Batched Function Secret Sharing: Parallel Generation and Key Compression

Tianpei Lu[1] Keyu Ji[1] Yuntian Chen[2] Yuan Zhao[3]
Shoumeng Yan[3] Bingsheng Zhang[1] Kui Ren[1]

[1]The State Key Laboratory of Blockchain and Data Security, Zhejiang University
[2]Northwest University [3]Ant Group

{lutianpei,jikeyu,bingsheng,kuiren}@zju.edu.cn
chenyt_x@163.com caesar.zy@antgroup.com Shoumeng.ysm@antgroup.com

Abstract

Function secret sharing (FSS) enables local evaluation of secret-shared functions and is an important building block for efficient secure computation, particularly for comparisons and private lookups. However, conventional tree-based distributed point functions (DPFs) and distributed comparison functions (DCFs) incur sequential dependencies during key generation and require $O(n\lambda)$-bit keys for fixed output length, where n is the input length and λ is the security parameter. We address these bottlenecks through parallel correction-word generation and compact key representation. By sampling the states along the target path in advance, we decouple correction computations across tree levels and enable their parallel generation. For compression, we give a shared-header batch representation under an explicit label-evaluation interface, using homomorphic seed expansion to reconstruct encrypted correction words locally. A uniform sublinear header bound makes the amortized key size approach $n+m$ bits per evaluator per function, where m is the payload length. For 20-bit inputs, parallelization reduces the communication rounds for DPF and DCF from 72 and 1,048, respectively, to 13, while batched FSS requires only one round. Key compression reduces the key size to 1% of that of conventional FSS schemes and achieves key-generation speedups ranging from $3\times$ to $40\times$.

1 Introduction

Function secret sharing (FSS) enables two evaluators to compute shares of a private function on a public input using separately held keys. The output shares reconstruct the function value, while either key hides the function beyond specified public information [6, 7]. Distributed point functions (DPFs) encode equality to a private target, and distributed comparison functions (DCFs) encode comparison against a private threshold. In secure multi-party computation (MPC), these primitives shift function-dependent work into preprocessing and enable local evaluation, reducing online interaction for operations such as comparisons, activations, and private lookups [8, 22]. Such operations are often a major source of communication and interaction in MPC [12, 30], making the cost of generating, delivering, and storing FSS keys an important consideration.

Standard DPF and DCF constructions use a Goldreich–Goldwasser–Micali (GGM)-style pseudorandom tree, in which an n-bit input selects a path through n levels. Each node carries a seed and a control bit; the seed length scales with the security parameter λ of the underlying pseudorandom generator (PRG). At each level, an evaluator expands its seed, selects a child according to the next input bit, and conditionally applies auxiliary values called *correction words* (CWs). The corrections preserve distinct states along the private target path and merge the two states after the first mismatch. A DPF assigns the payload at the target leaf,

while a DCF accumulates output contributions along the traversal to determine whether the input is below the private threshold [6, 7].

This architecture creates two costs. First, explicitly representing the state corrections requires $\Theta(\lambda)$ bits at every level, giving $O(n\lambda)$-bit keys for fixed output length [6, 7]. The multiplicative λ factor therefore reflects seed-sized corrections repeated throughout the tree. Second, generating these corrections is sequential: the generator computes a CW, applies it to obtain the next target-path states, and only then has the seeds needed to compute the following CW. The resulting chain of n dependent stages limits parallel generation.

The private function description is much shorter: a target or threshold $\alpha \in \{0,1\}^n$ and a payload $\beta \in \{0,1\}^m$ together take $n+m$ bits. We use this description length as an amortized compactness benchmark. Can we remove the dependencies between successive CWs and approach $n+m$ bits per evaluator per function, while keeping both α and β private?

Our Approach We first make all target-path states available before generating any CW. We sample the seeds and control bits that the evaluators should hold at every depth, keeping their control bits opposite, and choose a common destination for each branch leaving the target path. Each level's CWs then map the expanded parent states to these prescribed destinations. Because the parent states are already known, generating CW_{i+1} no longer depends on CW_i. The correction-generation depth is determined by the expansion and output maps rather than by a chain of tree levels, enabling shallow homomorphic generation when these maps have low-depth circuits. We further use the Half-Tree relation [18] to share state corrections across both branches while preserving this independence. Evaluation continues to follow the input path sequentially.

Parallel generation still leaves seed-sized corrections to be stored at every level. Our compression step changes their representation: using somewhat homomorphic encryption (SWHE), we replace plaintext state corrections with encrypted branch predicates that act directly on evaluator states, represented by labels. Under the stated SWHE/label interface, a matching branch preserves the difference between the labels, while a mismatch makes them equal. This enforces the target-path condition without recovering a plaintext CW. The DPF uses a terminal correction to assign its payload; the DCF accumulates payload-weighted comparison contributions along the path.

Sending these ciphertext CWs separately would retain a substantial cost at every level. We instead reconstruct them locally from one shared encrypted seed. The seed expands into distinct masks for the batch targets and, for DCF, their payloads; homomorphic expansion recovers the encrypted predicates during evaluation. Each function contributes an n-bit masked target and either an m-bit output correction for DPF or an m-bit masked payload for DCF. The encrypted seed, root labels, and evaluation parameters are shared across the batch.

We formalize this setting as *Batched Function Secret Sharing* (BFSS), with indexed local evaluation, and simulation-based joint privacy. For a batch of N functions, let $K_b^{(N)}$ include all material supplied to evaluator $b \in \{0,1\}$. Our constructions use N records of $n+m$ bits and a shared header of size at most H_T, giving

$$\frac{|K_b^{(N)}|}{N} \leq n+m+\frac{H_T}{N}.$$

All public and auxiliary evaluation material is charged to the header once per evaluator; each function's record is charged separately. A uniform header bound $H_T(\lambda,N)$ independent of n,m yields the overhead $S(\lambda,N) = H_T(\lambda,N)/N$. If this bound is sublinear in N for fixed λ, the overhead vanishes and the amortized key size approaches $n+m$ bits. This compactness result is conditional on N growth, and the privacy guarantee covers the entire batch key, including correlations introduced by the header.

Contributions We make two contributions.

- **Parallel CW generation.** We construct DPF and DCF schemes in which generating a CW does not depend on the preceding CW, so all level CWs can be generated in parallel. We then adapt the Half-Tree correlation to reduce the state-correction material from 4λ to 2λ bits per level while preserving parallel generation. We prove perfect correctness, single-key privacy of the basic schemes under the stated pseudorandomness assumptions, and single-key privacy of the Half-Tree variants in the classical random-oracle model.
- **Formal BFSS and secure key compression.** We give a formal BFSS model with joint batch-key privacy and complete key accounting, and construct batched DPF and DCF protocols using SWHE. Their complete keys have at most $N(n+m)+H_T$ bits per evaluator. A uniform sublinear header bound makes the amortized size approach the $n+m$ compactness benchmark. We prove correctness and joint privacy under the label and cryptographic assumptions stated in Section 4.4; DPF privacy additionally relies on offset-recovery hardness and an independent output random oracle.

Notes Piske et al. [24] introduced the batched FSS concept and proposed an HSS-based framework. The authors reported that their concrete instantiation is insecure [25]. Our contribution is a formal BFSS model and provably secure DPF/DCF constructions with quantified amortized compactness under the stated assumptions.

Organization Section 2 introduces notation, FSS, and the underlying cryptographic primitives. Section 3 defines BFSS and the system model. Section 4 presents the constructions and theorem statements. Section 5 discusses prior work, and Section 6 reports the experiments. Appendix A gives the expanded batch protocols, their correctness and privacy proofs, and the size analysis. Appendix B proves the parallel and Half-Tree results.

2 Preliminaries

2.1 Notation

Let λ denote the security parameter. All security statements are asymptotic in λ: adversaries run in probabilistic polynomial time, and public input lengths are polynomially bounded in λ. A function $\mathsf{negl}(\lambda)$ is negligible if it eventually becomes smaller than the inverse of every positive polynomial. We write $\approx_c$ for computational indistinguishability and $a \leftarrow S$ for uniform sampling from a finite set S. The same arrow denotes algorithm outputs and state updates when the context is clear.

The input length, output length, and batch size are $n, m, N \geq 1$. Write $[t] = \{1, \ldots, t\}$ for a positive integer t. The evaluator index is $b \in \{0,1\}$; $q \in [N]$ indexes a function in a batch, $i \in [n]$ a tree level, and $\ell \in [m]$ a payload bit. A branch is indexed by d or j in $\{0,1\}$, with $L = 0$ and $R = 1$. The target or threshold of instance q is $\alpha_q \in \{0,1\}^n$, and its payload is $\beta_q \in \{0,1\}^m$. We use Q for an adversary's oracle-query budget.

For $x = x_1 \cdots x_n \in \{0,1\}^n$, x_1 is the most significant bit and $x_{1:i}$ denotes its length-i prefix, with $x_{1:0}$ the empty string. Comparisons use the corresponding unsigned integer order. For a predicate P, $[P] \in \{0,1\}$ denotes its indicator. We write $\oplus$ for bitwise XOR, with operands of matching lengths, and LSB for the least significant bit of a specified encoding.

2.2 Function Secret Sharing

A two-party function secret sharing scheme for a public family $\mathscr{F}$ of functions from $\{0,1\}^n$ to $\{0,1\}^m$ consists of a randomized algorithm $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda, f)$ that takes a description of $f \in \mathscr{F}$, and a deterministic local algorithm $y_b \leftarrow \mathsf{Eval}(k_b, x)$ [6]. Both algorithms run in time polynomial in the security parameter and their input lengths. For every supported function and input, correctness requires

$$\Pr[\mathsf{Eval}(k_0, x) \oplus \mathsf{Eval}(k_1, x) = f(x)] \geq 1 - \mathsf{negl}(\lambda),$$

where the probability is over key generation. Correctness is perfect when this probability is one. Single-key privacy requires either evaluator's key to be computationally indistinguishable from the output of a probabilistic polynomial-time simulator given only the security parameter, evaluator index, and public family parameters.

We consider distributed point functions (DPFs) and distributed comparison functions (DCFs). For $\alpha \in \{0,1\}^n$ and $\beta \in \{0,1\}^m$, their functions are

$$f^{\mathsf{DPF}}_{\alpha,\beta}(x) = \beta[x = \alpha],$$
$$f^{\mathsf{DCF}}_{\alpha,\beta}(x) = \beta[x < \alpha].$$

Here $\beta[P]$ equals β if P holds and 0^m otherwise. The public family parameters are the type $T \in \{\mathsf{DPF}, \mathsf{DCF}\}$ and lengths (n,m); both α and β are private. For these families, we also write key generation as $\mathsf{Gen}(1^\lambda, \alpha, \beta)$ with the public type implicit.

2.3 Pseudorandomness and Random Oracles

A pseudorandom generator (PRG) deterministically expands a uniform secret seed of length $\Theta(\lambda)$ into a longer bit string that is computationally indistinguishable from a uniform string of the same length. The output length is public and polynomially bounded in λ. We also use truncations when fewer output bits are needed. When an expansion is parsed into several fields, pseudorandomness is required jointly for the complete output tuple. The seed and output lengths of each map are specified in the corresponding construction.

