# Half-Tree: Halving the Cost of Tree Expansion in COT and DPF

Xiaojie Guo[1,2], Kang Yang[1(✉)], Xiao Wang[3], Wenhao Zhang[3], Xiang Xie[4,5], Jiang Zhang[1], and Zheli Liu[2(✉)]

[1] State Key Laboratory of Cryptology, Beijing, China
yangk@sklc.org
[2] Nankai University, Tianjin, China
xiaojie.guo@mail.nankai.edu.cn, liuzheli@nankai.edu.cn, jiangzhang09@gmail.com
[3] Northwestern University, Evanston, USA
wangxiao@cs.northwestern.edu, wenhao.zhang@northwestern.edu
[4] Shanghai Qi Zhi Institute, Shanghai, China
xiexiangiscas@gmail.com
[5] PADO Labs, Hong Kong, China

**Abstract.** GGM tree is widely used in the design of correlated oblivious transfer (COT), subfield vector oblivious linear evaluation (sVOLE), distributed point function (DPF), and distributed comparison function (DCF). Often, the cost associated with GGM tree dominates the computation and communication of these protocols. In this paper, we propose a suite of optimizations that can reduce this cost by half.

- **Halving the cost of COT and sVOLE.** Our COT protocol introduces extra correlation to each level of a GGM tree used by the state-of-the-art COT protocol. As a result, it reduces both the number of AES calls and the communication by half. Extending this idea to sVOLE, we are able to achieve similar improvement with either halved computation or halved communication.
- **Halving the cost of DPF and DCF.** We propose improved two-party protocols for the distributed generation of DPF/DCF keys. Our tree structures behind these protocols lead to more efficient full-domain evaluation and halve the communication and the round complexity of the state-of-the-art DPF/DCF protocols.

All protocols are provably secure in the random-permutation model and can be accelerated based on fixed-key AES-NI. We also improve the state-of-the-art schemes of puncturable pseudorandom function (PPRF), DPF, and DCF, which are of independent interest in dealer-available scenarios.

## 1 Introduction

The construction of Goldreich-Goldwasser-Micali (GGM) tree [26] yields a pseudorandom function (PRF) family from any length-doubling pseudorandom generator (PRG). In this construction, a PRF key serves as a root and is expanded

The original version of this chapter was revised: this paper showed a wrong target acknowledgement. This now has been corrected. The correction to this chapter is available at https://doi.org/10.1007/978-3-031-30545-0_23


C. Hazay and M. Stam (Eds.): EUROCRYPT 2023, LNCS 14004, pp. 330–362, 2023.
https://doi.org/10.1007/978-3-031-30545-0_12

**Table 1. Improvements of our protocols in the random-permutation model.** Computation is measured as the number of fixed-key AES calls. In sVOLE, communication varies as per two field sizes $|\mathbb{F}|$ and $|\mathbb{K}|$. In DCF protocol, communication varies as per the range size $|\mathcal{R}|$ of comparison functions.

| **Protocol** | Computation | Communication | # Rounds |
|---|---|---|---|
| COT (§4.1) | 2× | 2× | – |
| sVOLE (§4.1) | 2× | $1 \sim 2\times$ | – |
| sVOLE (§4.2) | 1.33× | 2× | – |
| DPF (§5.2) | 1.33× | 3× | 2× |
| DCF (§5.3) | 1.6× | $2 \sim 3\times$ | 2× |

into a full binary tree, where each non-leaf node defines two child nodes from its PRG output. The PRF output for an input bit-string is defined as the leaf node labeled by this bit-string. GGM tree has been adapted widely for various cryptographic applications, especially in recent years.

A recent appealing application of GGM tree is to build efficient pseudorandom correlation generators (PCGs) [8,10,12,42,43,46], e.g., correlated oblivious transfer (COT), subfield vector oblivious linear evaluation (sVOLE), etc. In this context, a GGM tree essentially serves as a puncturable pseudorandom function (PPRF). PCGs serve as essential building blocks for secure multi-party computation (MPC) (e.g., [27,33]), zero-knowledge proofs (e.g., [2,21,43]), private set intersection (e.g., [23,40]), etc. Another related application of GGM tree is to build function secret sharing (FSS). In an FSS scheme, a dealer produces two keys, each defining an additive secret sharing of the full-domain evaluation result of some function $f$ without revealing the parameters of $f$. FSS is very useful even for a simple $f$, and the dealer can be emulated using an MPC protocol. A distributed point function (DPF) [25] is an FSS scheme for the family of point functions $f^{\bullet}_{\alpha,\beta}(x)$ that output $\beta$ if $x = \alpha$ and 0 otherwise. DPF has found various applications, including RAM-based secure computation [22], two-server PIR [13,25], private heavy hitters [6], oblivious linear evaluation (OLE) [12], etc. One important variation of DPF is distributed comparison function (DCF), which is an FSS scheme for the family of comparison functions $f^{<}_{\alpha,\beta}(x)$ that output $\beta$ if $x < \alpha$ and 0 otherwise. DCF has been applied to design mixed-mode MPC [7,14], secure machine-learning inference [30], etc.

In all applications above, the cost associated with GGM tree can often be significant. For example, in the most recent silent OT protocol [18], distributing GGM-tree-related correlations takes more than 70% of the computation and essentially all communication. Similar bottlenecks have also been observed in DPF. For example, in the DPF-based secure RAM computation [22], local expansion of DPF keys takes a majority of the time as well.

### 1.1 Our Contribution

We propose a suite of *half-trees* as tailored alternatives for several GGM-tree-based protocols, leading to halved computation/communication/round complexity (Table 1, detailed complexity is compared in the sections). Our constructions work in the random-permutation model (RPM) [4,41], which can be efficiently instantiated via, e.g., fixed-key AES-NI.

**Correlated GGM Trees for Half-Cost COT and sVOLE.** We introduce correlated GGM (cGGM), a tree structure leading to both improved computation and communication in COT. It has an invariant that all same-level nodes sum up to the same global offset. We keep this invariant by setting a left child as the hash of its parent and the associated right child as the parent minus the left child. By plugging this tree into the state-of-the-art COT protocols [18,46], we can prove the security of the whole protocol in the random-permutation model by carefully choosing the hash function. Compared to the optimized GGM tree [28], this tree reduces the number of random-permutation calls and the communication by half.

Using cGGM tree, we can realize sVOLE for any large field and its subfield. This protocol reduces the computation of the prior protocols [10,43] by 2× using a field-based random permutation. However, it only halves the communication when the subfield size is significantly smaller than the field size. Then, we modify our cGGM tree to obtain a pseudorandom correlated GGM (pcGGM) tree, which is similar to a cGGM tree but has pseudorandom leaves. In contrast, pcGGM tree leads to a 2× saving in communication and a 1.33× saving in computation.

**Halved communication and round complexity in distributed key generation of DPF and DCF.** We introduce another binary tree structure, which adapts our pcGGM tree into a secretly shared form. This tree leads to a new DPF scheme with an improved distributed key generation protocol. This DPF protocol reduces the computation, communication, and round complexity of the prior work roughly by 1.33×, 3×, and 2×, respectively. When the range of point functions is a general ring, this shared tree allows simpler secure computation than the prior works in terms of the last correction word.

We also use an extended version of this shared pcGGM tree to design a new DCF scheme also with an improved distributed key generation protocol. The tree expansion in our DCF is much simpler than the prior work [7], where each parent node has to quadruple in length to produce additional correction words. In our extended shared pcGGM tree, this expansion factor in length is two or three, and the resulting additional correction words are more 2PC-friendly. When used in our DCF protocol with typical parameters, this extended tree leads to about 1.6×, 2 ∼ 3×, and 2× savings in terms of computation, communication, and round complexity in contrast to the prior work.

### 1.2 Concurrent Work

Recently, Boyle et al. [9] propose two unpredictable punctured functions (UPFs) that can be converted to PPRF with additional $0.5N$ RO calls for $N$-sized domain. Their first UPF construction needs $N$ RO calls and is provably secure while the second UPF construction needs $N$ RP calls but relies on an ad-hoc conjecture. For $m$-sized sVOLE tuples, the sVOLE extension protocols based on their proposal either needs $1.5m$ RO calls, or needs $m$ RP calls plus $0.5m$ RO calls. They also propose an sVOLE extension protocol that is based on a stronger variation of UPF and requires $m$ RO calls in total.

In contrast, our protocol is secure in the random-permutation model without any conjecture. Our COT protocol, as a special case of sVOLE protocol, only

**Table 2. Comparison with the concurrent work.** "RO/ROM" (resp., "RP/RPM") is for random oracle (resp., permutation) and the model. $P_0$ is the sender with a global key, and $P_1$ is the receiver. Assume weight-$t$ regular LPN noises in sVOLE extension with output length $m$, field $\mathbb{F}$, and extension field $\mathbb{K}$. Computation is measured by the amount of symmetric-key operations, and there is also LPN-related computation in practice. Communication is measured by assuming $P_0$ and $P_1$ have access to random precomputed tuples: (i) [9]: $t \log \frac{m}{t}$ COTs (+ $t$ sVOLEs, for general sVOLE extension), (ii) our COT extension: $t \log \frac{m}{t}$ COTs, (iii) our first sVOLE extension: $t(\log \frac{m}{t} + 1)$ sVOLEs, and (iv) our second sVOLE extension: $t \log \frac{m}{t}$ COTs + $t$ sVOLEs.

| | Assump. | Corr. | Computation | Communication (bits) | |
|---|---|---|---|---|---|
| | | | | $P_0 \to P_1$ | $P_1 \to P_0$ |
| [9] | ROM | sVOLE | $m$ RO calls | $2t(\log \frac{m}{t} - 1)\lambda$ $+3t \log \lvert\mathbb{K}\rvert$ | $t \log \lvert\mathbb{F}\rvert$ |
| | Ad-hoc[a] | sVOLE | $m$ RP calls+ $0.5m$ RO calls | | |
| This work | RPM | COT | $m$ RP calls | $t(\log \frac{m}{t} - 1)\lambda + \lambda$ | – |
| | | sVOLE | $m$ RP calls | $t(\log \frac{m}{t} - 1) \log \lvert\mathbb{K}\rvert$ $+\lambda$ | $t(\log \frac{m}{t} + 1) \log \lvert\mathbb{F}\rvert$ |
| | | sVOLE | $1.5m$ RP calls | $t(\log \frac{m}{t} - 2)\lambda$ $+3t \log \lvert\mathbb{K}\rvert + \lambda$ | $t \log \lvert\mathbb{F}\rvert$ |

[a] Security relies on the conjecture that the punctured result of the RPM-based UPF is unpredictable. This UPF uses GGM-style tree expansion $G(x) := \mathsf{H}_0(x) \,\|\, \mathsf{H}_1(x)$ for $\mathsf{H}_0(x) := \mathsf{H}(x) \oplus x$ and $\mathsf{H}_1(x) := \mathsf{H}(x) + x \bmod 2^\lambda$.

requires $m$ RP calls and can reduce communication by half; our two sVOLE protocols need $m$ or $1.5m$ RP calls with different levels of communication reduction. More importantly, we also demonstrate how the idea can be applied to DPF/DCF protocols as well.

In Table 2, we compare the cost of sVOLE extension in the two works. The sVOLE extension in both works can be easily turned into the extension of random OTs via the standard transformation [3,10,34]. If we regard one (length-preserving) RO call as two RP calls according to the XOR-based construction of [5], our work also beats the concurrent one in terms of concrete efficiency.

## 2 Preliminaries

### 2.1 Notation

Let $\lambda$ denote the computational security parameter. $n = n(\lambda)$ means that $n \in \mathbb{N}$ is polynomial in $\lambda$. Let $\mathsf{negl}(\cdot)$ denote an unspecified negligible function and $\log(\cdot)$ denote the logarithm in base 2. Let $x \leftarrow S$ denote sampling $x$ uniformly at random from a finite set $S$. Let $[a, b) := \{a, \ldots, b-1\}$ and $[a, b] := \{a, \ldots, b\}$. Let $\mathbb{G}$ (resp., $\mathcal{R}$) denote finite group (resp., ring). We use bold lowercase letters (e.g., $\mathbf{a}$) for vectors. For $i \geq 0$, let $\mathbf{a}^{(i)}$ denote the $i$-th entry of vector $\mathbf{a}$. Let $\mathbf{unit}_{\mathbb{G}}(n, \alpha, \beta) \in \mathbb{G}^n$ denote the vector whose $\alpha$-th entry is $\beta$ and others are 0. For some field $\mathbb{F}$ and irreducible polynomial $f(X) \in \mathbb{F}[X]$, let $\mathbb{K} = \mathbb{F}[X]/f(X)$ denote an extension field. For some $n \in \mathbb{N}$, we interchangeably use $\mathbb{F}_{2^n}$, $\mathbb{F}_2^n$, and $\{0,1\}^n$, where $\oplus$ is for bitwise-XOR. For some bit-string $x \in \{0,1\}^n$, let $\mathsf{lsb}(x)$ denote its least significant bit (LSB), $\mathsf{hb}(x)$ denote its high $n-1$ bits, and $x_i$

denote its $i$-th bit such that $x_1$ is the most significant one. We use $\|$ for bit-string concatenation and $\circ$ for function composition. Let $\mathsf{Convert}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$ denote a function that maps random strings to pseudorandom $\mathbb{G}$ elements (see Appendix F.1 of the full version [29] for its implementation).

**Binary Trees.** In an $n$-level tree, let $X_i^j$ denote the $j$-th node on its $i$-level for $i \in [1,n]$ and $j \in [0,2^i)$. We can write the superscript $j$ into $i$-bit decomposition, i.e., $X_i^{j_1 \dots j_i} := X_i^j$. When a node $X_i^j \in \{0,1\}^n$, we can decompose it into a seed $s_i^j := \mathsf{hb}(X_i^j) \in \{0,1\}^{n-1}$ and a control bit $t_i^j := \mathsf{lsb}(X_i^j) \in \{0,1\}$ such that $X_i^j = (s_i^j \,\|\, t_i^j)$. We usually omit the superscript $j$ if it is the $i$-bit prefix of a path $\alpha \in \{0,1\}^n$ of particular interest in a given context. For completeness, let $X_0$ denote the root. For some $i \in [1,n]$ and $b \in \{0,1\}$, let $K_i^b$ denote the sum of the $2^{i-1}$ $b$-side (i.e., left or right) nodes on the $i$-th level.

**Secret Sharings.** For some additive Abelian group $\mathbb{G}$ and $x \in \mathbb{G}$, we use $\langle x \rangle^{\mathsf{A}}$ to mean that $x$ is additively shared between two parties and call it a secret for short. For some secret $\langle x \rangle^{\mathsf{A}}$ for $x \in \mathbb{G}$ and party $b \in \{0,1\}$, let $\langle x \rangle_b^{\mathsf{A}} \in \mathbb{G}$ denote the secret share of the party $b$ such that $x = \langle x \rangle_0^{\mathsf{A}} + \langle x \rangle_1^{\mathsf{A}}$. We abbreviate $\langle x \rangle^{\mathsf{A}}$ to $\langle x \rangle$ and $\langle x \rangle_b^{\mathsf{A}}$ to $\langle x \rangle_b$ if $\mathbb{G} = \{0,1\}^n$. For some secret $\langle x \rangle$ for $x \in \{0,1\}^n$ and efficiently computable (possibly non-linear) Boolean circuit $\mathsf{H} : \{0,1\}^n \to \{0,1\}^*$, let $\mathsf{H}(\langle x \rangle)$ denote such a *linear evaluation* that returns a secret $\langle y \rangle$ with share $\langle y \rangle_b := \mathsf{H}(\langle x \rangle_b)$ for each $b \in \{0,1\}$.

### 2.2 Security Model and Functionalities

We use the *universal composability* (UC) framework [15] to prove security in the presence of a semi-honest, static adversary. We say that a protocol $\Pi$ *UC-realizes* an ideal functionality $\mathcal{F}$ if for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exists a PPT adversary (simulator) $\mathcal{S}$ such that for any PPT environment $\mathcal{Z}$ with arbitrary auxiliary input $z$, the output distribution of $\mathcal{Z}$ in the *real-world* execution where the parties interact with $\mathcal{A}$ and execute $\Pi$ is computationally indistinguishable from the output distribution of $\mathcal{Z}$ in the *ideal-world* execution where the parties interact with $\mathcal{S}$ and $\mathcal{F}$.

Our protocols use the functionality $\mathcal{F}_{\mathsf{sVOLE}}$ (Fig. 1) of subfield vector oblivious linear evaluation. If $\mathbb{K} = \mathbb{F}_{2^\lambda}$ and $\mathbb{F} = \mathbb{F}_2$, $\mathcal{F}_{\mathsf{sVOLE}}$ degenerates to the COT functionality $\mathcal{F}_{\mathsf{COT}}$ in [46]. If $\mathbb{K} = \mathbb{F}$, $\mathcal{F}_{\mathsf{sVOLE}}$ serves as the VOLE functionality in [8,40,42]. We omit the session IDs and sub-session IDs in the functionalities for simplicity. By convention, we can write sVOLE tuples as two-party *information-theoretic message authentication codes* (IT-MACs) [20,38]. Let $\Delta_b \in \mathbb{K}$ denote the global key of one party $P_b$. $P_b$ authenticates a value $x \in \mathbb{F}$ of the other party $P_{1-b}$ by sampling a uniform one-time key $\mathsf{K}_b[x] \leftarrow \mathbb{K}$ and giving to $P_{1-b}$ the MAC $\mathsf{M}_{1-b}[x] := \mathsf{K}_b[x] + x \cdot \Delta_b \in \mathbb{K}$. If identity $b \in \{0,1\}$ is clear in a given context, we write $\Delta$, $\mathsf{K}[x]$, and $\mathsf{M}[x]$ for $\Delta_b$, $\mathsf{K}_b[x]$, and $\mathsf{M}_{1-b}[x]$, respectively.

### 2.3 Circular Correlation Robustness

Circular correlation robustness (CCR) [17,28] is the security notion first introduced for the circuit garbling with Free-XOR optimization [37], where there

**Functionality $\mathcal{F}_{\mathsf{sVOLE}}$**

**Parameters:** Field $\mathbb{F}$ and its extension field $\mathbb{K}$.

**Initialize:** Upon receiving (init) from $P_0$ and $P_1$, sample $\Delta \leftarrow \mathbb{K}$ if $P_0$ is honest; otherwise, receive $\Delta \in \mathbb{K}$ from the adversary. Store $\Delta$ and send it to $P_0$. Ignore all subsequent (init) commands.

