus \widehat{w}_\nu$	$G_*(b) \oplus \widehat{w}_\nu$
$z_\nu^{\mathcal{P}}$	$G_*(a) \oplus \widehat{w}_\nu$	$G_*(b) \oplus \widehat{w}_\nu$	$G_*(c) \oplus \widehat{w}_\nu$

Case C.2: $\{b_\nu^{\mathcal{P}}\} = (0,0,1)$:

	$\kappa^{\mathcal{P}_1}$ (via (16))	$\kappa^{\mathcal{P}_2}$ (via (15))	$\kappa^{\mathcal{P}_3}$ (via (15))
$x_\nu^{\mathcal{P}}$	$G_*(a) \oplus \widehat{w}_\nu$	$G_*(d) \oplus \widehat{w}_\nu$	$G_*(b) \oplus \widehat{w}_\nu$
$y_\nu^{\mathcal{P}}$	$G_*(d) \oplus \widehat{w}_\nu$	$G_*(b) \oplus \widehat{w}_\nu$	$G_*(a) \oplus \widehat{w}_\nu$
$z_\nu^{\mathcal{P}}$	$G_*(b) \oplus \widehat{w}_\nu$	$G_*(a) \oplus \widehat{w}_\nu$	$G_*(d) \oplus \widehat{w}_\nu$

Case 2: ν is the right child of μ.

The argument for this case is essentially identical to Case 1, making the symmetric replacements of $H_L \to H_R$, $r_l \to s_l$, and $t_l \to u_l$. Details are provided in the full version. □