A random oracle is a public random function that returns the same output on repeated queries to the same input. We model distinct map roles as independent random oracles where stated. These roles can be represented by disjoint domains of one oracle using prefix-free tags. Indistinguishability in this model includes the adversary's access to the public oracles.

2.4 Somewhat Homomorphic Encryption

A bit-encryption scheme SWHE provides setup $(\mathsf{pp}, \mathsf{sk}) \leftarrow \mathsf{SWHE.Gen}(1^\lambda)$, encryption $c \leftarrow \mathsf{Enc}_{\mathsf{sk}}(z)$, public homomorphic evaluation HEval, and plaintext decryption $\mathsf{Decrypt}_{\mathsf{sk}}$. The public material pp specifies the supported class of bounded-depth Boolean circuits and includes all public evaluation keys. Public parameters are implicit inputs to encryption and decryption. A hat denotes an encrypted value; for example, $\widehat{z}$ encrypts the bit z.

For a supported circuit C and bit vector $\boldsymbol{z}$, let $\boldsymbol{c}$ contain encryptions of its components. Homomorphic correctness requires

$$\Pr\big[\mathsf{Decrypt}_{\mathsf{sk}}\big(\mathsf{HEval}(\mathsf{pp}, C, \boldsymbol{c})\big) = C(\boldsymbol{z})\big] \geq 1 - \mathsf{negl}(\lambda),$$

with decryption applied componentwise for multiple outputs. The probability is over setup, encryption, and any evaluation randomness. On ciphertexts, XOR, complement, and multiplication denote homomorphic evaluation of the corresponding Boolean gates. Writing a reconstructed ciphertext as $\mathsf{Enc}_{\mathsf{sk}}(z)$ specifies its encrypted plaintext, without asserting byte-for-byte equality to a fresh encryption.

We require IND-CPA security: an efficient adversary with the public material and encryption access cannot distinguish encryptions of two chosen equal-length plaintext vectors with nonnegligible advantage. This requirement covers polynomially many bit encryptions under one setup and includes the public evaluation material in the adversary's view.

3 Problem Statement

3.1 Setting and Threat Model

A function owner holds a batch of N point or comparison functions and distributes one complete batch key to each of two evaluators P_0, P_1. Given a common function index $q \in [N]$ and input $x \in \{0,1\}^n$, each evaluator computes an m-bit share locally. XORing the shares reconstructs the selected function value. The targets or thresholds and payloads are private; the function type, lengths, and batch size are public.

We consider a static, semi-honest adversary that corrupts at most one evaluator. The owner generates the keys honestly, and the evaluators do not collude. The protected view consists of the corrupted evaluator's complete batch key and its local computations. Both the index and input are public to the evaluators. Disclosure through reconstructed function values is governed by the application's output policy; correctness assumes honest generation and evaluation.

3.2 Batched Function Secret Sharing

Let $\mathscr{F}^T_{n,m}$ be the point-function or strict-comparison family defined in Section 2.2, with public type $T \in \{\mathsf{DPF}, \mathsf{DCF}\}$. A description $\widehat{f_q} = (T, n, m, \alpha_q, \beta_q)$ specifies $f_q = f^T_{\alpha_q,\beta_q}$. A batch $\widehat{\boldsymbol{f}} = (\widehat{f_1}, \ldots, \widehat{f_N})$ contains functions with the same public type and lengths. We allow leakage

$$\mathsf{Leak}(\widehat{\boldsymbol{f}}) = (T, n, m, N).$$

Public cryptographic parameter choices depend only on λ and this leakage. All security requirements below apply to supported batches of polynomial total description length in λ.

Definition 1 (Batched Function Secret Sharing)**.** A two-party *Batched Function Secret Sharing* (BFSS) scheme with amortized overhead bound S consists of a randomized generation algorithm and a deterministic evaluation algorithm, each running in time polynomial in λ and its input length:

- $(K_0^{(N)}, K_1^{(N)}) \leftarrow \mathsf{Gen}(1^\lambda, \widehat{\boldsymbol{f}})$ generates one complete batch key for each evaluator.
- $y_b \leftarrow \mathsf{Eval}(K_b^{(N)}, q, x)$ returns a share in $\{0,1\}^m$ for $q \in [N]$ and $x \in \{0,1\}^n$, without interaction.

The function $S(\lambda, N)$ is nonnegative and independent of n, m. The scheme satisfies the following requirements.

1. **Correctness.** For every batch $\widehat{\boldsymbol{f}}$ and every fixed tuple of inputs $(x_1, \ldots, x_N)$, generate $(K_0^{(N)}, K_1^{(N)})$ as above and set $y_{b,q} = \mathsf{Eval}(K_b^{(N)}, q, x_q)$. Then

$$\Pr[\forall q \in [N] : \; y_{0,q} \oplus y_{1,q} = f_q(x_q)] \geq 1 - \mathsf{negl}(\lambda),$$

   where the probability is over generation randomness. Correctness is perfect when this probability is one.

2. **Joint batch privacy.** There exists a probabilistic polynomial-time simulator Sim such that, for either evaluator b and every supported sequence of batch descriptions,

$$\left\{K_b^{(N)}\right\}_\lambda \approx_c \left\{\mathsf{Sim}(1^\lambda, b, \mathsf{Leak}(\widehat{\boldsymbol{f}}))\right\}_\lambda,$$

   where the real key is evaluator b's output of $\mathsf{Gen}(1^\lambda, \widehat{\boldsymbol{f}})$. The view includes all shared and instance-specific evaluation material, as well as public-oracle access when the construction is analyzed in a random-oracle model.

3. **Amortized key size.** Under a fixed encoding with public lengths, every generated key satisfies

$$\frac{|K_b^{(N)}|}{N} \leq n + m + S(\lambda, N). \tag{1}$$

The key length counts all material supplied to enable evaluation, including public and auxiliary evaluation parameters. Material reused across the batch is counted once per evaluator's key; instance-specific material is counted for every function.

For $T = \mathsf{DPF}$ and $T = \mathsf{DCF}$, we call the resulting schemes BDPF and BDCF, respectively. Joint privacy protects the batch as a whole, including correlations among its key components. Since evaluation is deterministic and local, the same guarantee covers polynomially many adaptively chosen indices and inputs. The definition permits any internal representation of the complete batch key.

3.3 Design Objectives

The $n+m$ bits describing a target or threshold and its payload provide our compactness benchmark. We seek constructions whose overhead in (1) satisfies

$$S(\lambda, N) \longrightarrow 0 \qquad \text{as } N \longrightarrow \infty$$

for fixed λ. This convergence is an efficiency objective beyond Definition 1; cryptographic negligibility is measured separately as λ grows. For example, a total key bound $|K_b^{(N)}| \leq N(n+m) + H(\lambda, N)$ uniform in n, m gives $S(\lambda, N) = H(\lambda, N)/N$, and sublinear growth $H(\lambda, N) = o(N)$ suffices to approach the benchmark. An N-independent overhead $H(\lambda)$ gives the special case $S(\lambda, N) = H(\lambda)/N$.

We also seek to reduce sequential dependencies in CW generation. We measure generation work and circuit depth separately from evaluation work and sequential depth, and report communication volume and round complexity for interactive generation. Key representation size, reconstruction work, and working memory are separate costs. Parallel generation is an efficiency objective evaluated separately from the BFSS requirements.

4 Protocol Design

We first revisit FSS to identify the generation dependencies and key-size costs of classical tree-based constructions. We then present parallel CW generation and its Half-Tree optimization, followed by SWHE-based key compression and an analysis of the amortized batch key size.

4.1 Revisiting Classical Tree-Based FSS

A binary tree of depth n represents the input domain: reading x_i selects branch $L = 0$ or $R = 1$ at level i. Each evaluator stores a seed and a control bit at its current node. Write a full state as

$$U = (u, \tau) \in \{0,1\}^{\lambda - 1} \times \{0,1\},$$

where u is the $(\lambda - 1)$-bit seed and τ is the control bit. Throughout the tree constructions below, a full state has λ bits, including its control bit. The seed is expanded to produce children; the control bit selects whether to apply a shared correction. For a bit t and a bit string C, the product tC means C if $t = 1$ and the all-zero string otherwise.

The corrected state of evaluator $b \in \{0,1\}$ on the target prefix $\alpha_{1:i}$ is denoted by $U_b^i = (u_b^i, \tau_b^i)$, with $i = 0$ denoting the root. Generation initializes the roots with $\tau_0^0 \oplus \tau_1^0 = 1$. In evaluation, (s, t) denotes the current

local seed and control bit on the actual input path. After i steps it equals U_b^i when $x_{1:i} = \alpha_{1:i}$. Thus uppercase U denotes the complete state, lowercase u its seed component, and τ its control component.

At level i, expand each target-path seed as

$$(V_{b,L}^i, V_{b,R}^i) \leftarrow G(u_b^{i-1}), \qquad V_{b,j}^i = (v_{b,j}^i, \eta_{b,j}^i).$$

For branch $j \in \{L,R\}$, $V_{b,j}^i$ is an uncorrected child state: $v_{b,j}^i$ is its $(\lambda-1)$-bit seed and $\eta_{b,j}^i$ its control bit. The corrected states must satisfy two conditions [7]: on the target branch their control bits are opposite, while on the other branch their complete states coincide. Equal states then produce equal subsequent computations and cancel under XOR reconstruction. Equivalently, after reading $x_{1:i}$, the two current control bits XOR to $[x_{1:i} = \alpha_{1:i}]$.

To see how a CW enforces these conditions, let $\bar{\alpha}_i = 1 - \alpha_i$ be the branch leaving the target. In the standard shared-seed-correction form, define

$$a_i = v_{0,\bar{\alpha}_i}^i \oplus v_{1,\bar{\alpha}_i}^i,$$
$$c_{i,j} = \eta_{0,j}^i \oplus \eta_{1,j}^i \oplus [j = \alpha_i],$$

and store $CW_i = (a_i, c_{i,L}, c_{i,R})$. Thus a_i is the seed correction shared by the two branches and $c_{i,j}$ is a branch-specific control correction. Each evaluator applies these corrections to the raw child when its parent control bit is one. On the target branch $j = \alpha_i$, the corrected child is precisely the next target-path state $U_b^i = (u_b^i, \tau_b^i)$:

$$U_b^i = V_{b,\alpha_i}^i \oplus \tau_b^{i-1}(a_i, c_{i,\alpha_i}). \tag{2}$$

Because $\tau_0^{i-1} \oplus \tau_1^{i-1} = 1$, the two off-target children become identical and the two target children retain opposite control bits. Starting from roots with opposite controls, this maintains a single active target path. A terminal output correction assigns the DPF payload β at its leaf.

This update also explains the generation dependency. The generator computes CW_i from the current target-path expansions and then uses (2) to obtain U_0^i, U_1^i. These corrected states supply the seeds needed to generate CW_{i+1}:

$$CW_i \longrightarrow (U_0^i, U_1^i) \longrightarrow CW_{i+1}.$$

Consequently, the conventional generation circuit contains n dependent expansion-and-correction stages. The correction rule fixes the relationship between the children, while their actual target-path values are determined by the corrected expansion. Our next construction makes those values available before any correction is computed. The classical CW also carries a seed-sized field at every level, accounting for the $O(n\lambda)$ recurring state-correction cost [7].