**Extend:** This functionality allows polynomially many (extend) commands. Upon receiving $(\mathsf{extend}, m)$ from $P_0$ and $P_1$:

1. If $P_0$ is honest, sample $\mathbf{v} \leftarrow \mathbb{K}^m$; otherwise, receive $\mathbf{v} \in \mathbb{K}^m$ from the adversary.
2. If $P_1$ is honest, sample $\mathbf{u} \leftarrow \mathbb{F}^m$, and compute $\mathbf{w} := \mathbf{v} + \mathbf{u} \cdot \Delta \in \mathbb{K}^m$; otherwise, receive $(\mathbf{u}, \mathbf{w}) \in \mathbb{F}^m \times \mathbb{K}^m$ from the adversary, and recompute $\mathbf{v} := \mathbf{w} - \mathbf{u} \cdot \Delta \in \mathbb{K}^m$.
3. Send $\mathbf{v}$ to $P_0$ and $(\mathbf{u}, \mathbf{w})$ to $P_1$.

**Global-key queries:** If $P_1$ is corrupted, upon receiving $(\mathsf{guess}, \Delta')$, where $\Delta' \in \mathbb{K}$, from the adversary, send (success) to the adversary if $\Delta = \Delta'$; send (fail) to the adversary otherwise.

**Fig. 1.** Functionality for subfield VOLE.

exists a global key $\Delta$ offsetting the inputs and outputs of some function $\mathsf{H}$. [28] showed that a CCR function $\mathsf{H}$ can be constructed from a fixed-key block cipher (e.g., AES) modeled as random permutation and a *linear orthomorphism*[1]. In this construction, it takes one block-cipher call to invoke a CCR function.

**Definition 1 (Circular Correlation Robustness, [28]).** *Let* $\mathsf{H} : \{0,1\}^\lambda \to \{0,1\}^\lambda$*,* $\chi$ *be a distribution on* $\{0,1\}^\lambda$*, and* $\mathcal{O}^{\mathsf{ccr}}_{\mathsf{H},\Delta}(x,b) := \mathsf{H}(x \oplus \Delta) \oplus b \cdot \Delta$ *be an oracle for* $x, \Delta \in \{0,1\}^\lambda$ *and* $b \in \{0,1\}$*.* $\mathsf{H}$ *is* $(t, q, \rho, \epsilon)$*-CCR if, for any distinguisher* $\mathcal{D}$ *running in time at most* $t$ *and making at most* $q$ *queries to* $\mathcal{O}^{\mathsf{ccr}}_{\mathsf{H},\Delta}(\cdot,\cdot)$*, and any* $\chi$ *with min-entropy at least* $\rho$*, it holds that*

$$\left| \Pr_{\Delta \leftarrow \chi} \left[ \mathcal{D}^{\mathcal{O}^{\mathsf{ccr}}_{\mathsf{H},\Delta}(\cdot,\cdot)}(1^\lambda) = 1 \right] - \Pr_{f \leftarrow \mathcal{F}_{\lambda+1,\lambda}} \left[ \mathcal{D}^{f(\cdot,\cdot)}(1^\lambda) = 1 \right] \right| \leq \epsilon,$$

*where* $\mathcal{D}$ *cannot query both* $(x,0)$ *and* $(x,1)$ *for any* $x \in \{0,1\}^\lambda$*.*

In this work, $\mathcal{D}$ can only make CCR queries with *restricted* forms, which are reminiscent of those in the Half-Gate garbling scheme [47]. We defer the formal definition of these restricted queries to Appendix A of the full version [29].

[1] A mapping $\sigma : \mathbb{G} \to \mathbb{G}$ for an additive Abelian group $\mathbb{G}$ is a linear orthomorphism if (i) $\sigma$ is a permutation, (ii) $\sigma(x+y) = \sigma(x) + \sigma(y)$ for any $x, y \in \mathbb{G}$, and (iii) $\sigma'(x) := \sigma(x) - x$ is also a permutation. [28] presents two efficient instantiations of $\sigma$ (with well-defined efficient $\sigma^{-1}$, $\sigma'$, and $\sigma'^{-1}$): (i) if $\mathbb{G}$ is a field, $\sigma(x) := c \cdot x$ for some $c \neq 0, 1 \in \mathbb{G}$, and (ii) if $\mathbb{G} = \{0,1\}^n$, $\sigma(x) = \sigma(x_L \,\|\, x_R) := (x_L \oplus x_R) \,\|\, x_L$ where $x_L$ and $x_R$ are the left and right halves of $x$. .

### 2.4 Function Secret Sharing

A function secret sharing (FSS) is a secret sharing scheme where a dealer distributes the shares of a function $f$ to multiple parties, and each party can use its share to *locally* compute the share of $f(x)$ for any *public* $x$ in the domain of $f$. In this work, we focus on two-party FSS schemes.

**Definition 2 (Function Secret Sharing, [7,13]).** *For a family $\mathcal{F}_{\mathcal{X},\mathbb{G}}$ of functions with domain $\mathcal{X}$ and range $\mathbb{G}$, where $\mathbb{G}$ is an Abelian group, a two-party FSS scheme with key space $\mathcal{K}_0 \times \mathcal{K}_1$ has the following syntax:*

- $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, \hat{f})$. *On input $1^\lambda$ and the description $\hat{f} \in \{0,1\}^*$ of a function $f \in \mathcal{F}_{\mathcal{X},\mathbb{G}}$, output a key pair $(k_0, k_1) \in \mathcal{K}_0 \times \mathcal{K}_1$.*
- $f_b(x) \leftarrow \mathsf{Eval}(b, k_b, x)$. *On input the party identifier $b \in \{0,1\}$, the party's key $k_b \in \mathcal{K}_b$, and a point $x \in \mathcal{X}$, output the share $f_b(x) \in \mathbb{G}$.*

*A two-party FSS scheme* $(\mathsf{Gen}, \mathsf{Eval})$ *is secure for the function family $\mathcal{F}_{\mathcal{X},\mathbb{G}}$ with leakage $\mathsf{Leak} : \{0,1\}^* \to \{0,1\}^*$ if the following properties hold.*

- **Correctness**. *For any function $f \in \mathcal{F}_{\mathcal{X},\mathbb{G}}$ with description $\hat{f}$, and any $x \in \mathcal{X}$,*

$$\Pr\left[(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, \hat{f}) : \textstyle\sum_{b\in\{0,1\}} \mathsf{Eval}(b, k_b, x) = f(x)\right] = 1.$$

- **Security**. *There exists a PPT simulator* $\mathsf{Sim}$ *such that, for any function $f \in \mathcal{F}_{\mathcal{X},\mathbb{G}}$ with the description $\hat{f}$, any $b \in \{0,1\}$, and any PPT adversary $\mathcal{A}$,*

$$\left|\Pr\left[(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, \hat{f}) : \mathcal{A}(1^\lambda, k_b) = 1\right]\right.$$
$$\left. - \Pr\left[k_b \leftarrow \mathsf{Sim}(1^\lambda, b, \mathsf{Leak}(\hat{f})) : \mathcal{A}(1^\lambda, k_b) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

By default, the leakage $\mathsf{Leak}(\hat{f})$ only involves the domain and the range of $f$. The following two special FSS schemes have been proposed in [7,13].

**Distributed Point Functions (DPFs).** A two-party *distributed point function* $(\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ with domain $\mathcal{X}$ and range $\mathbb{G}$ is a two-party FSS scheme for the function family $\mathcal{F}_{\mathcal{X},\mathbb{G}} = \{f^{\bullet}_{\alpha,\beta}\}_{\alpha\in\mathcal{X},\beta\in\mathbb{G}}$ where $f^{\bullet}_{\alpha,\beta}$ is a *point function* such that $f^{\bullet}_{\alpha,\beta}(\alpha) = \beta$, and $f^{\bullet}_{\alpha,\beta}(x) = 0$ for $x \neq \alpha \in \mathcal{X}$.

**Distributed Comparison Functions (DCFs).** A two-party *distributed comparison function* $(\mathsf{DCF.Gen}, \mathsf{DCF.Eval})$ with domain $\mathcal{X}$ and range $\mathbb{G}$ is a two-party FSS scheme for the function family $\mathcal{F}_{\mathcal{X},\mathbb{G}} = \{f^{<}_{\alpha,\beta}\}_{\alpha\in\mathcal{X},\beta\in\mathbb{G}}$ where $f^{<}_{\alpha,\beta}$ is a *comparison function* such that $f^{<}_{\alpha,\beta}(x) = \beta$ if $x < \alpha \in \mathcal{X}$, and $f^{<}_{\alpha,\beta}(x) = 0$ otherwise.

## 3 Technical Overview

### 3.1 Improved COT/sVOLE from Correlated GGM Trees

Since COT/sVOLE can be constructed from its "single-point" version using an appropriate LPN assumption, we focus on single-point COT/sVOLE, where the vector $\mathbf{u}$ in a COT/sVOLE tuple $\mathbf{w} = \mathbf{v} + \mathbf{u} \cdot \Delta$ has exactly one non-zero entry.

**Correlated OT from Correlated GGM.** The core idea behind our single-point COT protocol is that, instead of using a GGM tree with pseudorandom nodes as the state-of-the-art works, our protocol uses a *correlated* GGM (cGGM) tree where *the sum of all same-level nodes equals a global offset* $\Delta$. This invariant can be maintained by using a generalized Davies-Meyer construction with a hash function $\mathsf{H}$: every parent $x$ has left child $\mathsf{H}(x)$ and right child $x - \mathsf{H}(x)$. cGGM tree leads to two improvements: (i) no additional hash computation is needed for every right child so that the computation is halved, and (ii) if the global offset $\Delta$ (i.e., the difference between two first-level nodes) is set up by precomputed random COT tuples, the single-point COT protocol sends only $\lambda$ bits per level, in contrast to $2\lambda$ bits from a standard OT per level in the state-of-the-art works.

To explain our second improvement in detail, we first recall the prior construction from the perspective of GGM tree. In this construction, the sender holds an $n$-level GGM tree, whose $2^n$ leaves in $\mathbb{F}_{2^\lambda}$ forms a vector $\mathbf{v} \in \mathbb{F}_{2^\lambda}^{2^n}$. The receiver with a punctured point $\alpha = \alpha_1 \dots \alpha_n \in \{0,1\}^n$ uses, for each $i \in [1,n]$, a standard OT to select the XOR of all $\overline{\alpha}_i$-side nodes on the $i$-th level. From these $n$ XORs, the receiver recovers the $n$ off-path GGM-tree nodes just leaving the path $\alpha$ and use these $n$ nodes to recover all leaves except the $\alpha$-th one, corresponding to a vector $\mathbf{w} \in \mathbb{F}_{2^\lambda}^{2^n}$ with the punctured entry $\mathbf{w}^{(\alpha)}$. The sender samples $\Delta \leftarrow \mathbb{F}_{2^\lambda}$, defines its output as $(\Delta, \mathbf{v})$, and sends $\psi := \Delta \oplus (\oplus_{j \in [0,2^n)} \mathbf{v}^{(j)}) \in \mathbb{F}_{2^\lambda}$ to the receiver. The receiver patches $\mathbf{w}^{(\alpha)} := \psi \oplus (\oplus_{j \neq \alpha} \mathbf{w}^{(j)})$ and defines its output as $(\mathbf{u}, \mathbf{w})$ for $\mathbf{u} = \mathbf{unit}_{\mathbb{F}_2}(2^n, \alpha, 1)$. The computation is dominated by the full GGM-tree expansion while the communication is from $n$ parallel standard OTs, which need $n$ precomputed COT tuples via the standard technique [3,34].

In contrast, our cGGM-tree single-point COT, where the global offset in a cGGM tree coincides with the global key in the $n$ precomputed COT tuples, can directly use these tuples. For each level $i \in [1,n]$, let $\mathsf{M}[r_i] = \mathsf{K}[r_i] \oplus r_i \cdot \Delta$ be such a tuple where the sender has $(\Delta, \mathsf{K}[r_i]) \in \mathbb{F}_{2^\lambda} \times \mathbb{F}_{2^\lambda}$ and the receiver has $(r_i, \mathsf{M}[r_i]) \in \mathbb{F}_2 \times \mathbb{F}_{2^\lambda}$, and $K_i^b \in \mathbb{F}_{2^\lambda}$ be the XOR of all $b$-side nodes for $b \in \{0,1\}$. To select $K_i^{\overline{\alpha}_i}$ as in the prior construction, the receiver sends $\overline{\alpha}_i \oplus r_i$ to the sender, receives back $c_i := K_i^0 \oplus \mathsf{K}[r_i] \oplus (\overline{\alpha}_i \oplus r_i) \cdot \Delta$, and computes

$$c_i \oplus \mathsf{M}[r_i] = K_i^0 \oplus \mathsf{K}[r_i] \oplus (\overline{\alpha}_i \oplus r_i) \cdot \Delta \oplus \mathsf{M}[r_i] = K_i^0 \oplus \overline{\alpha}_i \cdot \Delta = K_i^{\overline{\alpha}_i},$$

where the last equality holds since the cGGM tree uses $\Delta$ as global offset. For each level, the sender sends $\lambda$ bits to the receiver, only a half of the $2\lambda$ bits in a standard OT. When the point $\alpha$ is random, the message $\overline{\alpha}_i \oplus r_i$ can be avoided as well. The single-point COT outputs are defined as in the prior construction, except that the receiver locally patches $\mathbf{w}^{(\alpha)} := \oplus_{j \neq \alpha} \mathbf{w}^{(j)}$.

The security against the semi-honest sender is straightforward. However, a subtle issue arises in proving the security against the semi-honest receiver. Note that the environment $\mathcal{Z}$ can observe the global key $\Delta$ from the honest sender's output and use it to distinguish the two worlds. Let $\{X_i^{\alpha_1 \dots \alpha_{i-1}\overline{\alpha}_i}\}_{i \in [1,n]}$ be the cGGM-tree off-path nodes recovered by the receiver. In the real world, these off-path nodes satisfy the consistency with $\Delta$: for $j \in [2,n]$, $X_j^{\alpha_1 \dots \alpha_{j-1}\overline{\alpha}_j}$ equals

$$\mathsf{H}\left(\Delta \oplus \bigoplus_{i\in[1,j-1]} X_i^{\alpha_1\ldots\alpha_{i-1}\overline{\alpha}_i}\right) \oplus \overline{\alpha}_j \cdot \left(\Delta \oplus \bigoplus_{i\in[1,j-1]} X_i^{\alpha_1\ldots\alpha_{i-1}\overline{\alpha}_i}\right). \quad (1)$$

However, this consistency does not hold in the ideal world where $\{c_i\}_{i\in[1,n]}$ sent by the simulator are sampled at random so that the $n$ off-path nodes will be independently uniform in the ideal world. Thus, $\mathcal{Z}$ can trivially distinguish the two worlds by using the known $\Delta$ to check (1). Our security proof addresses this issue by carefully constructing $\mathsf{H}$ from a random permutation, allowing global-key queries in the single-point COT functionality, and programming the random permutation and its inverse to keep the consistency. The intuition is that, to distinguish the two worlds, $\mathcal{Z}$ must query the random permutation or its inverse with $\Delta$-related transcripts. Thus, the simulator can observe these queries and extract every potential $\Delta$ from them. Using global-key queries, the simulator checks whether an extracted $\Delta$ matches that in the single-point COT functionality or not. If so, it immediately programs the two permutation oracles using this $\Delta$ so that they are consistent with the simulated $\{c_i\}_{i\in[1,n]}$. Similar proof technique in the random-oracle model have been used in TinyOT [32,38].

**Subfield VOLE from Correlated GGM.** We further propose a cGGM-based blueprint of single-point sVOLE for field $\mathbb{F}$ and its exponentially large extension $\mathbb{K}$. In this blueprint, we construct an $n$-level cGGM tree from a hash function $\mathsf{H} : \mathbb{K} \to \mathbb{K}$ so that all nodes are in $\mathbb{K}$, and extend the spirit of our single-point COT. The spirit is that the equality $\mathbf{w}^{(\alpha)} = \mathbf{v}^{(\alpha)} \oplus \Delta$ at the punctured point $\alpha$ automatically holds by embedding $\Delta$ into a cGGM tree. For single-point sVOLE, we want to likewise keep $\mathbf{w}^{(\alpha)} = \mathbf{v}^{(\alpha)} + \beta \cdot \Delta$ for some $\beta \in \mathbb{F}^*$ and $\Delta \in \mathbb{K}$ at the punctured point $\alpha$. However, we cannot use $\beta \cdot \Delta$, which is unknown to the sender, as the cGGM-tree global offset. Instead, we can define this offset as the sender's additive share of $\beta \cdot \Delta$ so that the receiver can correct the automatically preserved result at the point $\alpha$ by using its additive share of $\beta \cdot \Delta$.

In detail, the two parties use a random sVOLE tuple $\mathsf{M}[\beta] = \mathsf{K}[\beta] + \beta \cdot \Delta$ for the $\beta \cdot \Delta$ term, where the sender has $(\Delta, \mathsf{K}[\beta]) \in \mathbb{K} \times \mathbb{K}$ and the receiver has $(\beta, \mathsf{M}[\beta]) \in \mathbb{F}^* \times \mathbb{K}$. The sender uses $\mathsf{K}[\beta]$ as the global offset of its cGGM tree, and the receiver selects, for each level $i$, the sum of all $\overline{\alpha}_i$-side nodes. For the $i$-th level, let $K_i^b \in \mathbb{K}$ be the sum of all $b$-side nodes for $b \in \{0, 1\}$, and let the two parties have access to a special sVOLE tuple[2] $\mathsf{M}[r_i] = \mathsf{K}[r_i] + r_i \cdot \mathsf{K}[\beta]$, where the sender has $\mathsf{K}[r_i] \in \mathbb{K}$ and the receiver has $(r_i, \mathsf{M}[r_i]) \in \mathbb{F}_2 \times \mathbb{K}$. The sender sends $c_i := \mathsf{K}[r_i] + K_i^0 \in \mathbb{K}$ to the receiver, who defines $\overline{\alpha}_i := r_i$ and can compute

$$(-1)^{r_i} \cdot (-\mathsf{M}[r_i] + c_i) = (-1)^{\overline{\alpha}_i} \cdot (K_i^0 - \overline{\alpha}_i \cdot \mathsf{K}[\beta]) = K_i^{\overline{\alpha}_i},$$

where the last equality holds due to the cGGM invariant. The $n$ selected sums allow the receiver to recover, in a top-down manner, the $n$ off-nodes with respect to $\alpha$ and the $2^n$ cGGM leaves except the $\alpha$-th one. The sender defines $\mathbf{v} \in \mathbb{K}^{2^n}$ from its $2^n$ cGGM-tree leaves, while the receiver defines $\mathbf{w} \in \mathbb{K}^{2^n}$ from the $\alpha$-exclusive $2^n - 1$ leaves and the locally patched punctured leaf $\mathbf{w}^{(\alpha)} := \mathsf{M}[\beta] -$

[2] The special sVOLE tuples for selecting $n$ sums can be obtained from $n$ precomputed random sVOLE tuples by the receiver sending $n \cdot \log|\mathbb{F}|$ bits.