The state updates also track prefix equality for strict comparison: the first differing input bit determines the answer. Equivalently,

$$[x < \alpha] = \bigoplus_{i=1}^{n} [x_{1:i-1} = \alpha_{1:i-1}]\alpha_i(1 - x_i).$$

The terms select disjoint left subtrees branching from the target path [6]. A DCF therefore uses these state updates to track prefix equality and accumulates output contributions along the path. This separates the two tasks needed below: state CWs track whether a prefix still matches, and output CWs encode the contribution to the comparison result.

4.2 Parallel CW Generation via Prescribed States

Correctness requires the two evaluators to have opposite control bits on matching prefixes and identical states after the first mismatch. These conditions leave the actual target-path state values free to choose. We exploit this freedom by sampling all target-path states before computing the CWs that connect them.

Concretely, for each $i \in \{0,\dots,n\}$, independently sample seeds $u_0^i, u_1^i \in \{0,1\}^{\lambda-1}$ and a control bit τ_0^i, and set $\tau_1^i = 1 \oplus \tau_0^i$. These define the prescribed full states $U_b^i = (u_b^i, \tau_b^i)$. For each level $i \in [n]$, also sample $Q_i \in \{0,1\}^\lambda$, the common state that both evaluators should reach on the branch leaving the target. Denote the desired child on branch j by

$$T_{b,j}^i = \begin{cases} U_b^i, & j = \alpha_i, \\ Q_i, & j \neq \alpha_i. \end{cases}$$

Thus U_b^i is a prescribed target-path state, Q_i a prescribed merged state, and $T_{b,j}^i$ selects the appropriate destination. All these states are inputs to CW generation.

Given these desired children, expanding u_b^{i-1} gives the raw children $V_{b,j}^i$ defined above. To reach an arbitrary prescribed destination, we use a control-dependent correction $C_{i,j} \in \{0,1\}^\lambda$ and a common alignment correction $R_{i,j} \in \{0,1\}^\lambda$. Both are included in the keys of both evaluators. We require

$$V_{b,j}^i \oplus \tau_b^{i-1} C_{i,j} \oplus R_{i,j} = T_{b,j}^i. \tag{3}$$

XORing the equations for $b = 0, 1$ and using the opposite parent controls gives $C_{i,j}$; substitution gives $R_{i,j}$:

$$\begin{aligned} C_{i,j} &= V_{0,j}^i \oplus V_{1,j}^i \oplus T_{0,j}^i \oplus T_{1,j}^i, \\ R_{i,j} &= T_{0,j}^i \oplus V_{0,j}^i \oplus \tau_0^{i-1} C_{i,j}. \end{aligned} \tag{4}$$

The first term enforces the desired difference between the two children; the second sets their absolute values. Each level stores $CW_i = (C_{i,L}, R_{i,L}, C_{i,R}, R_{i,R})$.

Every quantity on the right-hand side is determined by the preselected states and the current target bit. Computing CW_i therefore requires no earlier CW. At evaluation time, (s,t) is updated by $V_j \oplus t C_{i,j} \oplus R_{i,j}$ on branch $j = x_i$. A matching prefix reaches U_b^i; the first mismatch sends both parties to Q_i; subsequent equal states remain equal under the common deterministic update.

The DPF supports any payload length $m \geq 1$ through a public deterministic output map $F : \{0,1\}^{\lambda-1} \to \{0,1\}^m$. Set

$$CW_{\mathsf{out}} = F(u_0^n) \oplus F(u_1^n) \oplus \beta$$

and output $F(s) \oplus t CW_{\mathsf{out}}$ from terminal state (s,t). The map F converts the seed to an m-bit output, keeping the payload length independent of the seed length. Opposite terminal controls select one copy of the correction at the target, whereas equal terminal states cancel elsewhere. Since the terminal states were also sampled in advance, this output correction can be generated alongside all level CWs. Figure 1 specifies the protocol.

For a DCF, retain the same state update and extend G to output an m-bit mask $Z_{b,j}^i$ for each branch. Following the comparison decomposition in Section 4.1, the desired contribution is $\Gamma_{i,j} = \beta[\alpha_i = 1 \text{ and } j = L]$. Store

$$D_{i,j} = Z_{0,j}^i \oplus Z_{1,j}^i \oplus \Gamma_{i,j}.$$

Using its current parent control t, an evaluator accumulates $Z_j \oplus t D_{i,j}$ before updating (s,t). A matching branch contributes zero. The first differing branch contributes β exactly when $x < \alpha$, and later contributions cancel because the states have merged. Each $D_{i,j}$ depends only on preselected parent states, α_i, and β, so it preserves parallel generation. Figure 2 gives the protocol for general m-bit payloads.

The following theorem records correctness, displayed key lengths, and correction-generation depth.

Theorem 1 (Parallel DPF and DCF)**.** *Let $n, m \geq 1$. The protocols in Figures 1 and 2 are perfectly correct for their respective point and comparison functions with key lengths*

$$\begin{aligned} L_{\mathsf{PDPF}} &= (4n+1)\lambda + m, \\ L_{\mathsf{PDCF}} &= (4n+1)\lambda + 2nm. \end{aligned}$$

Protocol Π_{PDPF}

Use λ-bit states $U = (u, \tau)$, $G : \{0,1\}^{\lambda-1} \to (\{0,1\}^{\lambda})^2$, and a public output map $F : \{0,1\}^{\lambda-1} \to \{0,1\}^m$.

$\mathsf{Gen}(\alpha, \beta)$

1. For every $i \in \{0,\ldots,n\}$, sample independent $u_0^i, u_1^i \leftarrow \{0,1\}^{\lambda-1}$ and $\tau_0^i \leftarrow \{0,1\}$. Set $\tau_1^i = 1 \oplus \tau_0^i$ and $U_b^i = (u_b^i, \tau_b^i)$ for both b.
2. For each $i \in [n]$, sample $Q_i \leftarrow \{0,1\}^{\lambda}$.
3. For each level $i \in [n]$, in parallel, expand $(V_{b,L}^i, V_{b,R}^i) \leftarrow G(u_b^{i-1})$ for both b. Set $T_{b,\alpha_i}^i = U_b^i$ and $T_{b,1-\alpha_i}^i = Q_i$. For both branches $j \in \{L,R\}$, compute
$$C_{i,j} = V_{0,j}^i \oplus V_{1,j}^i \oplus T_{0,j}^i \oplus T_{1,j}^i,$$
$$R_{i,j} = T_{0,j}^i \oplus V_{0,j}^i \oplus \tau_0^{i-1} C_{i,j}.$$
Set $CW_i = (C_{i,L}, R_{i,L}, C_{i,R}, R_{i,R})$.
4. Set $CW_{\mathsf{out}} = F(u_0^n) \oplus F(u_1^n) \oplus \beta$. Output $k_b = (U_b^0, CW_1, \ldots, CW_n, CW_{\mathsf{out}})$ for both evaluators.

$\mathsf{Eval}(k_b, x)$

1. Initialize $(s,t) \leftarrow U_b^0$.
2. For $i = 1,\ldots,n$, compute $(V_L, V_R) \leftarrow G(s)$, set $j = x_i$, and update $(s,t) \leftarrow V_j \oplus t C_{i,j} \oplus R_{i,j}$.
3. Output $y_b = F(s) \oplus t CW_{\mathsf{out}}$.

Figure 1: DPF with parallel correction-word generation.

Let d_G and d_F be the Boolean circuit depth of function G and F. With sufficient parallel processors, correction-generation depth is $\max\{d_G, d_F\} + O(1)$ for the DPF and $d_G + O(1)$ for the DCF.

The proof is given in Appendix B.1.

Correctness uses only the prescribed-state identities. Key privacy additionally requires pseudorandom expansion: the entire output of G, including both state branches and any output masks, must be jointly pseudorandom. For the DPF, we also require $F(s)$ on a uniform secret seed $s \in \{0,1\}^{\lambda-1}$ to be computationally indistinguishable from a uniform m-bit string.

Theorem 2 (Parallel key privacy)**.** *Let $n, m \geq 1$ be polynomially bounded in λ. Under the pseudorandomness assumptions above, the protocols in Figures 1 and 2 satisfy single-key privacy: a simulator given $(1^\lambda, b, T, n, m)$ produces a key computationally indistinguishable from evaluator b's key for any target and payload.*

The proof is given in Appendix B.2.

The tree expansions and DPF terminal output maps are evaluated in parallel, followed by local corrections. Low-depth choices of these maps therefore allow homomorphic generation without a chain of corrected states. The storage cost of prescribing arbitrary destinations is four λ-bit state corrections per level. We next reduce this cost by arranging for both branches to share the same C and the same R.

Protocol Π_{PDCF}

Use λ-bit states $U = (u, \tau)$ and an expansion G producing a λ-bit state and an m-bit output mask for each branch.

$\mathsf{Gen}(\alpha, \beta)$

1. For every $i \in \{0, \ldots, n\}$, sample independent $u_0^i, u_1^i \leftarrow \{0,1\}^{\lambda-1}$ and $\tau_0^i \leftarrow \{0,1\}$. Set $\tau_1^i = 1 \oplus \tau_0^i$ and $U_b^i = (u_b^i, \tau_b^i)$ for both b.
2. For each $i \in [n]$, sample $Q_i \leftarrow \{0,1\}^{\lambda}$.
3. For each level $i \in [n]$, in parallel, compute
$$\big((V_{b,L}^i, Z_{b,L}^i), (V_{b,R}^i, Z_{b,R}^i)\big) \leftarrow G(u_b^{i-1})$$
for both b. Set $T_{b,\alpha_i}^i = U_b^i$ and $T_{b,1-\alpha_i}^i = Q_i$. For both branches $j \in \{L, R\}$, compute
$$\begin{aligned} C_{i,j} &= V_{0,j}^i \oplus V_{1,j}^i \oplus T_{0,j}^i \oplus T_{1,j}^i, \\ R_{i,j} &= T_{0,j}^i \oplus V_{0,j}^i \oplus \tau_0^{i-1} C_{i,j}, \\ \Gamma_{i,j} &= \beta[\alpha_i = 1 \text{ and } j = L], \\ D_{i,j} &= Z_{0,j}^i \oplus Z_{1,j}^i \oplus \Gamma_{i,j}. \end{aligned}$$
Set $CW_i = (C_{i,L}, R_{i,L}, D_{i,L}, C_{i,R}, R_{i,R}, D_{i,R})$.
4. Output $k_b = (U_b^0, CW_1, \ldots, CW_n)$ for both evaluators.

$\mathsf{Eval}(k_b, x)$

1. Initialize $(s,t) \leftarrow U_b^0$ and $y \leftarrow 0^m$.
2. For $i = 1, \ldots, n$, compute $((V_L, Z_L), (V_R, Z_R)) \leftarrow G(s)$ and set $j = x_i$. First accumulate $y \leftarrow y \oplus Z_j \oplus t D_{i,j}$; then update $(s,t) \leftarrow V_j \oplus t C_{i,j} \oplus R_{i,j}$.
3. Output $y_b = y$.

Figure 2: DCF with parallel correction-word generation.

4.3 Reducing the Parallel CW Size

The previous construction treats the two branches independently. Half-Tree [18] supplies a correlated expansion that lets us relate their correction requirements. We keep the same λ-bit full states, with $\lambda - 1$ seed bits and a control bit. For each level i, let $H_i : \{0,1\}^{\lambda} \to \{0,1\}^{\lambda}$ and expand the complete parent state by

$$V_{b,L}^i = H_i(U_b^{i-1}), \qquad V_{b,R}^i = U_b^{i-1} \oplus V_{b,L}^i.$$

Thus $V_{b,L}^i \oplus V_{b,R}^i = U_b^{i-1}$. Here H_i takes the full state, including its control bit; the preceding G took only the seed. We use the Half-Tree relation to share corrections while preserving the prescribed-state generation strategy.

To share the difference correction, choose a single seed offset $\delta \leftarrow \{0,1\}^{\lambda-1}$ and append a control offset of one to form $\Delta = (\delta, 1)$. Independently sample $U_0^i \leftarrow \{0,1\}^{\lambda}$ at every depth and set $U_1^i = U_0^i \oplus \Delta$. All target-path pairs now have the same full-state XOR difference Δ, and their controls are opposite.

Keep the desired destinations $T_{b,\alpha_i}^i = U_b^i$ and $T_{b,1-\alpha_i}^i = Q_i$. Across the two branches, the desired inter-party differences are Δ on the target branch and zero on the other branch. Applying (4) together with the Half-Tree relation therefore gives

$$C_{i,L} \oplus C_{i,R} = (U_0^{i-1} \oplus U_1^{i-1}) \oplus \Delta = 0.$$

Hence one correction C_i serves both branches. Reusing the same offset at adjacent depths is what makes the expanded-state and desired-state differences agree.

Protocol $\Pi_{\mathsf{HT\text{-}PDPF}}$

Use λ-bit states, $H_i : \{0,1\}^\lambda \to \{0,1\}^\lambda$, and $F : \{0,1\}^{\lambda-1} \to \{0,1\}^m$, with the independent domains specified in the text.

$\mathsf{Gen}(\alpha, \beta)$

1. Sample $\delta \leftarrow \{0,1\}^{\lambda-1}$ and set $\Delta = (\delta, 1)$.
2. For every $i \in \{0,\dots,n\}$, sample $U_0^i = (u_0^i, \tau_0^i) \leftarrow \{0,1\}^\lambda$ and set $U_1^i = U_0^i \oplus \Delta$.
3. For each level $i \in [n]$, in parallel, set the prescribed merged child to $Q_i = U_0^{i-1} \oplus U_0^i$. For both b, compute

$$V_{b,L}^i = H_i(U_b^{i-1}), \qquad V_{b,R}^i = U_b^{i-1} \oplus V_{b,L}^i.$$

   Using the target branch, compute

$$C_i = V_{0,\alpha_i}^i \oplus V_{1,\alpha_i}^i \oplus \Delta,$$
$$R_i = U_0^i \oplus V_{0,\alpha_i}^i \oplus \tau_0^{i-1} C_i.$$

   The same pair corrects both branches. Set $CW_i = (C_i, R_i)$.
4. Set $CW_{\mathsf{out}} = F(u_0^n) \oplus F(u_1^n) \oplus \beta$. Output $k_b = (U_b^0, CW_1, \dots, CW_n, CW_{\mathsf{out}})$ for both evaluators.

$\mathsf{Eval}(k_b, x)$

1. Initialize $(s,t) \leftarrow U_b^0$.
2. For $i = 1,\dots,n$, compute $V_L = H_i((s,t))$ and $V_R = (s,t) \oplus V_L$. Set $j = x_i$ and update $(s,t) \leftarrow V_j \oplus tC_i \oplus R_i$.
3. Output $y_b = F(s) \oplus tCW_{\mathsf{out}}$.

Figure 3: Parallel Half-Tree DPF with shared branch corrections.

A common C_i alone does not force the two alignment corrections to agree. Their remaining difference is

$$R_{i,L} \oplus R_{i,R} = U_0^i \oplus Q_i \oplus U_0^{i-1}.$$

We therefore choose $Q_i = U_0^{i-1} \oplus U_0^i$, making this difference zero as well. Both branches can now use $CW_i = (C_i, R_i)$, reducing four full-state corrections to two. Each Q_i is computed from already sampled states, so this constraint introduces no dependency on earlier CWs. Appendix B.3 verifies the complete state-update identities.

The Half-Tree DPF retains the terminal output map $F : \{0,1\}^{\lambda-1} \to \{0,1\}^m$ from Section 4.2, so it supports arbitrary m independently of the seed length. Set

$$CW_{\mathsf{out}} = F(u_0^n) \oplus F(u_1^n) \oplus \beta,$$

and output $F(s) \oplus tCW_{\mathsf{out}}$ from terminal state (s,t). This retains the same target-versus-merged-state reconstruction rule as the previous DPF. The terminal map has a separate domain from the state-expansion maps, as required by the privacy analysis below. Figure 3 specifies the protocol.

For comparison functions, the preceding DCF still stores an output correction for each branch. To reduce this to one correction per level, use one mask $Z_b^i = J_i(U_b^{i-1})$ for both branches, where $J_i : \{0,1\}^\lambda \to \{0,1\}^m$ has a domain independent of H_i and F. This requires a different contribution encoding: a single D_i cannot directly encode the two branch-dependent values $\Gamma_{i,L}, \Gamma_{i,R}$ used previously.

Set $\alpha_0 = 0$ and encode changes between adjacent threshold bits by

$$\Gamma_i = (\alpha_i \oplus \alpha_{i-1})\beta, \qquad D_i = Z_0^i \oplus Z_1^i \oplus \Gamma_i.$$

Protocol $\Pi_{\mathsf{HT\text{-}PDCF}}$

Use λ-bit states, $H_i : \{0,1\}^\lambda \to \{0,1\}^\lambda$, $J_i : \{0,1\}^\lambda \to \{0,1\}^m$, and $F : \{0,1\}^{\lambda-1} \to \{0,1\}^m$, with independent domains.

$\mathsf{Gen}(\alpha, \beta)$

1. Sample $\delta \leftarrow \{0,1\}^{\lambda-1}$ and set $\Delta = (\delta, 1)$.
2. For every $i \in \{0,\ldots,n\}$, sample $U_0^i = (u_0^i, \tau_0^i) \leftarrow \{0,1\}^\lambda$ and set $U_1^i = U_0^i \oplus \Delta$.
3. Set $\alpha_0 = 0$. For each level $i \in [n]$, in parallel, set the prescribed merged child to $Q_i = U_0^{i-1} \oplus U_0^i$. For both b, compute

$$V_{b,L}^i = H_i(U_b^{i-1}),$$
$$V_{b,R}^i = U_b^{i-1} \oplus V_{b,L}^i, \qquad Z_b^i = J_i(U_b^{i-1}).$$

   Compute

$$C_i = V_{0,\alpha_i}^i \oplus V_{1,\alpha_i}^i \oplus \Delta,$$
$$R_i = U_0^i \oplus V_{0,\alpha_i}^i \oplus \tau_0^{i-1} C_i,$$
$$\Gamma_i = (\alpha_i \oplus \alpha_{i-1})\beta,$$
$$D_i = Z_0^i \oplus Z_1^i \oplus \Gamma_i.$$

   Set $CW_i = (C_i, R_i, D_i)$.

4. Set $D_{\mathsf{out}} = F(u_0^n) \oplus F(u_1^n) \oplus \alpha_n\beta$. Output $k_b = (U_b^0, CW_1, \ldots, CW_n, D_{\mathsf{out}})$ for both evaluators.

$\mathsf{Eval}(k_b, x)$

1. Initialize $(s,t) \leftarrow U_b^0$ and $y \leftarrow 0^m$.
2. For $i = 1,\ldots,n$, compute $V_L = H_i((s,t))$, $V_R = (s,t) \oplus V_L$, and $Z = J_i((s,t))$. Set $j = x_i$. First accumulate $y \leftarrow y \oplus Z \oplus tD_i$; then update $(s,t) \leftarrow V_j \oplus tC_i \oplus R_i$.
3. Output $y_b = y \oplus F(s) \oplus tD_{\mathsf{out}}$.

Figure 4: Half-Tree DCF with shared state and output corrections.

Before each state update, an evaluator accumulates $Z \oplus tD_i$. If the first mismatch occurs at level d, the contributions up to that level telescope to $\alpha_d\beta$. This equals β precisely when $x_d = 0$ and $\alpha_d = 1$; all later terms cancel after the states merge. If $x = \alpha$, the accumulated value is $\alpha_n\beta$. The terminal correction

$$D_{\mathsf{out}} = F(u_0^n) \oplus F(u_1^n) \oplus \alpha_n\beta$$

removes that value on the matching input, enforcing strict comparison. Each level now stores (C_i, R_i, D_i). The full protocol appears in Figure 4; Appendix B.3 gives the proof.

All prescribed states and terminal seeds are available before CW generation. The following theorem records the resulting correctness, key lengths, and correction-generation depth.

Theorem 3 (Parallel Half-Tree DPF and DCF)**.** *Let $n, m \geq 1$. The protocols in Figures 3 and 4 are perfectly correct for their respective point and comparison functions with key lengths*

$$L_{\mathsf{HT\text{-}DPF}} = (2n+1)\lambda + m,$$
$$L_{\mathsf{HT\text{-}DCF}} = (2n+1)\lambda + (n+1)m.$$

Let d_H, d_J, d_F be the Boolean circuit depths of H_i, J_i, F, taking the maximum over levels where applicable. With sufficient parallel processors, correction-generation depth is $\max\{d_H, d_F\} + O(1)$ for the DPF and $\max\{d_H, d_J, d_F\} + O(1)$ for the DCF.

The proof is given in Appendix B.3.

For privacy, the H_i domains, the DCF domains J_i, and the terminal domain F are modeled as independent random oracles. They can be represented through one public hash interface with prefix-free tags for protocol type, public lengths, map role, and level where applicable. This separation is part of the construction analyzed by the proof, and covers the distinct state and output roles of the maps.

Theorem 4 (Half-Tree key privacy)**.** *Let $n, m \geq 1$ be polynomially bounded in λ. In the independent random-oracle model above, the protocols in Figures 3 and 4 satisfy single-key privacy. There is a simulator given $(1^\lambda, b, T, n, m)$ such that any classical adversary making at most Q oracle queries distinguishes its output from evaluator b's honestly generated key with advantage at most*

$$O\left(\frac{(n+Q+1)^2}{2^{\lambda-1}}\right).$$

This advantage is negligible for polynomial Q.

The proof is given in Appendix B.4.

Theorem 4 concerns one function key. Its proof and the role of domain separation are analyzed in Appendix B. These keys retain $\Theta(n\lambda)$ state-correction bits; independent repetition therefore does not give vanishing amortized overhead.

4.4 Key Compression with Encrypted Correction Words

Parallel generation removes the dependence between successive CWs, but the resulting keys still carry seed-sized corrections at every level. To reduce this representation cost, we change what the evaluator receives: a masked function description and a shared encrypted seed from which it reconstructs the required ciphertext CWs locally. Across a batch, the seed, evaluation parameters, and root labels are stored once; only the target and output information remain instance-specific. We first explain how ciphertext CWs update labels, then derive the batch key bound.