$\sum_{j\neq\alpha}\mathbf{w}^{(j)} = \mathsf{M}[\beta] - (\sum_{j\neq\alpha}\mathbf{w}^{(j)} + \mathbf{v}^{(\alpha)}) + \mathbf{v}^{(\alpha)} = \mathbf{v}^{(\alpha)} + \beta\cdot\Delta$. If the sender defines its output as $(\Delta, \mathbf{v})$ and the receiver defines its output as $(\mathbf{u}, \mathbf{w})$ for $\mathbf{u} := \mathbf{unit}_{\mathbb{F}}(2^n, \alpha, \beta)$, the two parties share a single-point sVOLE correlation.

Our cGGM-based single-point sVOLE protocol also has the issue in proving the security against the semi-honest receiver as the environment $\mathcal{Z}$ sees $\Delta$ from the honest sender's output. $\mathcal{Z}$ can compute the cGGM offset $\mathsf{K}[\beta] = \mathsf{M}[\beta] - \beta\cdot\Delta$ and, to distinguish the two worlds, check if the consistency (1) holds for $\mathsf{K}[\beta]$ or not. As in our cGGM-based single-point COT, our simulator addresses this issue by extracting every possible $\mathsf{K}[\beta]$ and the associated $\Delta = \beta^{-1}\cdot(\mathsf{M}[\beta] - \mathsf{K}[\beta])$, querying the single-point sVOLE functionality with $\Delta$, and programming the random permutation and its inverse if the global-key query succeeds.

**Subfield VOLE from Pseudorandom Correlated GGM.** There is another single-point sVOLE blueprint [10,43] basing its security on the pseudorandomness of GGM-tree nodes: for some path $\alpha\in\{0,1\}^n$, the $n$ off-path nodes and the $\alpha$-th leaf are pseudorandom. Our cGGM tree cannot be used in this blueprint since its same-level nodes are correlated under the global offset. However, we observe that a cGGM tree can be modified into a *pseudorandom* cGGM (pcGGM) tree with the required pseudorandomness.

In an $n$-level pcGGM tree, we preserve the cGGM invariant for the $\mathbb{F}_{2^\lambda}$ nodes on the first $n-1$ levels, i.e., using a hash function $\mathsf{H}':\mathbb{F}_{2^\lambda}\to\mathbb{F}_{2^\lambda}$ and Davies-Meyer construction to keep that all same-level nodes are XORed to a global offset $\Delta\in\mathbb{F}_{2^\lambda}$. Nevertheless, we break the last-level correlation in the pcGGM tree: every parent $x\in\mathbb{F}_{2^\lambda}$ on the $(i-1)$-th level has left child $\mathsf{H}'(x)$ and right child $\mathsf{H}'(x\oplus 1)$. In sVOLE protocols for $\mathbb{K}\neq\mathbb{F}_{2^\lambda}$, the pcGGM leaves will be further converted by the function $\mathsf{Convert}_{\mathbb{K}}:\mathbb{F}_{2^\lambda}\to\mathbb{K}$.

Our core observation for arguing the pseudorandomness of the $n+1$ pcGGM nodes is that the inputs of the hash function $\mathsf{H}'$ are of CCR forms. More specifically, a global $\Delta\in\mathbb{F}_{2^\lambda}$ offsets the two first-level nodes of the pcGGM tree and induces the first $n-1$ off-path nodes $\{X_i^{\alpha_1\cdots\alpha_{i-1}\overline{\alpha}_i}\}_{i\in[1,n-1]}$ according to (1) for $\mathsf{H}'$. Meanwhile, the last off-path node $X_n^{\alpha_1\cdots\alpha_{n-1}\overline{\alpha}_n}\in\mathbb{F}_{2^\lambda}$ and the $\alpha$-th pcGGM leaf $X_n\in\mathbb{F}_{2^\lambda}$ come from two hash calls of the following form: for $b\in\{0,1\}$,

$$X_n^{\alpha_1\cdots\alpha_{n-1}b} = \mathsf{H}'\Big(\Delta\oplus(\bigoplus\nolimits_{i\in[1,n-1]}X_i^{\alpha_1\cdots\alpha_{i-1}\overline{\alpha}_i})\oplus b\Big).$$

Intuitively, we can use a CCR hash function $\mathsf{H}'$ to argue the pseudorandomness of the $n$ off-path nodes and the $\alpha$-th leaf, which is sufficient for the single-point sVOLE blueprint. The challenge in this security reduction is to show that the CCR queries to $\mathsf{H}'$ are legal (i.e., no $(x,0)$ and $(x,1)$ for the same $x$) with overwhelming probability. We address this challenge by resorting to the observation that these inputs are *restricted* so that they are well-structured and are not arbitrarily chosen by the corrupted receiver (the only case where we need the pseudorandomness). Such restricted inputs are reminiscent of the "naturally derived keys" [28,47] in the Half-Gate garbling scheme so that we can bound the probability similarly. We defer the details to Appendix A of the full version [29]. Note that even if one uses $\mathsf{Convert}_{\mathbb{K}}$ to map the leaves into $\mathbb{K}$, the pseudorandomness of these nodes still holds due to the pseudorandomness of $\mathsf{Convert}_{\mathbb{K}}$.

By plugging our pcGGM tree into the prior single-point sVOLE blueprint, we obtain a more efficient protocol. The improvement owes to the cGGM invariant in its first $n-1$ levels. In terms of communication, the receiver can use $n-1$ precomputed random COTs to select the XORs on these levels and recover the first $n-1$ levels of the sender's pcGGM tree; in contrast, the prior protocols use a standard OT per level due to the two pseudorandom XORs. For the last level in our protocol, the two parties also need a standard OT due to the broken correlation of the two sums. Given the random-permutation-based CCR hash functions in [28], our pcGGM-based single-point sVOLE protocol is secure in the random-permutation model. In particular, this protocol can implement the single-point sVOLE functionality *without* global-key queries since $\Delta \in \mathbb{F}_{2^\lambda}$ is only used in the pcGGM tree and is not included in the sender's output.

### 3.2 DPF/DCF from Shared Pseudorandom Correlated GGM Trees

**DPF Sheme and Protocol.** Using a pcGGM-like trick, we present a new DPF scheme, followed by a more efficient distributed protocol. Recall that, in the prior DPF scheme [13], there are two parties sharing an $n$-level GGM-style tree where the $n$ nodes on some path $\alpha \in \{0,1\}^n$ are pseudorandom with LSB one, and others are zero. Then, the two-party shares of the $\alpha$-th leaf mask the DPF payload $\beta \in \mathbb{G}$. Our core observation is that we need the pseudorandom $\alpha$-th leaf to hide $\beta$, but the internal pseudorandom on-path nodes are not mandatory. Instead, the two parties can share an $n$-level pcGGM-style tree (say, spcGGM tree) where (i) the root $X_0$ and the first $n-1$ on-path nodes equal a global offset $\Delta \in \mathbb{F}_{2^\lambda}$ with $\mathsf{lsb}(\Delta) = 1$, (ii) the last on-path node (i.e., the $\alpha$-th leaf) is pseudorandom with LSB one, and (iii) other nodes are zero. As in the prior scheme, the per-party share of this tree is compressed as a key including an XOR share of the root and $n+1$ public *pseudorandom* correction words.

We explain our construction of these correction words in detail. To keep the invariant (i), the spcGGM tree uses a correction procedure different from the prior one. For each level $i \in [1, n-1]$ with a public correction word $\mathsf{CW}_i \in \mathbb{F}_{2^\lambda}$, and $b \in \{0,1\}$, the $b$-side secret child of the $(i-1)$-th on-path secret node $\langle X_{i-1} \rangle = \langle s_{i-1} \,\|\, t_{i-1} \rangle$ is defined as follows:

$$\langle X_i^{\alpha_1 \cdots \alpha_{i-1} b} \rangle := \mathsf{H}'(\langle X_{i-1} \rangle) \oplus b \cdot \langle X_{i-1} \rangle \oplus \langle t_{i-1} \rangle \cdot \mathsf{CW}_i.$$

Solving this *linear* equation for the *public* $\mathsf{CW}_i$ under the constraint (i), we have

$$\mathsf{CW}_i = \mathsf{H}'(\langle X_{i-1} \rangle_0) \oplus \mathsf{H}'(\langle X_{i-1} \rangle_1) \oplus \overline{\alpha}_i \cdot \Delta.$$

As for (ii), we use a public correction word $\mathsf{CW}_n = (\mathsf{HCW}, \mathsf{LCW}^0, \mathsf{LCW}^1) \in \mathbb{F}_{2^{\lambda-1}} \times \mathbb{F}_2 \times \mathbb{F}_2$ to follow the same last-level correction as the prior work. For $b \in \{0,1\}$, define a function $\mathsf{H}'_b(\cdot) := \mathsf{H}'(\cdot \oplus b)$ and the $b$-side secret child of the $(n-1)$-th on-path secret node $\langle X_{n-1} \rangle = \langle s_{n-1} \,\|\, t_{n-1} \rangle$ as follows:

$$\langle X_n^{\alpha_1 \cdots \alpha_{n-1} b} \rangle := \mathsf{H}'_b(\langle X_{n-1} \rangle) \oplus \langle t_{n-1} \rangle \cdot (\mathsf{HCW} \,\|\, \mathsf{LCW}^b).$$

Solving this *linear* equation for the *public* $\mathsf{CW}_n$ under the constraint (i) and (iii),

$$\begin{aligned} \mathsf{HCW} &= \mathsf{hb}\Big(\mathsf{H}'_{\overline{\alpha}_n}(\langle X_{n-1}\rangle_0) \oplus \mathsf{H}'_{\overline{\alpha}_n}(\langle X_{n-1}\rangle_1)\Big), \\ \forall b \in \{0,1\} : \mathsf{LCW}^b &= \mathsf{lsb}\Big(\mathsf{H}'_b(\langle X_{n-1}\rangle_0) \oplus \mathsf{H}'_b(\langle X_{n-1}\rangle_1)\Big) \oplus \alpha_n \oplus \overline{b}. \end{aligned} \tag{2}$$

Note that the $n$ off-path secret nodes $\{\langle X_i^{\alpha_1 \dots \alpha_{i-1}\overline{\alpha}_i}\rangle\}_{i\in[1,n]}$ are zero secrets according to the above correction procedures. As a result, the two parties hold identical shares of these $n$ off-path nodes and their subtrees, given that the share of a subtree is fully determined by the share of its root (i.e., an off-path node) and the public correction words. This implies the constraint (iii). Finally, the $(n+1)$-th public correction word is defined from the secret $\alpha$-th leaf $\langle X_n\rangle = \langle s_n \,\|\, t_n\rangle$ and the function $\mathsf{Convert}_{\mathbb{G}} : \mathbb{F}_{2^{\lambda-1}} \to \mathbb{G}$ as follows:

$$\mathsf{CW}_{n+1} = (\langle t_n\rangle_0 - \langle t_n\rangle_1) \cdot \Big(\mathsf{Convert}_{\mathbb{G}}(\langle s_n\rangle_1) - \mathsf{Convert}_{\mathbb{G}}(\langle s_n\rangle_0) + \beta\Big) \in \mathbb{G},$$

where the DPF payload $\beta$ is masked by the XOR shares of the $\alpha$-th leaf.

The DPF security primarily follows from that the first $n$ correction words are of CCR forms, i.e., for $i \in [0, n-1]$, $\langle X_i\rangle_0 \oplus \langle X_i\rangle_1 = X_i = \Delta$ according to the XOR secret sharing scheme and the invariant (i). The $\Delta$-circular correlation in $\mathsf{CW}_1, \dots, \mathsf{CW}_{n-1}$ is obvious for either corrupted party. In $\mathsf{CW}_n$, the honest party's $\mathsf{H}'$ inputs also differ from the corrupted party's $\mathsf{H}'$ inputs by $\Delta$. Intuitively, these $n$ correction words use CCR responses as one-time pads, and the underlying CCR queries are as structured as those in the original $\mathsf{pcGGM}$ tree. By using a CCR $\mathsf{H}'$ and upper bounding the probability of illegal CCR queries, we can prove the pseudorandomness of the first $n$ correction words and the high $\lambda - 1$ bits (i.e., $s_n$) of the $\alpha$-th leaf. The pseudorandom $s_n = \langle s_n\rangle_0 \oplus \langle s_n\rangle_1$ and $\mathsf{Convert}_{\mathbb{G}}$ ensure the pseudorandom $\mathsf{CW}_{n+1}$ for either corrupted party.

Our DPF scheme enables a more efficient distributed key generation protocol due to the construction of the first $n-1$ correction words. The insight is that the two parties, who share $\langle\alpha\rangle$ and $\langle\beta\rangle^{\mathsf{A}}$, can use their precomputed COT tuples to set up a secret $\langle\Delta\rangle$ with $\mathsf{lsb}(\Delta) = 1$ and share $\{\langle\overline{\alpha}_i \cdot \Delta\rangle\}_{i\in[1,n]}$ in two rounds, and use the black-box evaluation technique in [22] to locally share each secret $\mathsf{H}'(\langle X_{i-1}\rangle)$. This technique relies on the invariant (iii) so that, for each $i \in [1, n]$, summing the shares of the $2^i$ nodes on the $i$-th level returns the share of the $i$-th level on-path node. Given the two-party shares of $\langle\overline{\alpha}_i \cdot \Delta\rangle$ and $\mathsf{H}'(\langle X_{i-1}\rangle)$, the secure computation of each $\mathsf{CW}_i$ only needs one round for revealing $\langle\mathsf{CW}_i\rangle$, leading to $n-1$ rounds for the first $n-1$ correction words in total. In contrast, the prior protocol [22] uses (2) for each correction word, and the $i$-th level $\mathsf{HCW}$ depends on $\overline{\alpha}_i$ and should be computed level-by-level. Thus, it securely computes the first $n-1$ correction words in $2(n-1)$ rounds: for each level, one round is to share $\langle\mathsf{CW}_i\rangle$ from standard OTs, and another round is to reveal this secret.

We remark that our $\mathsf{CW}_{n+1}$ construction uses $\langle t_n\rangle_0 - \langle t_n\rangle_1$ to replace the $(-1)^{\langle t_n\rangle_1}$ term in the prior construction. The correctness is unaffected due to the non-zero LSB (i.e., $t_n$) of the $\alpha$-th leaf. However, when $\mathbb{G}$ is a ring, our $\mathsf{CW}_{n+1}$ allows the two parties to locally share $\langle t_n\rangle_0 - \langle t_n\rangle_1$ on the ring via the

black-box evaluation technique [22]. Thus, the secure computation of $\mathsf{CW}_{n+1}$ uses only one secure multiplication of two locally shared ring operands.

**DCF Scheme and Protocol.** We further show that our spcGGM tree can be extended to realize more efficient DCF scheme and its distributed protocol. Note that comparison function $f^{<}_{\alpha,\beta}(x)$ can be written as the sum of point function $f^{\bullet}_{\alpha,-\alpha_n\cdot\beta}(x)$ and a prefix function $V_{\alpha,\beta}(x)$, which returns $\alpha_{h+1}\cdot\beta\in\mathbb{G}$ such that $\alpha_1\dots\alpha_h = x_1\dots x_h$ is the longest common prefix of $\alpha$ and $x$ (for completeness, $\alpha_{n+1}:=\alpha_n$). We have shown how to realize the DPF scheme for point function $f^{\bullet}_{\alpha,-\alpha_n\cdot\beta}(x)$ from spcGGM tree. Then, we want to compute $V_{\alpha,\beta}(x)$ by reusing the prefix information with respect to $\alpha$ and $x$ when traversing the spcGGM tree to evaluate the point function. Following the GGM-style DCF scheme [7], we do this by introducing more nodes to the spcGGM tree and an additional correction procedure to ensure that the sum of the introduced nodes along the path $x$ equals $V_{\alpha,\beta}(x)$. However, our extended spcGGM tree can use less nodes and simpler correction words to compute $V_{\alpha,\beta}(x)$.

To give more details, we first recall how [7] works. It extends a shared GGM tree by replacing its length-doubling PRG with a length-quadrupling PRG so that each secret parent spawns two more secret children in $\mathbb{F}_{2^\lambda}$. For each level $i\in[1,n]$, let $\langle v_i^0\rangle$ and $\langle v_i^1\rangle$ denote such two secret children of the $(i-1)$-th on-path secret parent $\langle X_{i-1}\rangle = \langle s_{i-1}\,\|\,t_{i-1}\rangle$, and the two parties correct their additive shares for $V_{\alpha,\beta}(x)$ via the public correction word $\mathsf{VCW}_i$:

$$\mathsf{V}_{i-1} := \textstyle\sum_{b\in\{0,1\}}(-1)^{1-b}\cdot\Big(\mathsf{Convert}_{\mathbb{G}}(\langle v_{i-1}^{\overline{\alpha}_{i-1}}\rangle_b) - \mathsf{Convert}_{\mathbb{G}}(\langle v_{i-1}^{\alpha_{i-1}}\rangle_b)\Big)\in\mathbb{G},$$

$$\mathsf{VCW}_i := (-1)^{\langle t_{i-1}\rangle_1}\cdot\Big((\mathsf{Convert}_{\mathbb{G}}(\langle v_i^{\overline{\alpha}_i}\rangle_1) - \mathsf{Convert}_{\mathbb{G}}(\langle v_i^{\overline{\alpha}_i}\rangle_0)) - \mathsf{V}_{i-1} + (\alpha_i-\alpha_{i-1})\cdot\beta\Big)\in\mathbb{G}.\quad(\mathsf{V}_0:=0\in\mathbb{G},\alpha_0=0)$$

The DCF key per party includes its DPF key for $f^{\bullet}_{\alpha,-\alpha_n\cdot\beta}(x)$ and $\{\mathsf{VCW}_i\}_{i\in[1,n]}$. The DCF security also requires the pseudorandomness of the $n$ $\mathsf{VCW}_i$'s.