A state correction enforces a simple condition: the two states remain distinct while the input prefix matches the target, and coincide after the first mismatch. We encode this condition in the difference between two labels over a finite ring R_p. For compression, write the SWHE setup as $(\mathsf{pp}, \Delta) \leftarrow \mathsf{SWHE.Gen}(1^\lambda)$, where the secret parameter $\Delta \in R_p$ also serves as the label offset. In addition to the plaintext decryption of Section 2.4, the construction requires a public deterministic map $\mathsf{Dec}(A, \tau) \in R_p$ that takes a label and a ciphertext and returns a label. For every reachable A and supported ciphertext τ encrypting a bit z, we require

$$\begin{aligned} \mathsf{Dec}(A+\Delta, \tau) - \mathsf{Dec}(A, \tau) &= \mathsf{Dec}(\Delta, \tau) \\ &= z\Delta. \end{aligned} \tag{5}$$

The fixed label encoding must also satisfy

$$\mathsf{LSB}(A+\Delta) \oplus \mathsf{LSB}(A) = 1. \tag{6}$$

These requirements apply to ciphertexts obtained by homomorphic evaluation as well as fresh encryption, and are additional assumptions on the SWHE/label interface used by this construction.

By (5), an encryption of one preserves a label difference of Δ, whereas an encryption of zero cancels it. Equal labels remain equal because the map is deterministic. Together with (6), a difference $z\Delta$ for $z \in \{0,1\}$ therefore yields extracted bits whose XOR is z. Each evaluator applies the map locally to its label and the common ciphertext.

We use an encryption of the branch predicate as the ciphertext CW for level i and branch d:

$$\tau_{q,i,d} = \mathsf{Enc}_{\Delta}([d = \alpha_{q,i}]).$$

Starting from roots whose difference is Δ, each evaluator uses this ciphertext CW to update its label on branch x_i and adds the same PRF-derived term ρ_{q,i,x_i}. The additive terms cancel between evaluators, giving

$$A^i_{1,q}(x) - A^i_{0,q}(x) = [x_{1:i} = \alpha_{q,1:i}]\Delta.$$

An encrypted branch predicate therefore supplies the state relation previously enforced by a seed correction.

For a DPF, the final labels coincide away from α_q and differ by Δ at α_q. Let H_q be an m-bit public output map indexed by the instance. The generator follows the target path and computes

$$C_q = H_q(A^n_{0,q}) \oplus H_q(A^n_{1,q}) \oplus \beta_q.$$

An evaluator with final label A outputs $H_q(A) \oplus \mathsf{LSB}(A)C_q$. The extraction identity (6) makes the two extracted bits differ exactly at the target, so this m-bit correction assigns the payload there and gives zero elsewhere.

For a DCF, an equality label tracks whether the prefix still matches, and a comparison label for each payload bit records an already smaller prefix. Their inter-party differences encode $E_{q,i}\Delta$ and $L_{q,i,\ell}\Delta$, respectively. Starting from $E_{q,0} = 1$ and $L_{q,0,\ell} = 0$, the required recurrences are

$$\begin{aligned} E_{q,i} &= E_{q,i-1}[x_i = \alpha_{q,i}], \\ L_{q,i,\ell} &= L_{q,i-1,\ell} + E_{q,i-1}\beta_{q,\ell}[x_i < \alpha_{q,i}]. \end{aligned}$$

The first recurrence uses a ciphertext encoding the equality predicate. The second combines a label update using an encryption of one with the application of a ciphertext encrypting $\beta_{q,\ell}[d < \alpha_{q,i}]$ to the previous equality label. Its two contributions cannot both be one, so the difference remains in $\{0, \Delta\}$. Extracting the final comparison bits reconstructs $\beta_q[x < \alpha_q]$. Unlike the DPF, this representation needs no terminal output correction.

Sending these ciphertexts separately would replace the seed-correction cost with a ciphertext cost at every level. Instead, a single secret seed $s \in \{0,1\}^\lambda$ expands into a mask array indexed by the instance and bit position. We use $\mathsf{PRG}(s)$ with Nn output bits for DPF and $N(n+m)$ output bits for DCF: the former supplies the target masks $(r^{\alpha}_{q,i})_{q,i}$, while the latter also supplies payload masks $(r^{\beta}_{q,\ell})_{q,\ell}$. The generator includes the encrypted seed $\sigma = (\mathsf{Enc}_{\Delta}(s_h))_{h\in[\lambda]}$ and the masked targets

$$v^{\alpha}_{q,i} = \alpha_{q,i} \oplus r^{\alpha}_{q,i}.$$

Deterministic homomorphic evaluation of the same expansion circuit on σ recovers the encrypted masks $\widehat{r}^{\alpha}_{q,i}$, and public XOR gives $\widehat{\alpha}_{q,i} = \widehat{r}^{\alpha}_{q,i} \oplus v^{\alpha}_{q,i}$. This suffices to construct both branch predicates. For DCF, the same expansion also supplies a payload mask, giving $v^{\beta}_q = \beta_q \oplus r^{\beta}_q$; its encrypted bits recover encryptions of β_q and produce the comparison ciphertext CWs by homomorphic multiplication. The plaintext seed remains secret, since disclosing it together with the masked descriptions would reveal the functions. The expansion must fit the supported homomorphic depth. Every mask position, additive-map input, and output-map input is indexed by the function and its role.

Sharing the encrypted seed alone does not remove the cost of initial labels if fresh roots are stored for every function. Our batch extension also reuses one root $A^\star_b$ per evaluator, with $A^\star_1 - A^\star_0 = \Delta$. For DCF, a common baseline $B^\star$ initializes all comparison accumulators for both evaluators. Indexed additive terms distinguish the subsequent label updates; the root sharing introduces correlations that must be covered by joint privacy.

The complete batch key consists of a shared header and N instance records:

$$\begin{aligned}
K_b^{(N)} &= (h_b^T, R_1^T, \ldots, R_N^T),\\
h_b^{\mathsf{DPF}} &= (\mathsf{pp}, \sigma, \kappa, A_b^\star),\\
h_b^{\mathsf{DCF}} &= (\mathsf{pp}, \sigma, \kappa, A_b^\star, B^\star),\\
R_q^{\mathsf{DPF}} &= (v_q^\alpha, C_q),\\
R_q^{\mathsf{DCF}} &= (v_q^\alpha, v_q^\beta).
\end{aligned}$$

Here $\kappa \leftarrow \{0,1\}^\lambda$ is sampled once during generation and included in both batch headers. Both evaluators use the ring-valued keyed map PRF with κ and the same public indices to regenerate the common additive terms ρ locally; no array of ρ values is stored in the key. The key κ is public, and the privacy proof uses only deterministic regeneration of these terms. Each record has $n+m$ bits: an n-bit masked target and either an m-bit output correction or an m-bit masked payload. The public evaluation material, encrypted seed, additive-map key, and roots are charged once to each evaluator's batch key. Indexed evaluation uses the header and the selected record, reconstructing ciphertexts and intermediate labels locally. The algorithms in Figures 5 and 6 use exactly this representation. Appendix A gives the expanded specification and batch guarantees; Appendix A.7 records their $N = 1$ specializations. Two-party generation of batched DPF and DCF from XOR-shared inputs is given in Appendix A.8.

Both batch protocols support m-bit payloads and expose the indexed interface $\mathsf{Eval}(K_b^{(N)}, q, x)$. The DCF evaluates all payload components using one equality label and m comparison accumulators.

The following theorem states the reconstruction and complete key lengths under the label interface (5)–(6). Let $\varepsilon_{\mathsf{hom}}(\lambda)$ uniformly bound the probability of any reconstruction, label-update, or output-extraction failure in generation or either evaluator's computation for an indexed instance, over all supported batches and fixed inputs. This bound includes the DPF target path and all m DCF components. Let p bound the encoded public evaluation material, a the bit length of one encoded label, and c the bit length of one encrypted seed bit, using the fixed encoding with public lengths.

Theorem 5 (Batch DPF and DCF)**.** *Let $n, m, N \geq 1$. Under deterministic homomorphic reconstruction and the offset and extraction identities* (5)–(6) *for every reachable label and ciphertext, the protocols in Figures 5 and 6 correctly reconstruct all N respective point or comparison function outputs, for every fixed tuple of inputs, with probability at least $1 - N\varepsilon_{\mathsf{hom}}(\lambda)$. Their complete key lengths satisfy*

$$|K_{b,\mathsf{DPF}}^{(N)}| \leq N(n+m) + H_{\mathsf{DPF}}, \tag{7}$$

$$H_{\mathsf{DPF}} = p + a + \lambda c + \lambda,$$

$$|K_{b,\mathsf{DCF}}^{(N)}| \leq N(n+m) + H_{\mathsf{DCF}}, \tag{8}$$

$$H_{\mathsf{DCF}} = p + 2a + \lambda c + \lambda.$$

The proof is given in Appendix A.4.

Dividing the total key lengths by N gives

$$\frac{|K_{b,T}^{(N)}|}{N} \leq n + m + \frac{H_T}{N}.$$

The quantities p, a, c may depend on the public batch parameters. A uniform header bound $H_T(\lambda, N)$ independent of (n, m) gives the overhead bound $S(\lambda, N) = H_T(\lambda, N)/N$ in Definition 1. To achieve our compactness objective, this header bound must additionally be sublinear in N for fixed λ. An N-independent header gives

Protocol Π_{cBDPF}

Use the SWHE label interface (5)–(6), a low-depth PRG, and a ring-valued PRF. Set $\mathtt{tag} = (\mathsf{DPF}, n, m, N)$ and let $H_q(A) = H(\mathtt{tag}, \mathtt{out}, q, A)$ output m bits.

$\mathsf{Gen}(1^\lambda, (\alpha_q, \beta_q)_{q\in[N]})$

1. Run $(\mathsf{pp}, \Delta) \leftarrow \mathsf{SWHE.Gen}(1^\lambda)$. Sample $s, \kappa \leftarrow \{0,1\}^\lambda$ and encrypt the seed once: $\sigma_h \leftarrow \mathsf{Enc}_\Delta(s_h)$, $h \in [\lambda]$.
2. Sample $A_0^\star \leftarrow R_p$ and set $A_1^\star = A_0^\star + \Delta$. These roots are shared across all N instances.
3. Compute $(r^\alpha_{q,i})_{q,i} \leftarrow \mathsf{PRG}(s)$ and set $v^\alpha_{q,i} = \alpha_{q,i} \oplus r^\alpha_{q,i}$. Homomorphically evaluate the same PRG on σ to obtain $\widehat{r}^\alpha_{q,i}$ and reconstruct
$$\widehat{\alpha}_{q,i} = \widehat{r}^\alpha_{q,i} \oplus v^\alpha_{q,i},$$
$$\tau_{q,i,0} = 1 - \widehat{\alpha}_{q,i}, \quad \tau_{q,i,1} = \widehat{\alpha}_{q,i}.$$
4. For each q, initialize $A^0_{b,q} = A^\star_b$ for both b. For $i = 1, \ldots, n$, compute
$$\rho_{q,i,\alpha_{q,i}} = \mathsf{PRF}(\kappa, \mathtt{tag}, \mathtt{dpf}, q, i, \alpha_{q,i})$$
and update $A^i_{b,q} = \mathsf{Dec}(A^{i-1}_{b,q}, \tau_{q,i,\alpha_{q,i}}) + \rho_{q,i,\alpha_{q,i}}$ for both evaluators. Set $C_q = H_q(A^n_{0,q}) \oplus H_q(A^n_{1,q}) \oplus \beta_q$.
5. Set $h_b = (\mathsf{pp}, \sigma, \kappa, A^\star_b)$ and $R_q = (v^\alpha_q, C_q)$. Output $K^{(N)}_b = (h_b, R_1, \ldots, R_N)$ for $b \in \{0,1\}$.