In contrast, our DCF scheme shows that it is overkill to introduce two more secret children to each secret parent for the DCF security. For each $i\in[1,n]$, one additional secret child $\langle v_i\rangle = \langle v_i^0\rangle = \langle v_i^1\rangle$ of the secret parent $\langle X_{i-1}\rangle$ suffices, and the pseudorandomness of $\mathsf{VCW}_i$ relies on a random $v_i = \langle v_i\rangle_0\oplus\langle v_i\rangle_1\in\mathbb{F}_{2^\lambda}$ as $\mathsf{Convert}_{\mathbb{G}}$ maps random strings to pseudorandom $\mathbb{G}$ elements. We can argue the pseudorandomness of $v_i$ based on the CCR induced by $X_{i-1}=\Delta$, if we use $v_i := \mathsf{H}'(\langle X_{i-1}\rangle_0\oplus 2)\oplus\mathsf{H}'(\langle X_{i-1}\rangle_1\oplus 2)$. Collecting all $\mathsf{H}'$ inputs for the DPF part and $v_i$'s, we find that these inputs are as structured as those in the original $\mathsf{pcGGM}$ tree. The DCF security can follow from a similar hybrid argument.

Our DCF protocol is extended from our DPF protocol with the additional secure computation of $\{\mathsf{VCW}_i\}_{i\in[1,n]}$. Compared with the prior work, our DCF protocol achieves better efficiency due to not only its optimized DPF part but also the structure of each $\mathsf{VCW}_i$. This structure makes the $\mathsf{Convert}_{\mathbb{G}}$ difference term independent of $\overline{\alpha}_i$. This independence allows the two parties to locally share the $\mathsf{Convert}_{\mathbb{G}}$ difference via the black-box evaluation technique [22], in contrast

to the technique plus OT-based 2PC in the prior protocol. Since there is only one more secret child for each secret parent, the local computation for sharing this difference is halved as well. We can also replace the $(-1)^{\langle t_{i-1}\rangle_1}$ term in the prior $\mathsf{VCW}_i$ construction by a linear term $\langle t_{i-1}\rangle_0 - \langle t_{i-1}\rangle_1$, which can be locally shared via the same black-box evaluation technique if $\mathbb{G}$ is a ring. As a result, except the 2PC for sharing $\{\langle \alpha_i \cdot \beta\rangle^{\mathsf{A}}\}_{i\in[1,n]}$, the secure computation of $\{\mathsf{VCW}_i\}_{i\in[1,n]}$ requires $n$ secure multiplications of two shared ring elements. These secure multiplications can run in parallel with that for $\mathsf{CW}_{n+1}$.

In our DCF protocol, each $\langle \alpha_i \cdot \beta\rangle^{\mathsf{A}}$ is secretly shared by carefully reusing the two precomputed COT tuples, which were used to share $\langle \overline{\alpha}_i \cdot \Delta\rangle$, to run a COT-based multiplication between the XOR shared $\alpha_i$ and the additively shared $\beta$ on the ring. This multiplication generalizes the binary case [1,28] for an XOR shared bit and an XOR shared string by using the well-known arithmetic XOR on the ring: $\langle \alpha_i\rangle_0 \oplus \langle \alpha_i\rangle_1 = \langle \alpha_i\rangle_0 + \langle \alpha_i\rangle_1 - 2 \cdot \langle \alpha_i\rangle_0 \cdot \langle \alpha_i\rangle_1$.

**Functionality $\mathcal{F}_{\mathsf{spsVOLE}}$**

**Parameters:** Field $\mathbb{F}$ and its extension field $\mathbb{K}$.

**Initialize:** Upon receiving (init) from $P_0$ and $P_1$, sample $\Delta \leftarrow \mathbb{K}$ if $P_0$ is honest; otherwise, receive $\Delta \in \mathbb{K}$ from the adversary. Store $\Delta$ and send it to $P_0$. Ignore all subsequent (init) commands.

**Extend:** This functionality allows polynomially many (extend) commands. Upon receiving $(\mathsf{extend}, N)$ from $P_0$ and $P_1$:

1. If $P_0$ is honest, sample $\mathbf{v} \leftarrow \mathbb{K}^N$; otherwise, receive $\mathbf{v} \in \mathbb{K}^N$ from the adversary.
2. If $P_1$ is honest, sample $\mathbf{u} \leftarrow \mathbb{F}^N$ with exactly one nonzero entry, and compute $\mathbf{w} := \mathbf{v} + \mathbf{u} \cdot \Delta \in \mathbb{K}^N$; otherwise, receive $(\mathbf{u}, \mathbf{w}) \in \mathbb{F}^N \times \mathbb{K}^N$ from the adversary, where $\mathbf{u}$ has at most one nonzero entry, and recompute $\mathbf{v} := \mathbf{w} - \mathbf{u} \cdot \Delta \in \mathbb{K}^N$.
3. Send $\mathbf{v}$ to $P_0$ and $(\mathbf{u}, \mathbf{w})$ to $P_1$.

**Global-key queries:** If $P_1$ is corrupted, upon receiving $(\mathsf{guess}, \Delta')$, where $\Delta' \in \mathbb{K}$, from the adversary, send (success) to the adversary if $\Delta = \Delta'$; send (fail) to the adversary otherwise.

**Fig. 2.** Functionality for single-point subfield VOLE.

## 4 Subfield VOLE Extension

Our sVOLE extension follows the blueprint of [10,42,43,46], which uses LPN to locally convert $t$ single-point sVOLE (spsVOLE) tuples output by functionality $\mathcal{F}_{\mathsf{spsVOLE}}$ (Fig. 2) into an sVOLE tuple. We focus on the efficient spsVOLE protocol that UC-realizes $\mathcal{F}_{\mathsf{spsVOLE}}$. Note that the spsVOLE protocol dominates the computation and contributes all communication in sVOLE extension.

$\mathcal{F}_{\mathsf{spsVOLE}}$ is parameterized by a field $\mathbb{F}$ and its extension $\mathbb{K}$, and covers the single-point COT functionality $\mathcal{F}_{\mathsf{spCOT}}$ if $\mathbb{F} = \mathbb{F}_2$ and $\mathbb{K} = \mathbb{F}_{2^\lambda}$. This functionality

is the same as that in [43], except that $\mathcal{F}_{\mathsf{spsVOLE}}$ will not abort for an incorrect global-key query. Allowing for global-key queries has been considered in [32,38] and does not weaken the effective security. In the spsVOLE protocol based on pseudorandom correlated GGM, such global-key queries can be removed.

In essence, our spsVOLE protocols work as the PCG protocol [10–12,18] of spsVOLE correlation, although we do not divide the correlation generation into two explicit PCG phases. In Appendix E.1 of the full version [29], we show how to modify one of our spsVOLE protocols to define such two phases, in order to satisfy the "silent property" that a long spsVOLE tuple can be stored as two sublinearly short correlated seeds.

### 4.1 Single-Point COT and sVOLE from Correlated GGM

In Fig. 3, we present the two evaluation algorithms for our correlated GGM tree, which is defined by two first-level nodes $(k, \Delta - k) \in \mathbb{K}^2$. For every non-leaf node $x \in \mathbb{K}$, its left child is defined as $\mathsf{H}(x) \in \mathbb{K}$ while its right child is defined as $x - \mathsf{H}(x) \in \mathbb{K}$. The following claim is straightforward from an induction.

**Parameters:** Tree depth $n \in \mathbb{N}$. Field $\mathbb{K}$ with $|\mathbb{K}| \geq 2^\lambda$. Hash function $\mathsf{H} : \mathbb{K} \to \mathbb{K}$.

$\mathsf{cGGM.FullEval}(\Delta, k)$: Given $(\Delta, k) \in \mathbb{K}^2$,

1: $X_1^0 := k \in \mathbb{K}$, $X_1^1 := \Delta - k \in \mathbb{K}$.
2: **for** $i \in [2, n]$, $j \in [0, 2^{i-1})$ **do**
3: $\quad X_i^{2j} := \mathsf{H}(X_{i-1}^j) \in \mathbb{K}$, $X_i^{2j+1} := X_{i-1}^j - X_i^{2j} \in \mathbb{K}$.
4: $\mathbf{v} := (X_n^0, \ldots, X_n^{2^n-1}) \in \mathbb{K}^{2^n}$.
5: **for** $i \in [1, n]$ **do** $K_i^0 := \sum_{j \in [0, 2^{i-1})} X_i^{2j} \in \mathbb{K}$.
6: **return** $(\mathbf{v}, \{K_i^0\}_{i \in [1,n]})$

$\mathsf{cGGM.PuncFullEval}(\alpha, \{K_i^{\overline{\alpha}_i}\}_{i \in [1,n]})$: Given $(\alpha, \{K_i^{\overline{\alpha}_i}\}_{i \in [1,n]}) \in \{0,1\}^n \times \mathbb{K}^n$,

1: $X_1^{\overline{\alpha}_1} := K_1^{\overline{\alpha}_1} \in \mathbb{K}$.
2: **for** $i \in [2, n]$ **do**
3: $\quad$ **for** $j \in [0, 2^{i-1})$, $j \neq \alpha_1 \ldots \alpha_{i-1}$ **do**
4: $\quad\quad X_i^{2j} := \mathsf{H}(X_{i-1}^j) \in \mathbb{K}$, $X_i^{2j+1} := X_{i-1}^j - X_i^{2j} \in \mathbb{K}$.
5: $\quad X_i^{\alpha_1 \ldots \alpha_{i-1} \overline{\alpha}_i} := K_i^{\overline{\alpha}_i} - \sum_{j \in [0, 2^{i-1}), j \neq \alpha_1 \ldots \alpha_{i-1}} X_i^{2j+\overline{\alpha}_i} \in \mathbb{K}$.
6: $X_n^\alpha := -\sum_{j \in [0, 2^n), j \neq \alpha} X_n^j \in \mathbb{K}$, $\mathbf{w} := (X_n^0, \ldots, X_n^{2^n-1}) \in \mathbb{K}^{2^n}$.
7: **return** $\mathbf{w}$

**Fig. 3.** Two full-evaluation algorithms for correlated GGM tree.

*Claim (Leveled correlation).* For any two first-level nodes $(k, \Delta - k) \in \mathbb{K}^2$ and any $i \in [1, n]$, the offset $\Delta \in \mathbb{K}$ equals the sum of all nodes on the $i$-th level of the correlated GGM tree expanded from $(k, \Delta - k)$ as per $\mathsf{cGGM.FullEval}$.

**Corollary 1.** *For any* $\alpha \in [0, 2^n)$*, any* $(k, \Delta - k) \in \mathbb{K}^2$*, and*

$$(\mathbf{v}, \{K_i^0\}_{i\in[1,n]}) := \mathsf{cGGM.FullEval}(\Delta, k),$$
$$\mathbf{w} := \mathsf{cGGM.PuncFullEval}(\alpha, \{K_i^{\overline{\alpha}_i}\}_{i\in[1,n]}),$$

*where* $K_i^{\overline{\alpha}_i} := \overline{\alpha}_i \cdot \Delta + (-1)^{\overline{\alpha}_i} \cdot K_i^0$ *for* $i \in [1, n]$*, we have* $\mathbf{w}^{(\alpha)} - \mathbf{v}^{(\alpha)} = -\Delta$*.*

*Proof.* Claim 4.1 and the definition of $\mathsf{cGGM.FullEval}$ imply that $K_i^{\overline{\alpha}_i} \in \mathbb{K}$ in this corollary defines the sum of all $\overline{\alpha}_i$-side nodes on the $i$-th level of the correlated GGM tree. Then, it follows from the definition of $\mathsf{cGGM.PuncFullEval}$ that $\mathbf{v}^{(j)} = \mathbf{w}^{(j)}$ for any $j \neq \alpha \in [0, 2^n)$. Using Claim 4.1 for the last level, we have $\mathbf{w}^{(\alpha)} - \mathbf{v}^{(\alpha)} = -\sum_{j\in[0,2^n), j\neq\alpha} \mathbf{w}^{(j)} - \mathbf{v}^{(\alpha)} = -\sum_{j\in[0,2^n), j\neq\alpha} \mathbf{v}^{(j)} - \mathbf{v}^{(\alpha)} = -\Delta$.

**Single-Point COT.** Figure 4 describes our single-point COT protocol $\Pi_{\mathsf{spCOT}}$ that runs in the $\mathcal{F}_{\mathsf{COT}}$-hybrid model and uses the $\mathsf{cGGM}$ expansion in Fig. 3.

**The same $\Delta$in correlated GGM trees.** Note that $\mathcal{F}_{\mathsf{spCOT}}$ produces single-point COT tuples with the same global key $\Delta \in \mathbb{F}_{2^\lambda}$ in a number of **Extend** executions. To realize $\mathcal{F}_{\mathsf{spCOT}}$, our protocol $\Pi_{\mathsf{spCOT}}$ proceeds as sketched in Sect. 3.1 but uses the same $\Delta$ for the $\mathsf{cGGM}$ trees of these executions, each of which samples a fresh $k \leftarrow \mathbb{F}_{2^\lambda}$ for $\mathsf{cGGM.FullEval}(\Delta, k)$. A merit of using the same $\Delta$ in several tree instances is that $\Pi_{\mathsf{spCOT}}$ only invokes one $\mathcal{F}_{\mathsf{COT}}$ instance, and the amortized cost per precomputed COT tuple can be small.

**Protocol $\Pi_{\mathsf{spCOT}}$**

**Parameters:** Field $\mathbb{F}_2$ and its extension field $\mathbb{F}_{2^\lambda}$.

**Initialize:** This procedure is executed only once.

1. $P_0$ and $P_1$ send $(\mathsf{init})$ to $\mathcal{F}_{\mathsf{COT}}$, which returns $\Delta \in \mathbb{F}_{2^\lambda}$ to $P_0$. $P_0$ outputs $\Delta$.

**Extend:** This procedure can be executed many times. $P_0$ and $P_1$ input $N = 2^n$ and use $\mathsf{cGGM}$ (c.f. Figure 3) for $n$ and $\mathbb{F}_{2^\lambda}$.

2. $P_0$ and $P_1$ send $(\mathsf{extend}, n)$ to $\mathcal{F}_{\mathsf{COT}}$, which returns $(\mathsf{K}[r_1], \ldots, \mathsf{K}[r_n]) \in \mathbb{F}_{2^\lambda}^n$ to $P_0$ and $((r_1, \ldots, r_n), (\mathsf{M}[r_1], \ldots, \mathsf{M}[r_n])) \in \mathbb{F}_2^n \times \mathbb{F}_{2^\lambda}^n$ to $P_1$ such that $\mathsf{M}[r_i] = \mathsf{K}[r_i] \oplus r_i \cdot \Delta$ for $i \in [1, n]$.
3. $P_0$ samples $c_1 \leftarrow \mathbb{F}_{2^\lambda}$ and sets $k := \mathsf{K}[r_1] \oplus c_1$,

$$(\mathbf{v}, \{K_i^0\}_{i\in[1,n]}) := \mathsf{cGGM.FullEval}(\Delta, k),$$

   and $c_i := \mathsf{K}[r_i] \oplus K_i^0$ for $i \in [2, n]$. $P_0$ sends $(c_1, \ldots, c_n)$ to $P_1$.
4. $P_1$ sets $\alpha = \alpha_1 \ldots \alpha_n := \overline{r}_1 \ldots \overline{r}_n \in [0, N)$, $K_i^{\overline{\alpha}_i} := \mathsf{M}[r_i] \oplus c_i$ for $i \in [1, n]$, and

$$\mathbf{u} := \mathbf{unit}_{\mathbb{F}_2}(N, \alpha, 1), \quad \mathbf{w} := \mathsf{cGGM.PuncFullEval}(\alpha, \{K_i^{\overline{\alpha}_i}\}_{i\in[1,n]}).$$

5. $P_0$ outputs $\mathbf{v}$ and $P_1$ outputs $(\mathbf{u}, \mathbf{w})$.

**Fig. 4.** $\mathsf{cGGM}$-based single-point COT protocol in the $\mathcal{F}_{\mathsf{COT}}$-hybrid model.

**Security.** We prove Theorem 1 by following the sketched intuition in Sect. 3.1 and defer the proof to Appendix B.1 of the full version [29]. Our proof considers polynomially many concurrent **Extend** executions (strictly speaking, sub-sessions with unique sub-session IDs) that uses the one-time initialized $\Delta$.

**Theorem 1.** *Given random permutation* $\pi : \mathbb{F}_{2^\lambda} \to \mathbb{F}_{2^\lambda}$*, efficiently computable linear orthomorphism* $\sigma : \mathbb{F}_{2^\lambda} \to \mathbb{F}_{2^\lambda}$ *with efficiently computable* $\sigma^{-1}$*,* $\sigma'(x) := \sigma(x) \oplus x$*, and* $\sigma'^{-1}$ *(Footnote 1), and hash function* $\mathsf{H}(x) := \pi(\sigma(x)) \oplus \sigma(x)$*, protocol* $\Pi_{\mathsf{spCOT}}$ *(Fig. 4) UC-realizes functionality* $\mathcal{F}_{\mathsf{spCOT}}$ *(Fig. 2) against any semi-honest adversary in the* $\mathcal{F}_{\mathsf{COT}}$*-hybrid model and the RPM.*

**Communication Optimization.** For $t$ concurrent **Extend** executions (e.g., in COT extension), the random $c_1$'s in these executions can be compressed via a PRF $F : \mathbb{F}_{2^\lambda} \times \{0,1\}^* \to \mathbb{F}_{2^\lambda}$. Concretely, $P_0$ samples a PRF key $k_{\mathsf{prf}} \leftarrow \mathbb{F}_{2^\lambda}$ *after receiving its COT outputs in all executions* and sends this key to $P_1$. For each execution with sub-session ID $\mathsf{ssid}$, the two parties locally defines the element $c_1 := F(k_{\mathsf{prf}}, \mathsf{ssid})$. This PRF key is only used for the $t$ concurrent executions. The security of this optimization follows from the PRF security and the fact that, in the concurrent executions, the COT messages chosen by the corrupted receiver cannot depend on the PRF key to be sampled by the honest sender.

**Complexity Analysis.** Consider the complexity per execution when the PRF-based optimization is used in $t$ concurrent **Extend** executions. $\Pi_{\mathsf{spCOT}}$ needs $n$ precomputed COT tuples. $P_0$ sends $(n-1) \cdot \lambda + \frac{\lambda}{t}$ bits, and $P_1$ sends nothing. The computation per party comes from the tree expansion with $N$ RP calls.

In the $\mathcal{F}_{\mathsf{COT}}$-hybrid model, the prior single-point COT protocol [46] consumes $n$ precomputed COT tuples. However, $P_0$ sends $2n \cdot \lambda$ bits. Each party performs about $N$ length-doubling PRG calls, which in turn result in $2N$ RP calls. We can see that our protocol halves both the computation and communication in the prior work. When looking at the whole protocol, the improvement is still huge. For example, the micro benchmark in Silver [18] reported that 70% of the time is spent on GGM-tree-related computation, and thus our protocol will lead to at least 50% of end-to-end computational improvement in COT.

**Single-Point sVOLE.** We can also realize single-point sVOLE from our cGGM tree by using the high-level idea sketched in Sect. 3.1. This protocol extends $\Pi_{\mathsf{spCOT}}$ by using a cGGM tree whose nodes are in a general *exponentially large* extension field $\mathbb{K}$. The tree expansion therein uses a hash function constructed from a random permutation and a linear orthomorphism over $\mathbb{K}$. We defer the detailed protocol and its security proof to Appendix B.2 of the full version [29].

### 4.2 Single-Point sVOLE from Pseudorandom Correlated GGM

We can adapt our correlated GGM tree for a *pseudorandom* correlated one with the property that the leaf node at some punctured position $\alpha$ is pseudorandom.

This pseudorandom correlated GGM tree pcGGM is defined in Fig. 5, where the first $n-1$ levels preserve the correlation in Claim 4.1 but all last-level nodes are processed by $\mathsf{H}_S$ to break this correlation. The keyed hash function $\mathsf{H}_S$ uses some key $S \in \mathbb{F}_{2^\lambda}$, which can be sampled by the receiver in single-point sVOLE and, for simplicity, is assumed to have been sent to the sender before protocol execution. The implementation of $\mathsf{H}_S$ is given in Theorem 2. In fact, this pcGGM tree yields PPRF, which is proved in Appendix C of the full version [29].

The pseudorandomness only at the cost of the last-level correlation allows us to follow the single-point sVOLE blueprint in [10,43] but also take advantage of the correlation in the first $n-1$ levels. The protocol is presented in Fig. 6. In this protocol, the sender $P_0$ only sends $\lambda$ bits to the receiver $P_1$ for each of the first $n-1$ levels, given a precomputed COT tuple. For the last level, the two parties use a COT tuple and the standard technique [3,34] to emulate the string OT as in the prior protocols. To amortize the cost per precomputed COT tuple, the pcGGM trees in many **Extend** executions also use the same $\Delta$ set by $\mathcal{F}_{\mathsf{COT}}$.

**Security.** The security against the semi-honest $P_0$ resorts to the one-time pad $s$ from $\mathcal{F}_{\mathsf{sVOLE}}$. Meanwhile, the security against the semi-honest $P_1$ relies on that (i)

**Parameters:** Tree depth $n \in \mathbb{N}$. Field $\mathbb{K}$. Keyed hash function $\mathsf{H}_S : \mathbb{F}_{2^\lambda} \to \mathbb{F}_{2^\lambda}$. Function $\mathsf{Convert}_{\mathbb{K}} : \mathbb{F}_{2^\lambda} \to \mathbb{K}$.

$\mathsf{pcGGM.FullEval}(\Delta, k)$: Given $(\Delta, k) \in \mathbb{F}_{2^\lambda}^2$,

$X_1^0 := k \in \mathbb{F}_{2^\lambda}$, $X_1^1 := \Delta \oplus k \in \mathbb{F}_{2^\lambda}$.
**for** $i \in [2, n-1]$, $j \in [0, 2^{i-1})$ **do**
  $X_i^{2j} := \mathsf{H}_S(X_{i-1}^j) \in \mathbb{F}_{2^\lambda}$, $X_i^{2j+1} := X_{i-1}^j \oplus X_i^{2j} \in \mathbb{F}_{2^\lambda}$.
**for** $j \in [0, 2^{n-1})$, $b \in \{0,1\}$ **do** $X_n^{2j+b} := \mathsf{Convert}_{\mathbb{K}}(\mathsf{H}_S(X_{n-1}^j \oplus b)) \in \mathbb{K}$.
$\mathbf{v} := (X_n^0, \ldots, X_n^{2^n-1}) \in \mathbb{K}^{2^n}$.
**for** $i \in [1, n-1]$ **do** $K_i^0 := \oplus_{j \in [0,2^{i-1})} X_i^{2j} \in \mathbb{F}_{2^\lambda}$.
$(K_n^0, K_n^1) := (\sum_{j \in [0,2^{n-1})} X_n^{2j}, \sum_{j \in [0,2^{n-1})} X_n^{2j+1}) \in \mathbb{K}^2$.
**return** $(\mathbf{v}, \{K_i^0\}_{i \in [1,n-1]}, (K_n^0, K_n^1))$

$\mathsf{pcGGM.PuncFullEval}(\alpha, \{K_i^{\overline{\alpha}_i}\}_{i \in [1,n]}, \gamma)$: Given $(\alpha, \{K_i^{\overline{\alpha}_i}\}_i, \gamma) \in \{0,1\}^n \times \mathbb{K}^n \times \mathbb{K}$,

$X_1^{\overline{\alpha}_1} := K_1^{\overline{\alpha}_1} \in \mathbb{F}_{2^\lambda}$.
**for** $i \in [2, n-1]$ **do**
  **for** $j \in [0, 2^{i-1})$, $j \neq \alpha_1 \ldots \alpha_{i-1}$ **do**
    $X_i^{2j} := \mathsf{H}_S(X_{i-1}^j) \in \mathbb{F}_{2^\lambda}$, $X_i^{2j+1} := X_{i-1}^j \oplus X_i^{2j} \in \mathbb{F}_{2^\lambda}$.
  $X_i^{\alpha_1 \cdots \alpha_{i-1}\overline{\alpha}_i} := K_i^{\overline{\alpha}_i} \oplus (\oplus_{j \in [0,2^{i-1}), j \neq \alpha_1 \ldots \alpha_{i-1}} X_i^{2j+\overline{\alpha}_i}) \in \mathbb{F}_{2^\lambda}$.
**for** $j \in [0, 2^{n-1})$, $j \neq \alpha_1 \ldots \alpha_{n-1}$, $b \in \{0,1\}$ **do**
  $X_n^{2j+b} := \mathsf{Convert}_{\mathbb{K}}(\mathsf{H}_S(X_{n-1}^j \oplus b)) \in \mathbb{K}$.
$X_n^{\alpha_1 \ldots \alpha_{n-1}\overline{\alpha}_n} := K_n^{\overline{\alpha}_n} - \sum_{j \in [0,2^{n-1}), j \neq \alpha_1 \ldots \alpha_{n-1}} X_n^{2j+\overline{\alpha}_n} \in \mathbb{K}$.
$X_n^\alpha := \gamma - \sum_{j \in [0,2^n), j \neq \alpha} X_n^j \in \mathbb{K}$, $\mathbf{w} := (X_n^0, \ldots, X_n^{2^n-1}) \in \mathbb{K}^{2^n}$.
**return** $\mathbf{w}$

**Fig. 5.** Two full-evaluation algorithms for pseudorandom correlated GGM tree.

the pcGGM tree with a CCR structure has $n$ pseudorandom off-path nodes and the punctured leaf, giving pseudorandom $c_1, \ldots, c_{n-1}$ and $(c_n^{r_n}, \psi)$, and (ii) the mask of the unselected message $c_n^{\bar{r}_n}$ in the emulated last-level OT is computed by applying $\mathsf{Convert}_{\mathbb{K}}$ to a CCR response, which is for a legal CCR query with overwhelming probability due to the uniform $\mu$. The proof of Theorem 2 can be found in Appendix B.3 of the full version [29], where we consider polynomially many concurrent **Extend** executions, which use the one-time initialized $\Delta$.

**Theorem 2.** *Given CCR function* $\mathsf{H} : \mathbb{F}_{2^\lambda} \to \mathbb{F}_{2^\lambda}$*, function* $\mathsf{Convert}_{\mathbb{K}} : \mathbb{F}_{2^\lambda} \to \mathbb{K}$*, and keyed hash function* $\mathsf{H}_S(x) := \mathsf{H}(S \oplus x)$ *with some key* $S \leftarrow \mathbb{F}_{2^\lambda}$*, protocol* $\Pi_{\mathsf{spsVOLE-pcGGM}}$ *(Fig. 6) UC-realizes functionality* $\mathcal{F}_{\mathsf{spsVOLE}}$ *(Fig. 2) without global-key queries against any semi-honest adversary in the* $(\mathcal{F}_{\mathsf{COT}}, \mathcal{F}_{\mathsf{sVOLE}})$*-hybrid model.*

**Communication Optimization.** $\Pi_{\mathsf{spsVOLE-pcGGM}}$ can be optimized as follows:

- The two random $(c_1, \mu)$ to be sent by the sender in $\Pi_{\mathsf{spsVOLE-pcGGM}}$ can be compressed via the PRF technique for $\Pi_{\mathsf{spCOT}}$. In $t$ concurrent **Extend** executions, all such random messages can also be compressed in batch.
- The optimization for a large field $\mathbb{F}$ in $\Pi_{\mathsf{spsVOLE-cGGM}}$ also applies.
- If $\mathbb{F} = \mathbb{F}_2$, $\Pi_{\mathsf{spsVOLE-pcGGM}}$ degenerates to single-point COT and can do away with $\mathcal{F}_{\mathsf{sVOLE}}$ so that the receiver need not send a difference $d \in \mathbb{F}$. Instead, the sender locally samples $\Gamma \in \mathbb{K}$ and masks this value with the sum of all last-level nodes in a pcGGM tree. This optimization has been used in [10].

**Complexity Analysis.** Consider the complexity per execution when the PRF-based optimization is used in $t$ concurrent **Extend** executions. $\Pi_{\mathsf{spsVOLE-pcGGM}}$ uses $n$ precomputed COT tuples and one precomputed sVOLE tuple. $P_0$ sends $(n-2) \cdot \lambda + 3 \cdot \log|\mathbb{K}| + \frac{\lambda}{t}$ bits, and $P_1$ sends $\log|\mathbb{F}|$ bits. The computation is dominated by the tree expansion with $1.5N$ RP calls for each party. Compared with the prior works [10,43], our protocol roughly halve the communication, and the reduction in computation is 25%. This computation cost includes no PRG call in $\mathsf{Convert}_{\mathbb{K}}$, which can be implemented from cheap modulo operations for the field size $|\mathbb{K}|$ considered in many sVOLE applications, e.g., [40,43–45].

**Protocol $\Pi_{\mathsf{spsVOLE-pcGGM}}$**

**Parameters:** Field $\mathbb{F}$ and its extension field $\mathbb{K}$.

**Initialize:** This procedure is executed only once.

1. $P_0$ and $P_1$ send $(\mathsf{init})$ to $\mathcal{F}_{\mathsf{COT}}$, which returns $\Delta \in \mathbb{F}_{2^\lambda}$ to $P_0$.
2. $P_0$ and $P_1$ send $(\mathsf{init})$ to $\mathcal{F}_{\mathsf{sVOLE}}$, which returns $\Gamma \in \mathbb{K}$ to $P_0$. $P_0$ outputs $\Gamma$.

**Extend:** This procedure can be executed many times. $P_0$ and $P_1$ input $N = 2^n$ and use $\mathsf{pcGGM}$ (c.f. Figure 5) for $n$, $\mathbb{K}$, keyed hash function $\mathsf{H}_S : \mathbb{F}_{2^\lambda} \to \mathbb{F}_{2^\lambda}$, and function $\mathsf{Convert}_{\mathbb{K}} : \mathbb{F}_{2^\lambda} \to \mathbb{K}$.

3. $P_0$ and $P_1$ send $(\mathsf{extend}, n)$ to $\mathcal{F}_{\mathsf{COT}}$, which returns $(\mathsf{K}[r_1], \ldots, \mathsf{K}[r_n]) \in \mathbb{F}_{2^\lambda}^n$ to $P_0$ and $((r_1, \ldots, r_n), (\mathsf{M}[r_1], \ldots, \mathsf{M}[r_n])) \in \mathbb{F}_2^n \times \mathbb{F}_{2^\lambda}^n$ to $P_1$ such that $\mathsf{M}[r_i] = \mathsf{K}[r_i] \oplus r_i \cdot \Delta$ for $i \in [1, n]$.
4. $P_0$ and $P_1$ send $(\mathsf{extend}, 1)$ to $\mathcal{F}_{\mathsf{sVOLE}}$, which returns $\mathsf{K}[s] \in \mathbb{K}$ to $P_0$ and $(s, \mathsf{M}[s]) \in \mathbb{F} \times \mathbb{K}$ to $P_1$ such that $\mathsf{M}[s] = \mathsf{K}[s] + s \cdot \Gamma$.
5. $P_1$ samples $\beta \leftarrow \mathbb{F}^*$, sets $\mathsf{M}[\beta] := \mathsf{M}[s]$, and sends $d := s - \beta \in \mathbb{F}$ to $P_0$.
   $P_0$ sets $\mathsf{K}[\beta] := \mathsf{K}[s] + d \cdot \Gamma$ such that $\mathsf{M}[\beta] = \mathsf{K}[\beta] + \beta \cdot \Gamma$.
6. $P_0$ samples $(c_1, \mu) \leftarrow \mathbb{F}_{2^\lambda}^2$ and sets $k := \mathsf{K}[r_1] \oplus c_1$,

$$(\mathbf{v}, \{K_i^0\}_{i \in [1,n-1]}, (K_n^0, K_n^1)) := \mathsf{pcGGM.FullEval}(\Delta, k),$$

   $c_i := \mathsf{K}[r_i] \oplus K_i^0$ for $i \in [2, n-1]$, $c_n^b := \mathsf{Convert}_{\mathbb{K}}(\mathsf{H}_S(\mu \oplus \mathsf{K}[r_n] \oplus b \cdot \Delta)) + K_n^b$ for $b \in \{0, 1\}$, and $\psi := K_n^0 + K_n^1 - \mathsf{K}[\beta]$.
   $P_0$ sends $(c_1, \ldots, c_{n-1}, \mu, c_n^0, c_n^1, \psi)$ to $P_1$.
7. $P_1$ sets $\alpha = \alpha_1 \ldots \alpha_n := \overline{r}_1 \ldots \overline{r}_n \in [0, N)$, $K_i^{\overline{\alpha}_i} := \mathsf{M}[r_i] \oplus c_i$ for $i \in [1, n-1]$, $K_n^{\overline{\alpha}_n} := c_n^{r_n} - \mathsf{Convert}_{\mathbb{K}}(\mathsf{H}_S(\mu \oplus \mathsf{M}[r_n]))$, and

$$\mathbf{u} := \mathbf{unit}_{\mathbb{F}}(N, \alpha, \beta), \quad \mathbf{w} := \mathsf{pcGGM.PuncFullEval}(\alpha, \{K_i^{\overline{\alpha}_i}\}_{i \in [1,n]}, \psi + \mathsf{M}[\beta]).$$

8. $P_0$ outputs $\mathbf{v}$ and $P_1$ outputs $(\mathbf{u}, \mathbf{w})$.

**Fig. 6.** $\mathsf{pcGGM}$-based single-point sVOLE protocol in the $(\mathcal{F}_{\mathsf{COT}}, \mathcal{F}_{\mathsf{sVOLE}})$-hybrid model.

## 5 DPF and DCF Correlation Generation

We model DPF/DCF correlation generation in functionality $\mathcal{F}_{\mathsf{FSS}}$ (Fig. 7), which includes distributed key generation and local full-domain evaluation. By putting both procedures in the same functionality, we are able to model FSS as an ideal functionality and avoid caveats in the proof. $\mathcal{F}_{\mathsf{FSS}}$ focuses on $N = 2^n$ for $n \in \mathbb{N}$, and we can define a similar functionality for a general $N \in \mathbb{N}$. Using padding, our protocols for $\mathcal{F}_{\mathsf{FSS}}$ also works in this general case.

**Functionality $\mathcal{F}_{\mathsf{FSS}}$**

**Parameters:** Ring $\mathcal{R}$. $\mathsf{FSS} \in \{\mathsf{DPF}, \mathsf{DCF}\}$ with domain $[0, N)$, where domain size $N = 2^n$ for $n \in \mathbb{N}$, and range $\mathcal{R}$.

**Gen:** This functionality allows polynomially many ($\mathsf{gen}$) commands. Upon receiving $(\mathsf{gen}, \langle\alpha\rangle_b, \langle\beta\rangle_b^{\mathsf{A}})$ from $P_b$ for each $b \in \{0, 1\}$, where $(\langle\alpha\rangle_b, \langle\beta\rangle_b^{\mathsf{A}}) \in [0, N) \times \mathcal{R}$:

1. Set $\alpha := \langle\alpha\rangle_0 \oplus \langle\alpha\rangle_1 \in [0, N)$, $\beta := \langle\beta\rangle_0^{\mathsf{A}} + \langle\beta\rangle_1^{\mathsf{A}} \in \mathcal{R}$, and $\mathbf{r} \in \mathcal{R}^N$ such that
   - If $\mathsf{FSS} = \mathsf{DPF}$, $\mathbf{r}^{(j)} = 0$ for $j \in [0, N)$, $j \neq \alpha$, and $\mathbf{r}^{(\alpha)} = \beta$.
   - If $\mathsf{FSS} = \mathsf{DCF}$, $\mathbf{r}^{(j)} = 0$ for $j \in [0, N)$, $j \geq \alpha$, and $\mathbf{r}^{(j)} = \beta$ otherwise.
2. If both parties are honest, sample $\langle\mathbf{r}\rangle_0^{\mathsf{A}}, \langle\mathbf{r}\rangle_1^{\mathsf{A}} \leftarrow \mathcal{R}^N$ such that $\langle\mathbf{r}\rangle_0^{\mathsf{A}} + \langle\mathbf{r}\rangle_1^{\mathsf{A}} = \mathbf{r}$; otherwise (i.e., $P_b$ is corrupted), receive $\langle\mathbf{r}\rangle_b^{\mathsf{A}} \in \mathcal{R}^N$ from the adversary and recompute $\langle\mathbf{r}\rangle_{1-b}^{\mathsf{A}} := \mathbf{r} - \langle\mathbf{r}\rangle_b^{\mathsf{A}} \in \mathcal{R}^N$.
3. Send $\langle\mathbf{r}\rangle_0^{\mathsf{A}}$ to $P_0$ and $\langle\mathbf{r}\rangle_1^{\mathsf{A}}$ to $P_1$.