$\mathsf{Eval}(K^{(N)}_b, q, x)$

1. Parse h_b, R_q. From σ and v^α_q, reconstruct the indexed ciphertext CWs as in generation. Initialize $A \leftarrow A^\star_b$.
2. For $i = 1, \ldots, n$, set $d = x_i$, compute
$$\rho_{q,i,d} = \mathsf{PRF}(\kappa, \mathtt{tag}, \mathtt{dpf}, q, i, d),$$
and update $A \leftarrow \mathsf{Dec}(A, \tau_{q,i,d}) + \rho_{q,i,d}$.
3. Output $y_b = H_q(A) \oplus \mathsf{LSB}(A) C_q$.

Figure 5: Batch DPF with a shared encrypted seed and roots. Indexed evaluation reconstructs $\beta_q[x = \alpha_q]$.

the particular overhead $H_T(\lambda)/N$. Circuit descriptions and evaluation keys are included in p, so their growth also enters this calculation.

Compression reduces stored material while retaining local reconstruction work. Evaluation still follows n sequential levels; DCF uses m comparison accumulators, and DPF generation still traverses the target path to compute C_q.

Privacy concerns the entire batch key, including the shared roots, encrypted seed, and all instance records. We assume that PRG jointly expands its seed pseudorandomly and that SWHE has the IND-CPA security specified in Section 2.4. For DPF, we additionally require that recovering Δ from (pp, s, σ) be hard even when the seed s is known; Appendix A.5 gives the formal recovery experiment. We model the domain-separated DPF output map H as a random oracle independent of setup and the other maps. The DPF target-path label relations of Theorem 5 must also hold except with the stated negligible failure probability. These assumptions yield the following joint guarantee.

Theorem 6 (Batch key privacy)**.** *Let $n, m, N \geq 1$ be polynomially bounded in λ. Under the assumptions above, the protocols in Figures 5 and 6 satisfy joint batch-key privacy against static corruption of either evaluator, with leakage (T, n, m, N). A simulator given $(1^\lambda, b, T, n, m, N)$ produces a complete batch key computationally indistinguishable from an honestly generated key. The DPF guarantee holds against classical adversaries*

Protocol Π_{cBDCF}

Use the SWHE label interface (5)–(6), a low-depth PRG, and a ring-valued PRF. Set $\mathsf{tag} = (\mathsf{DCF}, n, m, N)$.

$\mathsf{Gen}(1^\lambda, (\alpha_q, \beta_q)_{q\in[N]})$

1. Run $(\mathsf{pp}, \Delta) \leftarrow \mathsf{SWHE.Gen}(1^\lambda)$. Sample $s, \kappa \leftarrow \{0,1\}^\lambda$ and set $\sigma_h \leftarrow \mathsf{Enc}_\Delta(s_h)$ for $h \in [\lambda]$.
2. Sample $A_0^\star, B^\star \leftarrow R_p$ independently and set $A_1^\star = A_0^\star + \Delta$. All instances and payload accumulators use these shared roots.
3. Compute the target-mask and payload-mask arrays
$$\big((r^\alpha_{q,i})_{q,i}, (r^\beta_{q,\ell})_{q,\ell}\big) \leftarrow \mathsf{PRG}(s).$$
Set
$$v^\alpha_{q,i} = \alpha_{q,i} \oplus r^\alpha_{q,i}, \qquad v^\beta_{q,\ell} = \beta_{q,\ell} \oplus r^\beta_{q,\ell}.$$
4. Set $h_b = (\mathsf{pp}, \sigma, \kappa, A_b^\star, B^\star)$ and $R_q = (v^\alpha_q, v^\beta_q)$. Output $K_b^{(N)} = (h_b, R_1, \ldots, R_N)$ for both evaluators.

$\mathsf{Eval}(K_b^{(N)}, q, x)$

1. Parse h_b, R_q. Homomorphically expand σ to the mask slice for q and recover
$$\widehat{\alpha}_{q,i} = \widehat{r}^\alpha_{q,i} \oplus v^\alpha_{q,i}, \qquad \widehat{\beta}_{q,\ell} = \widehat{r}^\beta_{q,\ell} \oplus v^\beta_{q,\ell}.$$
2. Set $\tau^0_q = \widehat{\beta}_{q,1} \oplus \widehat{\beta}_{q,1}$ and $\tau^{\mathsf{id}}_q = 1 - \tau^0_q$. For every i, ℓ, reconstruct
$$\tau^{=}_{q,i,0} = 1 - \widehat{\alpha}_{q,i}, \qquad \tau^{=}_{q,i,1} = \widehat{\alpha}_{q,i},$$
$$\tau^{<}_{q,i,\ell,0} = \widehat{\alpha}_{q,i}\widehat{\beta}_{q,\ell}, \qquad \tau^{<}_{q,i,\ell,1} = \tau^0_q.$$
3. Initialize $A \leftarrow A_b^\star$ and $B_\ell \leftarrow B^\star$ for every $\ell \in [m]$.
4. For $i = 1, \ldots, n$, set $d = x_i$ and compute
$$\rho^A_{q,i,d} = \mathsf{PRF}(\kappa, \mathsf{tag}, \mathtt{eq}, q, i, d),$$
$$\rho^B_{q,\ell,i,d} = \mathsf{PRF}(\kappa, \mathsf{tag}, \mathtt{lt}, q, \ell, i, d)$$
for all $\ell \in [m]$. Save $A_{\mathsf{old}} \leftarrow A$ and update
$$A \leftarrow \mathsf{Dec}(A_{\mathsf{old}}, \tau^{=}_{q,i,d}) + \rho^A_{q,i,d},$$
$$B_\ell \leftarrow \mathsf{Dec}(B_\ell, \tau^{\mathsf{id}}_q) + \mathsf{Dec}(A_{\mathsf{old}}, \tau^{<}_{q,i,\ell,d}) + \rho^B_{q,\ell,i,d}$$
for all $\ell \in [m]$, using the same saved equality label.
5. Output $y_b = (\mathsf{LSB}(B_1), \ldots, \mathsf{LSB}(B_m))$.

Figure 6: Batch DCF for general m-bit payloads. Indexed evaluation reconstructs $\beta_q[x < \alpha_q]$.

in the stated random-oracle model; the DCF guarantee requires only the stated encryption and PRG assumptions.

The proof is given in Appendix A.5.

Combining reconstruction and privacy with a uniform header bound gives the BFSS realization below.

Corollary 1 (BFSS realization and amortized size)**.** *Let $n, m, N \geq 1$ be polynomially bounded in λ. Assume the hypotheses of Theorems 5 and 6, with negligible $\varepsilon_{\mathsf{hom}}(\lambda)$ and polynomial-time setup and evaluation. If*

the shared header admits a uniform bound $H_T(\lambda, N)$ independent of (n, m), the protocols in Figures 5 and 6 realize BDPF and BDCF, respectively, in their stated security models, with amortized key lengths at most

$$n + m + S(\lambda, N), \qquad S(\lambda, N) = \frac{H_T(\lambda, N)}{N}.$$

If additionally $H_T(\lambda, N) = o(N)$ for fixed λ, these amortized key lengths approach $n + m$.

The proof is given in Appendix A.6.

The proof of Theorem 6 treats the additive-map key κ as public. It uses encryption security to replace the encrypted seed before replacing its expanded masks, and separately hides the DPF output corrections. Thus it covers all shared and instance-specific key material together, rather than relying on marginal privacy of individual records.

5 Related Work

DPF and DCF constructions Gilboa and Ishai introduced distributed point functions (DPFs) [16]. Boyle, Gilboa, and Ishai generalized this viewpoint to function secret sharing (FSS) and subsequently improved the concrete efficiency of DPFs and extended the supported function families [6, 7]. These works established the tree-based constructions underlying efficient point and comparison sharing. Later improvements address both representation and computation: Boyle et al. reduce DCF key sizes and develop FSS for mixed-mode and fixed-point operations [3], while Half-Tree reduces the cost of GGM-tree expansion and distributed DPF/DCF generation [18]. We build on this line of work by prescribing target-path states before CW generation and using the Half-Tree correlation to share corrections across branches. Our analysis separates generation depth from evaluation cost and stored key size.

Distributed and programmable key generation Floram develops distributed DPF generation for secure RAM computation, moving PRG evaluations outside the secure computation through an iterative protocol [15]. Xing et al. give distributed generation protocols for DPFs and DCFs supporting arithmetic-shared inputs and outputs [29]. Other constructions change the setup structure. Programmable DPFs reuse an input-independent offline key; the same work also derives constant-round distributed DPF generation for polynomial-size domains [10]. Non-interactive DPFs allow parties to derive function shares from public keys encoding their private parameters, with a one-round simultaneous-message setup under standard public-key assumptions and public keys sublinear in the domain size [4]. These are important precedents for reducing interaction and reusing setup. Our prescribed-state result concerns the circuit depth of CW generation; communication and round complexity of a distributed realization remain separate measures.

Note on programmability Programmable DPFs fix one evaluator's complete key independently of the target function [10]. Our parallel construction preselects internal target-path states to eliminate dependencies between successive CWs, while retaining $O(n)$-step single-point evaluation on $\{0,1\}^n$. This is a guarantee about CW-generation circuit depth; programmability of a complete key and constant-round distributed generation require separate constructions.

Batched and multi-point FSS Piske, Singh, and Trieu introduced batched FSS and proposed an HSS-based framework for compressing DPF keys using a sparse-LPN variant and quadratic residuosity [24]. Their CFAIL 2026 version identifies a deterministic polynomial-time linear attack on the regular-noise LPN parameters used in the concrete instantiation, breaking the compressed-input indistinguishability on which its security relies [25, Sec. 1.3]. The authors leave a secure code and noise choice preserving the framework's efficiency

as an open problem. We give a formal BFSS model with joint privacy and complete key accounting, and secure constructions for both point and comparison functions under the assumptions in Section 4.4. The complete key bound $N(n+m)+H_T$ yields amortized size approaching $n+m$ when the shared header has a uniform sublinear bound.

Multi-point FSS provides another way to compress collections of point contributions. De Castro and Polychroniadou develop verifiable point and multi-point FSS, including packed verification and a method for packing DPFs into multi-point shares [14]. Boyle et al. give improved DMPF constructions and implementations for different parameter regimes, with applications to correlation generation and private set intersection [5]. For point functions, an indexed batch can itself be encoded as a DMPF on the enlarged domain of pairs (q,x), with support contained in $\{(q,\alpha_q)\}_{q\in[N]}$. DMPFs are therefore direct baselines for the point-function case, with their domain, support, and evaluation costs included in the comparison.