**Fig. 7.** Functionality for DPF/DCF correlation generation.

One can view $\mathcal{F}_{\mathsf{FSS}}$ as an alternative to the FSS key generation functionality that outputs each FSS key in the key pair to the designated party, who locally uses its key to evaluate its shares of the evaluation results at several points. We note that the full-domain evaluation included in $\mathcal{F}_{\mathsf{FSS}}$ does not complicate its implementation in contrast to the known protocols [7,22] of the FSS key generation functionality. The reason is that, using the black-box evaluation technique [22], these protocols also perform full-domain evaluation. If FSS correlations are generated for immediate use without long-term storage (e.g., [22]), $\mathcal{F}_{\mathsf{FSS}}$ can be a drop-in replacement of the FSS key generation functionality. However, we also show in Appendix E.2 of the full version [29] that our protocols for $\mathcal{F}_{\mathsf{FSS}}$ can be adapted to realize this key generation functionality.

### 5.1 DPF and DCF Schemes

Note that DPF/DCF scheme may be used in not only distributed settings (e.g., [22]) but also the scenarios where a trusted dealer is available (e.g., two-server PIR [13,25]). It would be better for us to present the two schemes alone.

We present in Fig. 8 (resp., Fig. 9) our DPF (resp., DCF) scheme, which is implicitly constructed from a *shared* pseudorandom correlated GGM tree. For simplicity of exposition, we slightly abuse the function $\mathsf{Convert}_{\mathbb{G}} : \{0, 1\}^* \to \mathbb{G}$ so that it can map random strings of either $\lambda$ or $\lambda - 1$ bits to pseudorandom group elements in $\mathbb{G}$. Our DCF scheme makes *non-black-box* use of our DPF scheme.

Note that our DPF and DCF schemes use a keyed hash function $\mathsf{H}_S$. When there is a trusted dealer, the key $S$ can be uniformly sampled by the dealer. In our DPF and DCF protocols in the upcoming sections, it can be jointly sampled by two parties using one-time public coin-tossing. This hash key can be reused across polynomially many FSS key pairs.

**Complexity Analysis.** Consider the group $\mathbb{G}$ (e.g., in [7,13,14,22,25]) with the PRG-free implementation of $\mathsf{Convert}_{\mathbb{G}}$ (c.f. Appendix F.1 of the full version [29]).

Our DPF scheme has a full-domain evaluation that takes $1.5N$ RP calls, in contrast to the $2N$ RP calls in the state-of-the-art construction of [13]. Its key generation algorithm uses about $2n+2$ RP calls while this figure is about $4n$ in the prior work. In our scheme, the key size is $n \cdot \lambda + (\lambda+1) + \log|\mathbb{G}|$ bits, and the evaluation algorithm takes about $n$ RP calls, both remaining the same complexity as those in the prior work. In our DCF scheme, the full-domain evaluation requires $2.5N$ RP calls, in contrast to $4N$ RP calls in the state-of-the-art construction [7]. Its key generation needs about $4n+2$ RP calls, in contrast to $8n$ RP calls in the prior work. The key size is $n \cdot \lambda + (\lambda+1) + (n+1) \cdot \log|\mathbb{G}|$ bits, and the evaluation requires about $2n$ RP calls, without any improvement.

**Parameters:** Domain size $N = 2^n$ for $n \in \mathbb{N}$. Group $\mathbb{G}$. Keyed hash function $\mathsf{H}_S : \mathbb{F}_{2^\lambda} \to \mathbb{F}_{2^\lambda}$. Function $\mathsf{Convert}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$.

$\mathsf{DPF.Gen}(1^\lambda, (\alpha, \beta, n, \mathbb{G}))$:

1: Parse $\alpha = \alpha_1 \dots \alpha_n \in \{0,1\}^n$ and $\beta \in \mathbb{G}$.
2: Sample $\Delta \leftarrow \{0,1\}^\lambda$ such that $\mathsf{lsb}(\Delta) = 1$.
3: Sample $\langle s_0 \,\|\, t_0\rangle_0, \langle s_0 \,\|\, t_0\rangle_1 \leftarrow \{0,1\}^\lambda$ such that $\langle s_0 \,\|\, t_0\rangle_0 \oplus \langle s_0 \,\|\, t_0\rangle_1 = \Delta$.
4: **for** $i \in [1, n-1]$ **do**
5: $\quad \mathsf{CW}_i := \mathsf{H}_S(\langle s_{i-1} \,\|\, t_{i-1}\rangle_0) \oplus \mathsf{H}_S(\langle s_{i-1} \,\|\, t_{i-1}\rangle_1) \oplus \overline{\alpha}_i \cdot \Delta$
6: $\quad \langle s_i \,\|\, t_i\rangle_0 := \mathsf{H}_S(\langle s_{i-1} \,\|\, t_{i-1}\rangle_0) \oplus \alpha_i \cdot \langle s_{i-1} \,\|\, t_{i-1}\rangle_0 \oplus \langle t_{i-1}\rangle_0 \cdot \mathsf{CW}_i$
7: $\quad \langle s_i \,\|\, t_i\rangle_1 := \mathsf{H}_S(\langle s_{i-1} \,\|\, t_{i-1}\rangle_1) \oplus \alpha_i \cdot \langle s_{i-1} \,\|\, t_{i-1}\rangle_1 \oplus \langle t_{i-1}\rangle_1 \cdot \mathsf{CW}_i$
8: $\langle \mathsf{high}^\sigma \,\|\, \mathsf{low}^\sigma\rangle_0 := \mathsf{H}_S(\langle s_{n-1} \,\|\, t_{n-1}\rangle_0 \oplus \sigma)$ for $\sigma \in \{0,1\}$
9: $\langle \mathsf{high}^\sigma \,\|\, \mathsf{low}^\sigma\rangle_1 := \mathsf{H}_S(\langle s_{n-1} \,\|\, t_{n-1}\rangle_1 \oplus \sigma)$ for $\sigma \in \{0,1\}$
10: $\mathsf{HCW} := \langle \mathsf{high}^{\overline{\alpha}_n}\rangle_0 \oplus \langle \mathsf{high}^{\overline{\alpha}_n}\rangle_1$
11: $\mathsf{LCW}^0 := \langle \mathsf{low}^0\rangle_0 \oplus \langle \mathsf{low}^0\rangle_1 \oplus \overline{\alpha}_n, \quad \mathsf{LCW}^1 := \langle \mathsf{low}^1\rangle_0 \oplus \langle \mathsf{low}^1\rangle_1 \oplus \alpha_n$
12: $\mathsf{CW}_n := (\mathsf{HCW} \,\|\, \mathsf{LCW}^0 \,\|\, \mathsf{LCW}^1)$
13: $\langle s_n \,\|\, t_n\rangle_0 := \langle \mathsf{high}^{\alpha_n} \,\|\, \mathsf{low}^{\alpha_n}\rangle_0 \oplus \langle t_{n-1}\rangle_0 \cdot (\mathsf{HCW} \,\|\, \mathsf{LCW}^{\alpha_n})$
14: $\langle s_n \,\|\, t_n\rangle_1 := \langle \mathsf{high}^{\alpha_n} \,\|\, \mathsf{low}^{\alpha_n}\rangle_1 \oplus \langle t_{n-1}\rangle_1 \cdot (\mathsf{HCW} \,\|\, \mathsf{LCW}^{\alpha_n})$
15: $\mathsf{CW}_{n+1} := (\langle t_n\rangle_0 - \langle t_n\rangle_1) \cdot (\mathsf{Convert}_{\mathbb{G}}(\langle s_n\rangle_1) - \mathsf{Convert}_{\mathbb{G}}(\langle s_n\rangle_0) + \beta)$
16: $k_b := (\langle s_0 \,\|\, t_0\rangle_b, \{\mathsf{CW}_i\}_{i\in[1,n+1]})$ for $b \in \{0,1\}$
17: **return** $(k_0, k_1)$

$\mathsf{DPF.Eval}(b, k_b, x)$:

1: Parse $k_b = (\langle s_0^0 \,\|\, t_0^0\rangle_b, \{\mathsf{CW}_i\}_{i\in[1,n+1]})$, $\mathsf{CW}_n = (\mathsf{HCW} \,\|\, \mathsf{LCW}^0 \,\|\, \mathsf{LCW}^1)$, and $x = x_1 \dots x_n \in \{0,1\}^n$.
2: **for** $i \in [1, n-1]$ **do**
3: $\quad \langle s_i^{x_1\dots x_i} \,\|\, t_i^{x_1\dots x_i}\rangle_b := \mathsf{H}_S(\langle s_{i-1}^{x_1\dots x_{i-1}} \,\|\, t_{i-1}^{x_1\dots x_{i-1}}\rangle_b) \oplus x_i \cdot \langle s_{i-1}^{x_1\dots x_{i-1}} \,\|\, t_{i-1}^{x_1\dots x_{i-1}}\rangle_b \oplus \langle t_{i-1}^{x_1\dots x_{i-1}}\rangle_b \cdot \mathsf{CW}_i$
4: $\langle \mathsf{high} \,\|\, \mathsf{low}\rangle_b := \mathsf{H}_S(\langle s_{n-1}^{x_1\dots x_{n-1}} \,\|\, t_{n-1}^{x_1\dots x_{n-1}}\rangle_b \oplus x_n)$
5: $\langle s_n^x \,\|\, t_n^x\rangle_b := \langle \mathsf{high} \,\|\, \mathsf{low}\rangle_b \oplus \langle t_{n-1}^{x_1\dots x_{n-1}}\rangle_b \cdot (\mathsf{HCW} \,\|\, \mathsf{LCW}^{x_n})$
6: **return** $y_b := (-1)^b \cdot (\mathsf{Convert}_{\mathbb{G}}(\langle s_n^x\rangle_b) + \langle t_n^x\rangle_b \cdot \mathsf{CW}_{n+1})$

**Fig. 8.** Our DPF scheme with domain $[0, N)$ and range $\mathbb{G}$.

**Security.** We prove the following theorems in Appendix D.2 and Appendix D.3 of the full version [29]. These theorems turn to the intuition that $\mathsf{CW}_1, \ldots, \mathsf{CW}_n$ are masked by pseudorandom CCR outputs (as the root and the first $n-1$ on-path shared nodes are $\Delta$), and $\mathsf{CW}_{n+1}, \mathsf{VCW}_1, \ldots, \mathsf{VCW}_n$ are masked by some pseudorandom $\mathsf{Convert}_{\mathbb{G}}$ terms taking (pseudo)random CCR outputs as input.

**Parameters:** Domain size $N = 2^n$ for $n \in \mathbb{N}$. Group $\mathbb{G}$. Keyed hash function $\mathsf{H}_S : \mathbb{F}_{2^\lambda} \to \mathbb{F}_{2^\lambda}$. Function $\mathsf{Convert}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$.

$\mathsf{DCF.Gen}(1^\lambda, (\alpha, \beta, n, \mathbb{G}))$:

1: Parse $\alpha = \alpha_1 \ldots \alpha_n \in \{0,1\}^n$ and $\beta \in \mathbb{G}$. Let $\alpha_0 := 0$.
2: Run $(k'_0, k'_1) \leftarrow \mathsf{DPF.Gen}(1^\lambda, (\alpha, -\alpha_n \cdot \beta, n, \mathbb{G}))$ and store its internal variables.
3: **for** $i \in [1, n]$ **do**
4: $\quad \langle v_i \rangle_0 := \mathsf{H}_S(\langle s_{i-1} \,\|\, t_{i-1} \rangle_0 \oplus 2)$
5: $\quad \langle v_i \rangle_1 := \mathsf{H}_S(\langle s_{i-1} \,\|\, t_{i-1} \rangle_1 \oplus 2)$
6: $\quad \mathsf{VCW}_i := (\langle t_{i-1} \rangle_0 - \langle t_{i-1} \rangle_1) \cdot (\mathsf{Convert}_{\mathbb{G}}(\langle v_i \rangle_1) - \mathsf{Convert}_{\mathbb{G}}(\langle v_i \rangle_0) + (\alpha_i - \alpha_{i-1}) \cdot \beta)$
7: $k_b := (k'_b, \{\mathsf{VCW}_i\}_{i \in [1,n]})$ for $b \in \{0,1\}$
8: **return** $(k_0, k_1)$

$\mathsf{DCF.Eval}(b, k_b, x)$:

1: Parse $k_b = (k'_b, \{\mathsf{VCW}_i\}_{i \in [1,n]})$. Let $V_b^0 := 0 \in \mathbb{G}$.
2: Run $y'_b := \mathsf{DPF.Eval}(b, k'_b, x)$ and store its internal variables.
3: **for** $i \in [1, n]$ **do**
4: $\quad \langle v_i^{x_1 \ldots x_{i-1}} \rangle_b := \mathsf{H}_S(\langle s_{i-1}^{x_1 \ldots x_{i-1}} \,\|\, t_{i-1}^{x_1 \ldots x_{i-1}} \rangle_b \oplus 2)$
5: $\quad V_b^i := V_b^{i-1} + (-1)^b \cdot (\mathsf{Convert}_{\mathbb{G}}(\langle v_i^{x_1 \ldots x_{i-1}} \rangle_b) + \langle t_{i-1}^{x_1 \ldots x_{i-1}} \rangle_b \cdot \mathsf{VCW}_i)$
6: **return** $y_b := y'_b + V_b^n$

**Fig. 9.** Our DCF scheme with domain $[0, N)$ and range $\mathbb{G}$.

**Theorem 3.** *Given CCR function* $\mathsf{H} : \mathbb{F}_{2^\lambda} \to \mathbb{F}_{2^\lambda}$, *function* $\mathsf{Convert}_{\mathbb{G}} : \mathbb{F}_{2^{\lambda-1}} \to \mathbb{G}$, *and keyed hash function* $\mathsf{H}_S(x) := \mathsf{H}(S \oplus x)$ *with some key* $S \leftarrow \mathbb{F}_{2^\lambda}$, *Fig. 8 gives a DPF scheme with domain* $[0, N)$ *and range* $\mathbb{G}$.

**Theorem 4.** *Given CCR function* $\mathsf{H} : \mathbb{F}_{2^\lambda} \to \mathbb{F}_{2^\lambda}$, *function* $\mathsf{Convert}_{\mathbb{G}} : \mathbb{F}_{2^\ell} \to \mathbb{G}$ *with* $\ell \in \{\lambda - 1, \lambda\}$, *and keyed hash function* $\mathsf{H}_S(x) := \mathsf{H}(S \oplus x)$ *with some key* $S \leftarrow \mathbb{F}_{2^\lambda}$, *Fig. 9 gives a DCF scheme with domain* $[0, N)$ *and range* $\mathbb{G}$.

### 5.2 DPF Correlation Generation

We define a leveled evaluation algorithm $\mathsf{DPF.NextLevel}$ such that, on input a level index $i \in [1, n]$, all nodes on the $(i-1)$-th level of the share of a shared pseudorandom correlated GGM tree, and the public correction word $\mathsf{CW}_i$ for the $i$-th level, outputs all nodes one the $i$-th level.

**Protocol $\Pi_{\mathsf{DPF}}$**

**Parameters:** Domain size $N = 2^n$ for $n \in \mathbb{N}$. Ring $\mathcal{R}$. Keyed hash function $\mathsf{H}_S : \mathbb{F}_{2^\lambda} \to \mathbb{F}_{2^\lambda}$. Function $\mathsf{Convert}_{\mathcal{R}} : \{0,1\}^* \to \mathcal{R}$. Let $\mathsf{H}' := \mathsf{hb} \circ \mathsf{H}_S$.