Extensions of function secret sharing Incremental DPFs enable secret sharing of labels along a tree path and support the prefix queries used in private heavy-hitter discovery [2]. PRGs with encoded-output homomorphism yield direct FSS constructions for bit-fixing predicates and branching programs, and a strengthened variant supports finite automata [11]. Streaming FSS supports key reuse for secure computation over successive encrypted messages [27]. These extensions broaden the functionality and reuse patterns available beyond individual point and comparison queries.

The party structure, privacy guarantee, and reconstruction rule also affect achievable key sizes. In the honest-majority setting, information-theoretic DPFs provide statistically private three-server and perfectly private four-server constructions [9]. Goel, Wang, and Wang obtain multiparty DPFs from one-way functions with share sizes polynomial in the number of parties [17]. Kumar, Patranabis, and Mukhopadhyay study compact multiparty sharing of point, comparison, and interval functions using a nonlinear decoder with an additional comparison operation [23]. These results make it important to state the reconstruction and corruption models when comparing compactness. Our batch model uses two evaluators, local evaluation, and XOR reconstruction.

FSS-based secure computation Boyle, Gilboa, and Ishai establish a general framework for secure computation with preprocessing via FSS, covering useful nonlinear gates and conversions between representations [8]. AriaNN develops FSS-based neural-network training and inference with a trusted dealer [26], while Pika supplies protocols for nonlinear functions over rings and techniques for malicious security with an honest majority [28]. Orca accelerates FSS-based training and inference with GPUs [22], and Sigma develops GPU-accelerated FSS protocols for transformer inference [20]. SHARK extends FSS-based inference to active two-party security using, among other techniques, interactive FSS [19]. These systems motivate efficient preprocessing and compact stored correlations. Our focus is on the underlying generation and representation costs; end-to-end MPC performance also depends on the application protocols and their security model.

6 Benchmark

Using our implementation [1], we evaluate how parallel correction-word generation and batch key compression affect interaction, computation, and storage. We first compare the parallel (PDPF/PDCF) and Half-Tree (HDPF/HDCF) constructions with the distributed FSS implementation of Xing et al. [29]. We then study batch generation using the direct-SIMD GPU implementation and an EzPC/LLAMA dealer reference [13].

Table 1: Two-party FSS base sweep. Key sizes are bytes per evaluator. Keygen is distributed for baseline/P/H; Eval is local per public point.

Construction	Rounds	Keygen (ms)			Eval (μs/pt)			Key size
		n=4	n=12	n=20	n=4	n=12	n=20	
Baseline DPF [29]	40–72	3.667	4.35	45.746	0.0958867	0.240168	0.375879	96–384
PDPF	15	1.141	2.343	3.688	0.0597773	0.215172	0.374121	280–1304
HDPF	13	0.867	0.9	0.969	0.090918	0.162711	0.29666	152–664
Baseline DCF [29]	216–1048	32.49	99.879	724.482	0.294078	0.803711	1.31953	96–384
PDCF	13	1.093	2.705	3.33	0.0857539	0.254332	0.383625	288–1376
HDCF	13	0.816	0.971	2	0.0735078	0.240398	0.415117	154–698

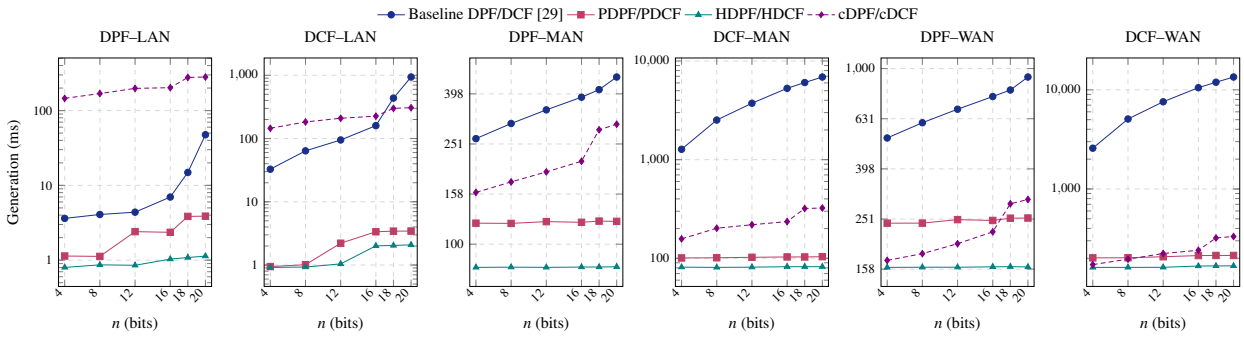

Figure 7: Two-party single-function FSS generation under LAN, MAN, and WAN. The cDPF/cDCF curves use $N = 1, m = 16$ and plot $T_{\text{Gen}} + T_{\text{Eval,off}}$, charging local evaluation preparation to generation. Shared Setup and public-header provisioning are excluded, as is online evaluation.

6.1 Experimental Setup

The distributed baseline and P/H experiments run on an AMD EPYC 9754 server with 1.5 TiB of memory and Ubuntu 22.04.3 LTS. Two evaluator processes communicate over TCP on loopback, with bandwidth and delay controlled by Linux `tc netem`. LAN, MAN, WAN1, and WAN2 use 1 Gbps/0.05 ms, 200 Mbps/10 ms, 40 Mbps/20 ms, and 1 Mbps/20 ms, respectively. WAN1 is labeled WAN in the single-function plots. Delays are configured per-packet values, rather than measured round-trip times. We sweep $n \in \{4, 8, 12, 16, 18, 20\}$ with a 16-bit payload. The C++17 implementation is built with Clang 14.0.0 in Release mode and uses one worker per evaluator. Symmetric operations use 128-bit blocks. The batched FSS prototype uses $R_q = \mathbb{Z}_q[X]/(X^{1024}+1)$, $p = 2^{32}$, $q = p \cdot 536881153$, and a binary gadget with $\ell = 62$ (a 124×124 matrix).

The GPU batch implementation runs on NVIDIA H800 GPUs and uses a 256-bit seed and eight concurrent shards, each processing $N/8$ functions; its backend options are detailed in Appendix C. EzPC/LLAMA uses the same network profiles, input lengths, payload length, and batch sizes as BatchFSS, with concurrent key delivery to two receivers. The SWHE backend parameters are implementation settings without a validated lattice-security estimate.

Distributed Keygen measures interactive generation between evaluators. For batched compressed FSS, Setup includes generation and distribution of the shared header. BatchFSS Gen includes function-dependent key generation, distribution of the key records, and evaluation preparation charged to generation, including DCF offline work. EzPC/LLAMA Gen+send includes dealer generation, serialization, concurrent delivery of both key batches, and receipt acknowledgments. Local Eval measures one public query and excludes output-share reconstruction.

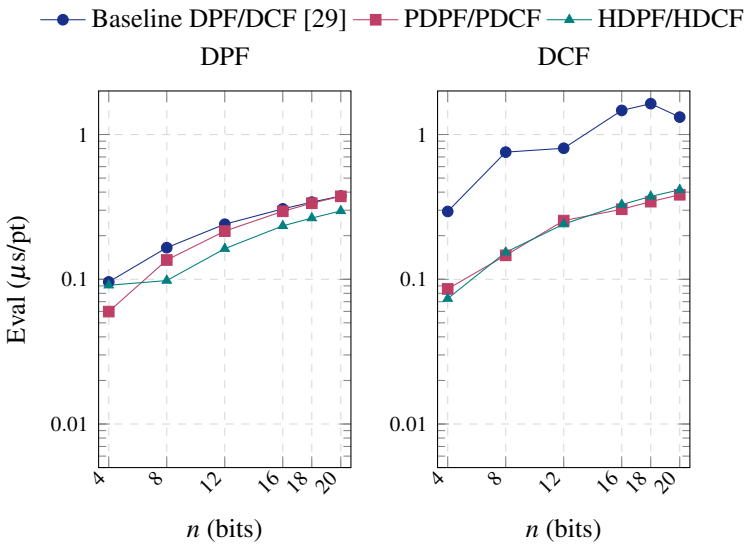

Figure 8: Two-party FSS evaluation. Each panel shows network-free base-sweep measurements excluding output-share opening.

6.2 Parallel and Half-Tree FSS

Table 1 reports the two-party LAN base sweep: communication rounds, distributed Keygen, local per-point Eval, and key bytes per evaluator. Figure 7 reports a separate network sweep, including the single-function compressed variants.

Interaction and key storage. As n increases from 4 to 20, the baseline requires 40–72 communication rounds for DPF and 216–1048 for DCF. PDPF uses 15 rounds, while PDCF, HDPF, and HDCF use 13 throughout the sweep. Prescribing the target-path states removes the dependency between correction words at successive levels, allowing their generation to proceed in parallel. Half-Tree further shares corrections between the left and right expansions. At $n = 20$, HDPF and HDCF keys occupy 664 and 698 bytes per evaluator, approximately half the 1304- and 1376-byte PDPF and PDCF keys. The baseline keys remain smaller at 384 bytes; P/H trade additional key storage for fewer sequential interactions.

Generation cost. In the LAN base sweep of Table 1, DPF generation at $n = 20$ falls from 45.746 ms for the baseline to 3.688 ms for PDPF and 0.969 ms for HDPF, reductions of 12.4× and 47.2×. DCF generation falls from 724.482 ms to 3.330 ms for PDCF and 2.000 ms for HDCF, reductions of 217.6× and 362.2×. Thus, Half-Tree reduces both key storage and generation time relative to the parallel construction in this sweep.

Figure 7 shows how this interaction reduction affects sensitivity to network delay. For example, HDPF generation changes from 80.782 to 81.250 ms in MAN and from 161.087 to 161.758 ms in WAN as n grows from 4 to 20. In contrast, baseline DCF generation in WAN rises from 2.579 to 13.468 s. At $n = 20$, HDPF and HDCF take 161.758 and 167.622 ms in WAN, compared with 925.387 and 13,467.8 ms for the respective baselines, yielding reductions of 5.7× and 80.3×. These near-flat latencies reflect a fixed number of sequential exchanges; additional levels still increase local expansion work and correction bytes.

The cDPF/cDCF curves in the same figure use $N = 1, m = 16$ and charge offline evaluation preparation to generation, plotting $T_{\text{Gen}} + T_{\text{Eval,off}}$. At $n = 20$, these costs are 281.036/307.294 ms for cDPF/cDCF in LAN and 300.502/331.394 ms in WAN. The archived summaries contain five runs per configuration and retain an unverified source status; these points characterize the reported single-function preparation cost. They do not benefit from batch amortization.