**DPF Gen:** This procedure can be executed many times. For each $b \in \{0,1\}$, $P_b$ inputs $(\langle\alpha\rangle_b, \langle\beta\rangle_b^{\mathsf{A}}) \in [0,N) \times \mathcal{R}$ and proceeds as follows:

1. The two parties run sub-protocol $\Pi_{\mathsf{PREP}}$ (Figure 11), which, for each $b \in \{0,1\}$, returns $\langle\Delta\rangle_b$ and $\{(\mathsf{K}_b[\langle\alpha_i\rangle_{1-b}], \mathsf{M}_b[\langle\alpha_i\rangle_b])\}_{i\in[1,n]}$ to $P_b$ such that $\mathsf{lsb}(\langle\Delta\rangle_0 \oplus \langle\Delta\rangle_1) = 1$, and $\mathsf{M}_b[\langle\alpha_i\rangle_b] = \mathsf{K}_{1-b}[\langle\alpha_i\rangle_b] \oplus \langle\alpha_i\rangle_b \cdot \langle\Delta\rangle_{1-b}$ for $i \in [1,n]$.
2. The two parties send $(\mathsf{sample}, \lambda)$ to $\mathcal{F}_{\mathsf{Rand}}$, which returns $W \in \{0,1\}^\lambda$ to them.
3. $P_b$ computes $\langle s_0^0 \,\|\, t_0^0\rangle_b := \langle\Delta\rangle_b \oplus W$. For $i \in [1, n-1]$, $P_b$ sends to $P_{1-b}$
$$\langle \mathsf{CW}_i\rangle_b := (\oplus_{j\in[0,2^{i-1})}\mathsf{H}_S(\langle s_{i-1}^j \,\|\, t_{i-1}^j\rangle_b)) \oplus \overline{\langle\alpha_i\rangle_b} \cdot \langle\Delta\rangle_b \oplus \mathsf{K}_b[\langle\alpha_i\rangle_{1-b}] \oplus \mathsf{M}_b[\langle\alpha_i\rangle_b],$$
receives $\langle \mathsf{CW}_i\rangle_{1-b}$ from $P_{1-b}$, and computes $\mathsf{CW}_i := \langle \mathsf{CW}_i\rangle_b \oplus \langle \mathsf{CW}_i\rangle_{1-b}$ and
$$\{\langle s_i^j \,\|\, t_i^j\rangle_b\}_{j\in[0,2^i)} := \mathsf{DPF.NextLevel}(i, \{\langle s_{i-1}^j \,\|\, t_{i-1}^j\rangle_b\}_{j\in[0,2^{i-1})}, \mathsf{CW}_i).$$
4. $P_b$ samples $\mu_b \leftarrow \{0,1\}^\lambda$, computes
$$\langle \mathsf{Xhigh}^\sigma \,\|\, \mathsf{Xlow}^\sigma\rangle_b := \oplus_{j\in[0,2^{n-1})}\mathsf{H}_S(\langle s_{n-1}^j \,\|\, t_{n-1}^j\rangle_b \oplus \sigma) \quad \text{for } \sigma \in \{0,1\},$$
$$d_b := \mathsf{H}'(\mu_b \oplus \mathsf{K}_b[\langle\alpha_n\rangle_{1-b}]) \oplus \mathsf{H}'(\mu_b \oplus \mathsf{K}_b[\langle\alpha_n\rangle_{1-b}] \oplus \langle\Delta\rangle_b) \oplus \langle \mathsf{Xhigh}^0 \oplus \mathsf{Xhigh}^1\rangle_b,$$
sends $(\mu_b, d_b)$ to $P_{1-b}$, and receives $(\mu_{1-b}, d_{1-b})$ from $P_{1-b}$. Then, $P_b$ computes
$$\langle \mathsf{HCW}\rangle_b := \langle \mathsf{Xhigh}^{\overline{\langle\alpha_n\rangle_b}}\rangle_b \oplus \mathsf{H}'(\mu_b \oplus \mathsf{K}_b[\langle\alpha_n\rangle_{1-b}]) \oplus \mathsf{H}'(\mu_{1-b} \oplus \mathsf{M}_b[\langle\alpha_n\rangle_b]) \oplus \langle\alpha_n\rangle_b \cdot d_{1-b},$$
$$\langle \mathsf{LCW}^0\rangle_b := \langle \mathsf{Xlow}^0\rangle_b \oplus \langle\alpha_n\rangle_b \oplus b, \qquad \langle \mathsf{LCW}^1\rangle_b := \langle \mathsf{Xlow}^1\rangle_b \oplus \langle\alpha_n\rangle_b,$$
sends $\langle \mathsf{CW}_n\rangle_b := (\langle \mathsf{HCW}\rangle_b \,\|\, \langle \mathsf{LCW}^0\rangle_b \,\|\, \langle \mathsf{LCW}^1\rangle_b)$ to $P_{1-b}$, receives $\langle \mathsf{CW}_n\rangle_{1-b}$ from $P_{1-b}$, and computes $\mathsf{CW}_n := \langle \mathsf{CW}_n\rangle_b \oplus \langle \mathsf{CW}_n\rangle_{1-b}$ and
$$\{\langle s_n^j \,\|\, t_n^j\rangle_b\}_{j\in[0,N)} := \mathsf{DPF.NextLevel}(n, \{\langle s_{n-1}^j \,\|\, t_{n-1}^j\rangle_b\}_{j\in[0,2^{n-1})}, \mathsf{CW}_n).$$
5. **(Binary field $\mathcal{R} = \mathbb{F}_{2^\ell}$, without $\mathcal{F}_{\mathsf{OLE}}$)**
$P_b$ computes $\langle \mathsf{CW}_{n+1}\rangle_b^{\mathsf{A}} := (\sum_{j\in[0,N)} \mathsf{Convert}_{\mathcal{R}}(\langle s_n^j\rangle_b)) + \langle\beta\rangle_b^{\mathsf{A}}$.
**(General ring $\mathcal{R}$, using $\mathcal{F}_{\mathsf{OLE}}$)**
The two parties run sub-protocol $\Pi_{\mathsf{MULT}}$ (Figure 12), which, for each $b \in \{0,1\}$, takes as input
$$\langle A\rangle_b^{\mathsf{A}} := (-1)^b \cdot \textstyle\sum_{j\in[0,N)} \langle t_n^j\rangle_b \in \mathcal{R},$$
$$\langle B\rangle_b^{\mathsf{A}} := (-1)^{1-b} \cdot \textstyle\sum_{j\in[0,N)} \mathsf{Convert}_{\mathcal{R}}(\langle s_n^j\rangle_b) + \langle\beta\rangle_b^{\mathsf{A}} \in \mathcal{R},$$
and returns $\langle \mathsf{CW}_{n+1}\rangle_b^{\mathsf{A}}$ to $P_b$.
In either case, $P_b$ sends $\langle \mathsf{CW}_{n+1}\rangle_b^{\mathsf{A}}$ to $P_{1-b}$, receives $\langle \mathsf{CW}_{n+1}\rangle_{1-b}^{\mathsf{A}}$ from $P_{1-b}$, and computes $\mathsf{CW}_{n+1} := \langle \mathsf{CW}_{n+1}\rangle_b^{\mathsf{A}} + \langle \mathsf{CW}_{n+1}\rangle_{1-b}^{\mathsf{A}}$.
6. $P_b$ computes $k_b := (\langle\Delta\rangle_b \oplus W, \{\mathsf{CW}_i\}_{i\in[1,n+1]})$ and $\langle \mathbf{r}^{(j)}\rangle_b^{\mathsf{A}} := \mathsf{DPF.Eval}(b, k_b, j)$ for $j \in [0,N)$, and outputs $\langle \mathbf{r}\rangle_b^{\mathsf{A}} \in \mathcal{R}^N$.

**Fig. 10.** DPF correlation generation in the $(\mathcal{F}_{\mathsf{COT}}, \mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{OLE}})$-hybrid model.

In Fig. 10, we present our DPF correlation generation protocol $\Pi_{\mathsf{DPF}}$. This protocol operates in the $(\mathcal{F}_{\mathsf{COT}}, \mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{OLE}})$-hybrid model. $\mathcal{F}_{\mathsf{Rand}}$ is the standard coin-tossing functionality that outputs a uniform string to both parties. $\mathcal{F}_{\mathsf{OLE}}$ is the functionality for oblivious linear evaluation (OLE) on ring $\mathcal{R}$, where $P_0$ (resp., $P_1$) is given *random* $(\mathbf{x}_0, \mathbf{z}_0) \in \mathcal{R}^N \times \mathcal{R}^N$ (resp., $(\mathbf{x}_1, \mathbf{z}_1) \in \mathcal{R}^N \times \mathcal{R}^N$) such that $\mathbf{z}_0 + \mathbf{z}_1$ equals the component-wise multiplication $\mathbf{x}_0 \odot \mathbf{x}_1$. We refer readers to Appendix F.2 and Appendix F.3 of the full version [29] for the definitions and instantiations of $\mathcal{F}_{\mathsf{Rand}}$ and $\mathcal{F}_{\mathsf{OLE}}$. If $\beta$ is a bit-string, $\Pi_{\mathsf{DPF}}$ never uses $\mathcal{F}_{\mathsf{OLE}}$.

$\Pi_{\mathsf{DPF}}$ requires $\mathcal{F}_{\mathsf{Rand}}$ for the following reason. Note that $\Pi_{\mathsf{DPF}}$ uses the same global offset $\Delta$ as the roots of polynomially many shared trees, each of which defines a fresh DPF correlation. So, the two shares of this identical root should be "re-randomized" to avoid the identical per-party shares of the defined correlations. The two parties do this re-randomization by calling $\mathcal{F}_{\mathsf{Rand}}$ for a public randomness $W$ and XORing this value to their shares of $\Delta$, respectively.

**Protocol $\Pi_{\mathsf{PREP}}$**

**Initialize:** This procedure is executed only once for each $b \in \{0,1\}$. The two parties send $(\mathsf{init})$ to $\mathcal{F}_{\mathsf{COT}}^b$ with identifier $b$, which returns $\Delta'_b \in \{0,1\}^\lambda$ to $P_b$. $P_b$ sends $\mathsf{lsb}(\Delta'_b)$ to $P_{1-b}$, receives $\mathsf{lsb}(\Delta'_{1-b})$ from $P_{1-b}$, and sets $\langle \Delta \rangle_b := \Delta'_b \oplus (0^{\lambda-1} \,\|\, (\mathsf{lsb}(\Delta'_{1-b}) \oplus b))$ such that $\mathsf{lsb}(\langle \Delta \rangle_0 \oplus \langle \Delta \rangle_1) = 1$.

For each $b \in \{0,1\}$: $P_b$ inputs $\langle \alpha \rangle_b \in \{0,1\}^n$ and proceeds as follows.

1-1. The two parties send $(\mathsf{extend}, n)$ to $\mathcal{F}_{\mathsf{COT}}^b$ with identifier $b$, which returns $\mathbf{k}_b \in \mathbb{F}_{2^\lambda}^n$ to $P_b$ and $(\mathbf{r}_{1-b}, \mathbf{m}_{1-b}) \in \mathbb{F}_2^n \times \mathbb{F}_{2^\lambda}^n$ to $P_{1-b}$ such that $\mathbf{m}_{1-b} = \mathbf{k}_b \oplus \mathbf{r}_{1-b} \cdot \Delta'_b$.

1-2. $P_b$ sets $\mathbf{g}_b := \langle \alpha \rangle_b \oplus \mathbf{r}_b$, sends $\mathbf{g}_b$ to $P_{1-b}$, and receives $\mathbf{g}_{1-b}$ from $P_{1-b}$. For $i \in [1,n]$, $P_b$ sets

$$\mathsf{K}_b[\langle \alpha_i \rangle_{1-b}] := \mathbf{k}_b^{(i)} \oplus \mathbf{g}_{1-b}^{(i)} \cdot \langle \Delta \rangle_b,$$
$$\mathsf{M}_b[\langle \alpha_i \rangle_b] := \mathbf{m}_b^{(i)} \oplus \mathbf{r}_b^{(i)} \cdot (0^{\lambda-1} \,\|\, (\mathsf{lsb}(\Delta'_b) \oplus (1-b))).$$

1-3. $P_b$ outputs $\langle \Delta \rangle_b$ and $\{(\mathsf{K}_b[\langle \alpha_i \rangle_{1-b}], \mathsf{M}_b[\langle \alpha_i \rangle_b])\}_{i \in [1,n]}$.

**Fig. 11.** Preprocessing sub-protocol for DPF/DCF correlation generation.

**Protocol $\Pi_{\mathsf{MULT}}$**

For each $b \in \{0,1\}$: $P_b$ inputs $(\langle A \rangle_b^{\mathsf{A}}, \langle B \rangle_b^{\mathsf{A}}) \in \mathcal{R}^2$ and proceeds as follows.

1. The two parties send $(\mathsf{extend}, 2)$ to $\mathcal{F}_{\mathsf{OLE}}$, which, for each $b \in \{0,1\}$, returns $(\mathbf{x}_b, \mathbf{z}_b) \in \mathcal{R}^2 \times \mathcal{R}^2$ to $P_b$ such that $\mathbf{z}_0 + \mathbf{z}_1 = \mathbf{x}_0 \cdot \mathbf{x}_1$.
2. $P_b$ computes $(\gamma_b, \zeta_b) := (\langle A \rangle_b^{\mathsf{A}}, \langle B \rangle_b^{\mathsf{A}}) + (\mathbf{x}_b^{(b)}, \mathbf{x}_b^{(1-b)})$, sends $(\gamma_b, \zeta_b)$ to $P_{1-b}$, and receives $(\gamma_{1-b}, \zeta_{1-b})$ from $P_{1-b}$.
3. $P_b$ outputs $\langle A \cdot B \rangle_b^{\mathsf{A}} := \langle A \rangle_b^{\mathsf{A}} \cdot \langle B \rangle_b^{\mathsf{A}} + \langle A \rangle_b^{\mathsf{A}} \cdot \zeta_{1-b} - \mathbf{x}_b^{(1-b)} \cdot \gamma_{1-b} + \mathbf{z}_b^{(0)} + \mathbf{z}_b^{(1)}$.

**Fig. 12.** OLE-based multiplication sub-protocol.

In $\Pi_{\mathsf{DPF}}$, the key $S$ of the keyed hash function $\mathsf{H}_S$ can be produced by one $\mathcal{F}_{\mathsf{Rand}}$ invocation before protocol execution, and we omit this setup for simplicity.

**Security.** We prove Theorem 5 in Appendix D.4 of the full version [29]. This proof will consider polynomially many concurrent **Gen** executions that uses the one-time initialized $\Delta$. Intuitively, the security primarily follows from the COT-based secure computation of correction words, where the COT tuples are related to the global offset $\Delta$ so that the transcripts are masked by CCR responses. In particular, the intermediate transcript $d_b$ is masked by a CCR response coming from a legal CCR query with overwhelming probability due to the uniform $\mu_b$.

**Theorem 5.** *Given CCR function* $\mathsf{H} : \mathbb{F}_{2^\lambda} \to \mathbb{F}_{2^\lambda}$*, function* $\mathsf{Convert}_{\mathcal{R}} : \mathbb{F}_{2^{\lambda-1}} \to \mathcal{R}$*, and keyed hash function* $\mathsf{H}_S(x) := \mathsf{H}(S \oplus x)$ *with some key* $S \leftarrow \mathbb{F}_{2^\lambda}$*, protocol* $\Pi_{\mathsf{DPF}}$ *(Fig. 10) UC-realizes functionality* $\mathcal{F}_{\mathsf{DPF}}$ *(Fig. 7) against any semi-honest adversary in the* $(\mathcal{F}_{\mathsf{COT}}, \mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{OLE}})$*-hybrid model. If* $\mathcal{R} = \mathbb{F}_{2^\ell}$ *for* $\ell \in \mathbb{N}$*, protocol* $\Pi_{\mathsf{DPF}}$ *never invokes* $\mathcal{F}_{\mathsf{OLE}}$*.*

**Table 3.** The efficiency of distributed correlation generation for our DPF scheme. All numbers are in milliseconds (*ms*).

| | | $n = 20$ | $n = 22$ | $n = 24$ | $n = 26$ | $n = 28$ |
|---|---|---|---|---|---|---|
| $\mathcal{R} = \mathbb{F}_{2^{127}}$ | LAN | 50 | 120 | 397 | 1501 | 5920 |
| | WAN | 2752 | 3020 | 3492 | 4786 | 9355 |
| $\mathcal{R} = \mathbb{F}_2$ | LAN | 29 | 30 | 34 | 52 | 120 |
| | WAN | 2930 | 3132 | 3337 | 3554 | 3823 |

**Communication Optimization.** $\Pi_{\mathsf{DPF}}$ has the following two optimizations:

- For $t$ concurrent **Gen** executions (e.g., in its applications to RAM-based computation [22], FSS-based MPC [7], and OLE extension [12], etc.), each $P_b$ can compress all $\mu_b$'s in these executions via a PRF $F : \mathbb{F}_{2^\lambda} \times \{0,1\}^* \to \mathbb{F}_{2^\lambda}$ with a fresh key $k_{\mathsf{prf},b} \leftarrow \mathbb{F}_{2^\lambda}$ sampled *after receiving its COT outputs (from both* $\mathcal{F}^b_{\mathsf{COT}}$ *and* $\mathcal{F}^{1-b}_{\mathsf{COT}}$*) in all executions*. For each execution with sub-session ID $\mathsf{ssid}$, the two parties define $\mu_b := F(k_{\mathsf{prf},b}, \mathsf{ssid})$.
- All invocations of $\mathcal{F}_{\mathsf{Rand}}$ can be compressed via another independent PRF key sampled *after the one-time initialization of* $\mathcal{F}^b_{\mathsf{COT}}$ *and* $\mathcal{F}^{1-b}_{\mathsf{COT}}$ so that the root of each $P_b$'s tree is (pseudo)random.
- Another method to save the communication for random $\mu_b$'s is to replace $\mathsf{H}_S$ by a hash function that meets "CCR for naturally derived keys" [28,47], which can also be implemented in one RP call. Note that $\mu_b$ is introduced to prevent the replay attack, which results from the manipulation of COT outputs, against the hashing mask in $d_b$. The alternative hash function addresses this attack by adding non-repeating tweaks.

**Complexity Analysis (Binary Field).** Consider the complexity per execution when the first PRF-based optimization is used in $t$ concurrent **Gen** executions. The cost is symmetric. $\Pi_{\mathsf{DPF}}$ uses $n$ COT tuples per party and one $\mathcal{F}_{\mathsf{Rand}}$ call. Each party sends $(n+1)+(n+1)\cdot\lambda+\frac{\lambda}{t}+\log|\mathcal{R}|$ bits. The computation per party is dominated by the tree expansion in $n$ $\mathsf{DPF.NextLevel}$ calls, or $1.5N$ RP calls. $\Pi_{\mathsf{DPF}}$ runs in $n+3$ rounds (without counting the one-time setup).

In contrast, the binary-field protocol [22] can be implemented from GMW-style 2PC and $n$ string OTs each with $(\lambda-1)$-bit payloads. One can cast these string OTs into $n$ precomputed COT tuples according to [3,34]. Using these tuples, each party sends $n+n\cdot(3\lambda-1)+\log|\mathcal{R}|$ bits, and the computation per party is dominated by the $2N$ RP calls in GGM tree expansion. This protocol can proceed in $2n+2$ rounds: one for sending $n$ masked choice bits, two for sharing and revealing each of the first $n$ correction words, and one for revealing the $(n+1)$-th correction word. Our savings in computation, communication, and round complexity are about 25%, 66.6%, and 50%, respectively.

We implement $\Pi_{\mathsf{PREP}}$ and $\Pi_{\mathsf{DPF}}$ in `C++`, and perform benchmarks on a pair of Amazon EC2 `R5.xlarge` instances. We take binary fields $\mathcal{R}=\mathbb{F}_{2^{127}}$ and $\mathcal{R}=\mathbb{F}_2$ under computational security parameter $\lambda\approx 128$. The reported time include both distributed key generation and full-domain evaluation. We set 1Gbps bandwidth with no latency as our LAN setting, and 20Mbps bandwidth with $100ms$ latency as our WAN setting. The results are shown in Table 3. We can see that our protocol is practically efficient, especially for two-server PIR. Although all numbers are reported based on one thread, performing one correlation generation for $2^{28}$ 127-bit values takes about 6 s, which is about 30% to 40% faster than the performance from a prior implementation in the same threads [22].

**Complexity Analysis (General Ring).** The two parties additionally need two precomputed OLE tuples for the secure multiplication. Overall, each party sends $(n+1)+(n+1)\cdot\lambda+\frac{\lambda}{t}+3\cdot\log|\mathcal{R}|$ bits, and the protocol runs in $n+4$ rounds.

In contrast, the binary-field protocol [22] can be adapted for the general-ring $\mathsf{CW}_{n+1}$ in the DPF scheme [13]. Securely computing this $\mathsf{CW}_{n+1}$ consumes two OLE tuples and needs the level-by-level 2PC, which leads to two additional bits in each OT payload per level, to share the last-level control bit $\langle t_n\rangle_1$. Each party sends at most $n+n\cdot(3\lambda+3)+3\cdot\log|\mathcal{R}|$ bits, and the protocol runs in $2n+3$ rounds. The improvement is the same as the binary-field case.

### 5.3 DCF Correlation Generation

Our DCF protocol $\Pi_{\mathsf{DCF}}$ in Fig. 13 extends $\Pi_{\mathsf{DPF}}$ by also computing $n$ value correction words and defining the evaluation result as per our DCF scheme. If $\beta$ is a bit-string, the two parties can compute $n$ value correction words without using precomputed OLE tuples. Otherwise, for a general ring element $\beta$, these correction words are obtained from OLE-based secure multiplication.

**Security.** We prove Theorem 6 in Appendix D.5 of the full version [29], where polynomially many concurrent **Gen** executions are considered. The security is also based on the COT- and OLE-based secure computation of the $n$ additional correction words of our DCF scheme. Note that the intermediate $y_b^i$'s are pseudorandom due the masking CCR responses, which are for the legal CCR queries with overwhelming probability in the presence of uniform $x_b^i$'s.

**Theorem 6.** *Given CCR function* $\mathsf{H} : \mathbb{F}_{2^\lambda} \to \mathbb{F}_{2^\lambda}$, *function* $\mathsf{Convert}_{\mathcal{R}} : \mathbb{F}_{2^\ell} \to \mathcal{R}$ *for* $\ell \in \{\lambda - 1, \lambda\}$, *and keyed hash function* $\mathsf{H}_S(x) := \mathsf{H}(S \oplus x)$ *with some key* $S \leftarrow \mathbb{F}_{2^\lambda}$, *protocol* $\Pi_{\mathsf{DCF}}$ *(Fig. 13) UC-realizes functionality* $\mathcal{F}_{\mathsf{DCF}}$ *(Fig. 7) against any semi-honest adversary in the* $(\mathcal{F}_{\mathsf{COT}}, \mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{OLE}})$*-hybrid model. If* $\mathcal{R} = \mathbb{F}_{2^\ell}$ *for* $\ell \in \mathbb{N}$, *protocol* $\Pi_{\mathsf{DCF}}$ *never invokes* $\mathcal{F}_{\mathsf{OLE}}$.

**Protocol $\Pi_{\mathsf{DCF}}$**

**Parameters:** Domain size $N = 2^n$ for $n \in \mathbb{N}$. Ring $\mathcal{R}$. Keyed hash function $\mathsf{H}_S : \mathbb{F}_{2^\lambda} \to \mathbb{F}_{2^\lambda}$. Function $\mathsf{Convert}_{\mathcal{R}} : \{0,1\}^* \to \mathcal{R}$. Let $\mathsf{H}^* := \mathsf{Convert}_{\mathcal{R}} \circ \mathsf{H}_S$.

**DCF Gen:** This procedure can be executed many times. For each $b \in \{0,1\}$, $P_b$ inputs $(\langle\alpha\rangle_b, \langle\beta\rangle_b^{\mathsf{A}}) \in [0, N) \times \mathcal{R}$ and proceeds as in $\Pi_{\mathsf{DPF}}$ (Figure 8), with the same Step 1, 2 and the following modifications to the subsequent steps:

3. Along with $\langle\mathsf{CW}_i\rangle_b$ for $i \in [1, n-1]$, $P_b$ samples $x_b^i \leftarrow \{0,1\}^\lambda$, computes
$$y_b^i := \mathsf{H}^*(x_b^i \oplus \mathsf{K}_b[\langle\alpha_i\rangle_{1-b}]) - \mathsf{H}^*(x_b^i \oplus \mathsf{K}_b[\langle\alpha_i\rangle_{1-b}] \oplus \langle\Delta\rangle_b) + \langle\beta\rangle_b^{\mathsf{A}} - 2 \cdot \langle\alpha_i\rangle_b \cdot \langle\beta\rangle_b^{\mathsf{A}},$$
sends $(x_b^i, y_b^i)$ to $P_{1-b}$, receive $(x_{1-b}^i, y_{1-b}^i)$ from $P_{1-b}$, and computes
$$\langle\alpha_i \cdot \beta\rangle_b^{\mathsf{A}} := \langle\alpha_i\rangle_b \cdot \langle\beta\rangle_b^{\mathsf{A}} - \mathsf{H}^*(x_b^i \oplus \mathsf{K}_b[\langle\alpha_i\rangle_{1-b}]) + \mathsf{H}^*(x_{1-b}^i \oplus \mathsf{M}_b[\langle\alpha_i\rangle_b]) + \langle\alpha_i\rangle_b \cdot y_{1-b}^i.$$
4. Along with $\langle\mathsf{CW}_n\rangle_b$, $P_b$ repeats Step 3 for $i = n$ and computes $\langle\alpha_n \cdot \beta\rangle_b^{\mathsf{A}}$.
5. For $i \in [1, n]$ and $j \in [0, 2^{i-1})$, $P_b$ computes $\langle v_i^j\rangle_b := \mathsf{H}_S(\langle s_{i-1}^j \,\|\, t_{i-1}^j\rangle_b \oplus 2)$ and $\langle\alpha_0 \cdot \beta\rangle_b^{\mathsf{A}} := 0$. $P_b$ computes $\langle\mathsf{CW}_{n+1}\rangle_b^{\mathsf{A}}$ by using $\langle\alpha_n \cdot \beta\rangle_b^{\mathsf{A}}$ instead of $\langle\beta\rangle_b^{\mathsf{A}}$, and:
**(Binary field $\mathcal{R} = \mathbb{F}_{2^\ell}$, without $\mathcal{F}_{\mathsf{OLE}}$)** For $i \in [1, n]$ in parallel:
$P_b$ computes $\langle\mathsf{VCW}_i\rangle_b^{\mathsf{A}} := (\sum_{j \in [0, 2^{i-1})} \mathsf{Convert}_{\mathcal{R}}(\langle v_i^j\rangle_b)) + \langle\alpha_i \cdot \beta\rangle_b^{\mathsf{A}} - \langle\alpha_{i-1} \cdot \beta\rangle_b^{\mathsf{A}}$.
**(General ring $\mathcal{R}$, using $\mathcal{F}_{\mathsf{OLE}}$)** For $i \in [1, n]$ in parallel:
The two parties run sub-protocol $\Pi_{\mathsf{MULT}}$ (Figure 12), which, for each $b \in \{0,1\}$, takes as input
$$\langle A_i\rangle_b^{\mathsf{A}} := (-1)^b \cdot \textstyle\sum_{j \in [0, 2^{i-1})} \langle t_{i-1}^j\rangle_b \in \mathcal{R},$$
$$\langle B_i\rangle_b^{\mathsf{A}} := (-1)^{1-b} \cdot \textstyle\sum_{j \in [0, 2^{i-1})} \mathsf{Convert}_{\mathcal{R}}(\langle v_i^j\rangle_b) + \langle\alpha_i \cdot \beta\rangle_b^{\mathsf{A}} - \langle\alpha_{i-1} \cdot \beta\rangle_b^{\mathsf{A}} \in \mathcal{R},$$
and returns $\langle\mathsf{VCW}_i\rangle_b^{\mathsf{A}}$ to $P_b$.
In either case, along with $\langle\mathsf{CW}_{n+1}\rangle_b^{\mathsf{A}}$, $P_b$ sends $\langle\mathsf{VCW}_i\rangle_b^{\mathsf{A}}$ to $P_{1-b}$, receives $\langle\mathsf{VCW}_i\rangle_{1-b}^{\mathsf{A}}$ from $P_{1-b}$, and computes $\mathsf{VCW}_i := \langle\mathsf{VCW}_i\rangle_b^{\mathsf{A}} + \langle\mathsf{VCW}_i\rangle_{1-b}^{\mathsf{A}}$.
6. $P_b$ computes $k_b := (\langle\Delta\rangle_b \oplus W, \{\mathsf{CW}_i\}_{i \in [1, n+1]}, \{\mathsf{VCW}_i\}_{i \in [1, n]})$ and $\langle\mathbf{r}^{(j)}\rangle_b^{\mathsf{A}} := \mathsf{DCF.Eval}(b, k_b, j)$ for $j \in [0, N)$, and outputs $\langle\mathbf{r}\rangle_b^{\mathsf{A}} \in \mathcal{R}^N$.

**Fig. 13.** DCF correlation generation in the $(\mathcal{F}_{\mathsf{COT}}, \mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{OLE}})$-hybrid model.

**Communication Optimization.** The optimizations in Sect. 5.2 also applies to the DCF protocol $\Pi_{\mathsf{DCF}}$. Moreover, the random elements $\{x_b^i\}_{i\in[1,n]}$ in $\Pi_{\mathsf{DCF}}$ can also be compressed using the same technique for the random $\mu_b$'s.

**Complexity Analysis (Binary Field).** Consider the complexity per execution when the first PRF-based optimization is used in $t$ concurrent **Gen** executions. The cost is symmetric. $\Pi_{\mathsf{DCF}}$ consumes $n$ COT tuples per party and one $\mathcal{F}_{\mathsf{Rand}}$ call. Each party sends $(n+1)+(n+1)\cdot\lambda+\frac{\lambda}{t}+(2n+1)\cdot\log|\mathcal{R}|$ bits, and the computation per party comes from the $2.5N$ RP calls in the tree expansion. $\Pi_{\mathsf{DCF}}$ has round complexity $n+3$, the same as $\Pi_{\mathsf{DPF}}$ in the binary-field case.

In contrast, the state-of-the-art protocol of [7] requires $n$ string OTs to run GMW-style 2PC. The string OTs consume $n$ precomputed COT tuples and have payloads of $(\lambda-1)+2\cdot\log|\mathcal{R}|$ bits. Using $n$ COT tuples, each party sends $n+n\cdot(3\lambda-1+5\cdot\log|\mathcal{R}|)+\log|\mathcal{R}|$ bits, and the computation per party is dominated by the $4N$ RP calls in GGM tree expansion in $2n+2$ rounds. Our savings in computation and round complexity are 37.5% and 50%, respectively. For a typical ring $\mathcal{R}$ with size $|\mathcal{R}|\approx 2^\lambda$, the communication reduction is about 62.5%. When $\mathcal{R}$ is sufficiently small, this reduction can be 66.6%.

**Complexity Analysis (General Ring).** $\Pi_{\mathsf{DCF}}$ also works for general $\mathcal{R}$ at the cost of additionally using $2n+2$ precomputed OLE tuples. This general-ring version proceeds in $n+4$ rounds, and the overall outgoing communication per party is $(n+1)+(n+1)\cdot\lambda+\frac{\lambda}{t}+(4n+3)\cdot\log|\mathcal{R}|$ bits.

In contrast, the OT-based protocol [7] can run in $2n+3$ rounds. Each party sends at most $n+n\cdot(3\lambda+3+4\cdot\log|\mathcal{R}|)+(3n+3)\cdot\log|\mathcal{R}|$ bits and uses $2n+2$ OLE tuples. Our savings in communication and round complexity are about $50\%\sim 66.6\%$ and 50%, respectively, for typical ring size $|\mathcal{R}|\le 2^\lambda$.

**Acknowledgements.** Work of Kang Yang is supported by the National Key Research and Development Program of China (Grant No. 2022YFB2702000), and by the National Natural Science Foundation of China (Grant Nos. 62102037, 61932019). Work of Xiao Wang is supported by DARPA under Contract No. HR001120C0087, NSF award #2016240, #2236819, and research awards from Meta and Google. 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. Work of Jiang Zhang is supported by the National Key Research and Development Program of China (Grant No. 2022YFB2702000), and by the National Natural Science Foundation of China (Grant Nos. 62022018, 61932019). Work of Zheli Liu is supported by the National Natural Science Foundation of China (Grant No. 62032012).

## References


1. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013, pp. 535–548. ACM Press (Nov 2013). https://doi.org/10.1145/2508859.2516738
2. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac′n′Cheese: zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12828, pp. 92–122. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84259-8_4
3. Beaver, D.: Precomputing oblivious transfer. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 97–109. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-44750-4_8
4. Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient Garbling from a Fixed-Key Blockcipher. In: 2013 IEEE Symposium on Security and Privacy, pp. 478–492. IEEE Computer Society Press (May 2013). https://doi.org/10.1109/SP.2013.39
5. Bhattacharya, S., Nandi, M.: Full indifferentiable security of the Xor of two or more random permutations using the $\chi^2$ method. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 387–412. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_15
6. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Lightweight Techniques for Private Heavy Hitters. In: 2021 IEEE Symposium on Security and Privacy, pp. 762–776. IEEE Computer Society Press (May 2021). https://doi.org/10.1109/SP40001.2021.00048
7. Boyle, E., et al.: Function secret sharing for mixed-mode and fixed-point secure computation. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12697, pp. 871–900. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77886-6_30
8. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing Vector OLE. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 896–912. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243868
9. Boyle, E., et al.: Correlated Pseudorandomness from Expand-Accumulate Codes. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 603–633. Springer, Heidelberg (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4_21
10. Boyle, E., et al.: Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 291–308. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3354255
11. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: silent OT extension and more. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 489–518. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_16
12. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators from ring-LPN. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12171, pp. 387–416. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_14

13. Boyle, E., Gilboa, N., Ishai, Y.: Function Secret Sharing: Improvements and Extensions. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 1292–1303. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978429
14. Boyle, E., Gilboa, N., Ishai, Y.: Secure computation with preprocessing via function secret sharing. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11891, pp. 341–371. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_14
15. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
16. Chen, S., Steinberger, J.: Tight security bounds for key-alternating ciphers. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 327–350. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_19
17. Choi, S.G., Katz, J., Kumaresan, R., Zhou, H.-S.: On the security of the Free-XOR technique. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 39–53. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28914-9_3
18. Couteau, G., Rindal, P., Raghuraman, S.: Silver: silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12827, pp. 502–534. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84252-9_17
19. Damgård, I., Nielsen, J.B., Nielsen, M., Ranellucci, S.: The tinytable protocol for 2-party secure computation, or: gate-scrambling revisited. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 167–187. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_6
20. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_38
21. Dittmer, S., Ishai, Y., Ostrovsky, R.: Line-Point Zero Knowledge and Its Applications. In: 2nd Conference on Information-Theoretic Cryptography (2021). https://doi.org/10.4230/LIPIcs.ITC.2021.5
22. Doerner, J., shelat, a.: Scaling ORAM for Secure Computation. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 523–535. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3133967
23. Garimella, G., Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Oblivious Key-Value Stores and Amplification for Private Set Intersection. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 395–425. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84245-1_14
24. Ghosh, S., Nielsen, J.B., Nilges, T.: Maliciously secure oblivious linear function evaluation with constant overhead. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 629–659. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_22
25. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_35
26. Goldreich, O., Goldwasser, S., Micali, S.: How to Construct Random Functions (Extended Abstract). In: 25th FOCS. pp. 464–479. IEEE Computer Society Press (Oct 1984). https://doi.org/10.1109/SFCS.1984.715949

27. Goldreich, O., Micali, S., Wigderson, A.: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987). https://doi.org/10.1145/28395.28420
28. Guo, C., Katz, J., Wang, X., Yu, Y.: Efficient and Secure Multiparty Computation from Fixed-Key Block Ciphers. In: 2020 IEEE Symposium on Security and Privacy, pp. 825–841. IEEE Computer Society Press (May 2020). https://doi.org/10.1109/SP40000.2020.00016
29. Guo, X., et al.: Half-Tree: Halving the Cost of Tree Expansion in COT and DPF. Cryptology ePrint Archive, Report 2022/1431 (2022), https://eprint.iacr.org/2022/1431
30. Gupta, K., Kumaraswamy, D., Chandran, N., Gupta, D.: LLAMA: A Low Latency Math Library for Secure Inference. Privacy Enhancing Technologies Symposium (PETS 2022) (2022). 10.56553/popets-2022-0109
31. Efficient Secure Two-Party Protocols. ISC, Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14303-8
32. Hazay, C., Scholl, P., Soria-Vazquez, E.: Low cost constant round MPC combining BMR and oblivious transfer. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 598–628. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_21
33. Heath, D., Kolesnikov, V.: One Hot Garbling. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021, pp. 574–593. ACM Press (Nov 2021). https://doi.org/10.1145/3460120.3484764
34. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_9
35. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 830–842. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978357
36. Keller, M., Pastro, V., Rotaru, D.: Overdrive: making SPDZ great again. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 158–189. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_6
37. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free xor gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_40
38. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_40
39. Patarin, J.: The coefficients H technique. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 328–345. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04159-4_21
40. Rindal, P., Schoppmann, P.: VOLE-PSI: fast OPRF and circuit-PSI from vector-OLE. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12697, pp. 901–930. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77886-6_31
41. Rogaway, P., Steinberger, J.: Constructing cryptographic hash functions from fixed-key blockciphers. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 433–450. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_24

42. Schoppmann, P., Gascón, A., Reichert, L., Raykova, M.: Distributed Vector-OLE: Improved Constructions and Implementation. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 1055–1072. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3363228
43. Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits. In: 2021 IEEE Symposium on Security and Privacy. pp. 1074–1091. IEEE Computer Society Press (May 2021). https://doi.org/10.1109/SP40001.2021.00056
44. Weng, C., Yang, K., Xie, X., Katz, J., Wang, X.: Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning. In: Bailey, M., Greenstadt, R. (eds.) USENIX Security 2021, pp. 501–518. USENIX Association (Aug 2021)
45. Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021, pp. 2986–3001. ACM Press (Nov 2021). https://doi.org/10.1145/3460120.3484556
46. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast Extension for Correlated OT with Small Communication. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020, pp. 1607–1626. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3417276
47. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8