Local evaluation. Figure 8 plots Eval from the base sweep. Evaluation still traverses a depth-n path, so

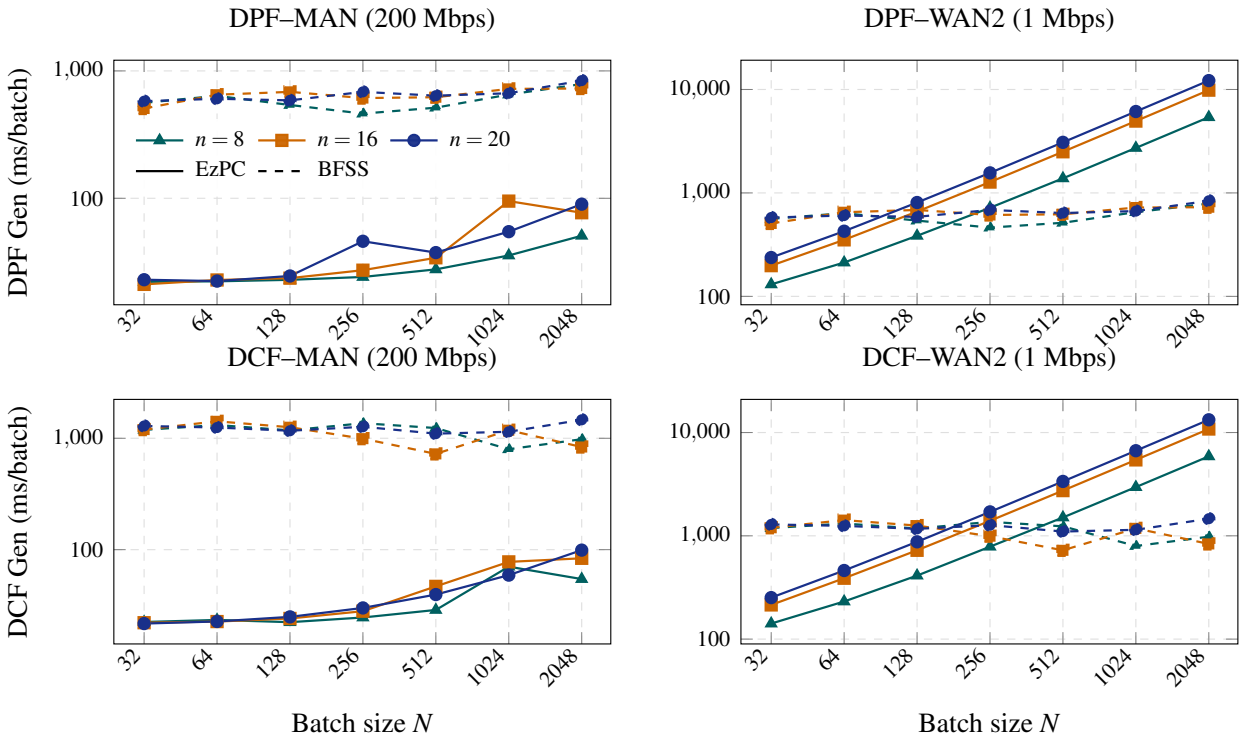

Figure 9: BatchFSS and EzPC/LLAMA for $n \in \{8, 16, 20\}$ and $m = 16$ under matched MAN and WAN2 settings. Colors and markers identify n; solid curves show EzPC/LLAMA Gen+send and dashed curves show BatchFSS Gen. Times are totals per batch. BatchFSS Gen includes key-record delivery and DCF offline preparation; shared Setup and online Eval are excluded from both plotted metrics.

parallel generation does not remove its sequential tree work. At $n = 20$, HDPF takes 0.297 μs per point, compared with 0.376 μs for the baseline DPF. PDCF and HDCF take 0.384 and 0.415 μs, compared with 1.320 μs for baseline DCF. HDCF is slightly slower than PDCF at this input length: its key-size and generation benefits do not imply an evaluation advantage.

6.3 Batched Compressed FSS

The batch sweep varies $N \in \{32, 64, 128, 256, 512, 1024, 2048\}$ using the input lengths and payload specified in Section 6.1. Figure 9 overlays $n \in \{8, 16, 20\}$ under MAN and WAN2; additional phase measurements are reported in Appendix C.

Shared Setup takes approximately 2.7 s on average (LAN). Within each repetition, a phase's wall time is the maximum over the concurrent shards; each BatchFSS point is the median of three repetitions. EzPC/LLAMA points are medians of five repetitions after one warmup.

Our batch representation amortizes the shared header across N functions, with an $(n+m)$-bit record for each function. The encrypted seed and roots are transmitted once per batch, and evaluators reconstruct encrypted correction words locally. Batching therefore reduces the key material to distribute while introducing local reconstruction and preparation work. Its performance benefit depends on whether this computation or conventional-key communication is the larger cost.

Figure 9 shows how this tradeoff changes with batch size. At $n = 20$, increasing N from 32 to 2048 expands the batch by 64×, but increases BatchFSS DPF Gen and DCF Gen/preparation by only 1.45× and 1.14×, respectively. Over the same sweep, EzPC/LLAMA Gen+send in WAN2 grows by 51.3× and 52.7×.

Across the three plotted input lengths at $N = 2048$, BatchFSS Gen remains within 727.7–839.4 ms for DPF and 834.8–1465.5 ms for DCF. These endpoints show that its generation-and-delivery cost grows much more slowly than the batch, although individual measurements are nonmonotonic and do not establish a smooth scaling law.

In MAN, the performance gap is dominated by BatchFSS's local computation. At $n = 20, N = 2048$, its DPF and DCF Gen costs, including key-record delivery, are 839.4 and 1465.5 ms, whereas EzPC/LLAMA completes Gen+send in 89.9 and 99.0 ms. At 200 Mbps, distributing conventional keys is relatively inexpensive compared with compressed-key preparation, so the additional computation outweighs the communication savings.

In WAN2, communication dominates the conventional pipeline as the batch grows. At $n = 20, N = 2048$, EzPC/LLAMA's local DPF and DCF generation takes only 8.6 and 12.0 ms, but Gen+send reaches 12.186 and 13.292 s under the 1 Mbps cap. BatchFSS therefore reduces the corresponding generation-and-delivery times by factors of $14.5\times$ and $9.1\times$, respectively. In the sampled $n = 20$ configurations, BatchFSS first has a lower generation-and-delivery cost at $N = 128$ for DPF and $N = 256$ for DCF, and retains this advantage at all larger sampled batch sizes. This shift from computation to communication as the larger cost explains where batching is most useful: shared key material avoids repeating substantial correction-word communication for every function, while the measured generation-and-delivery cost grows modestly over the batch range.

Both plotted metrics include function-dependent generation and key-record distribution; shared Setup and online Eval are excluded. Table 2 reports Online Eval separately.

References

[1] Our implementation code. `https://anonymous.4open.science/r/bitfss2027/`.

[2] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy*, pages 762–776, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press. `doi:10.1109/SP40001.2021.00048`.

[3] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 871–900, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland. `doi:10.1007/978-3-030-77886-6_30`.

[4] Elette Boyle, Lalita Devadas, and Sacha Servan-Schreiber. Non-interactive distributed point functions. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025: 28th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 15674 of *Lecture Notes in Computer Science*, pages 3–35, Røros, Norway, May 12–15, 2025. Springer, Cham, Switzerland. `doi:10.1007/978-3-031-91820-9_1`.

[5] Elette Boyle, Niv Gilboa, Matan Hamilis, Yuval Ishai, and Yaxin Tu. Improved constructions for distributed multi-point functions. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 2414–2432, San Francisco, CA, USA, May 12–15, 2025. IEEE Computer Society Press. `doi:10.1109/SP61157.2025.00044`.

[6] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Advances in Cryptology – EUROCRYPT 2015*, pages 337–367, 2015. `doi:10.1007/978-3-662-46803-6_12`.

[7] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303, 2016. doi:10.1145/2976749.2978429.

[8] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 341–371, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-36030-6_14.

[9] Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Information-theoretic distributed point functions. In Dana Dachman-Soled, editor, *ITC 2022: 3rd Conference on Information-Theoretic Cryptography*, volume 230 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:14, Cambridge, MA, USA, July 5–7, 2022. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITC.2022.17.

[10] Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Programmable distributed point functions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 121–151, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-15985-5_5.

[11] Elette Boyle, Lisa Kohl, Zhe Li, and Peter Scholl. Direct FSS constructions for branching programs and more from PRGs with encoded-output homomorphism. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024, Part VI*, volume 15489 of *Lecture Notes in Computer Science*, pages 266–298, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore. doi:10.1007/978-981-96-0938-3_9.

[12] A. Brüggemann, R. Hundt, T. Schneider, A. Suresh, and H. Yalame. Flute: Fast and secure lookup table evaluations. In *S&P*, 2023.

[13] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. EzPC: Programmable and efficient secure two-party computation for machine learning. In *2019 IEEE European Symposium on Security and Privacy*, pages 496–511, Stockholm, Sweden, June 17–19, 2019. IEEE Computer Society Press. doi:10.1109/EuroSP.2019.00043.

[14] Leo de Castro and Antigoni Polychroniadou. Lightweight, maliciously secure verifiable function secret sharing. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 150–179, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-06944-4_6.

[15] Jack Doerner and abhi shelat. Scaling ORAM for secure computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 523–535, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. doi:10.1145/3133956.3133967.

[16] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 640–658, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-55220-5_35.

[17] Aarushi Goel, Mingyuan Wang, and Zhiheng Wang. Multiparty distributed point functions. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part IV*, volume 16003 of *Lecture Notes in Computer Science*, pages 140–173, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland. doi:10.1007/978-3-032-01884-7_5.

[18] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. Half-tree: Halving the cost of tree expansion in COT and DPF. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part I*, volume 14004 of *Lecture Notes in Computer Science*, pages 330–362, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-30545-0_12.

[19] Kanav Gupta, Nishanth Chandran, Divya Gupta, Jonathan Katz, and Rahul Sharma. SHARK: Actively secure inference using function secret sharing. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 2472–2490, San Francisco, CA, USA, May 12–15, 2025. IEEE Computer Society Press. doi:10.1109/SP61157.2025.00175.

[20] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. SIGMA: Secure GPT inference with function secret sharing. *Proceedings on Privacy Enhancing Technologies*, 2024(4):61–79, October 2024. doi:10.56553/popets-2024-0107.

[21] Shai Halevi. Homomorphic encryption, 2017. Tutorial, Sections 2.2.3–2.2.4. URL: https://shaih.github.io/pubs/he-chapter.pdf.

[22] Neha Jawalkar, Kanav Gupta, Arkaprava Basu, Nishanth Chandran, Divya Gupta, and Rahul Sharma. Orca: Fss-based secure training and inference with gpus. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 597–616, 2024. doi:10.1109/SP54263.2024.00063.

[23] Chandan Kumar, Sikhar Patranabis, and Debdeep Mukhopadhyay. Compact key function secret sharing with non-linear decoder. *IACR Communications in Cryptology (CiC)*, 1(2):8, 2024. doi:10.62056/a3c3c3w9p.

[24] Lucas Piske, Jaspal Singh, and Ni Trieu. Batched distributed point function from sparse LPN and homomorphic secret sharing. Cryptology ePrint Archive, Report 2024/823, 2024. URL: https://eprint.iacr.org/2024/823.

[25] Lucas Piske, Jaspal Singh, and Ni Trieu. Batched distributed point function from sparse LPN and homomorphic secret sharing. CFAIL 2026, 2026. Workshop version. URL: https://www.cfail.org/_files/ugd/6a3e24_771bad4dac054a4092c9b18ff1d57d56.pdf.

[26] Théo Ryffel, Pierre Tholoniat, David Pointcheval, and Francis R. Bach. AriaNN: Low-interaction privacy-preserving deep learning via function secret sharing. *Proceedings on Privacy Enhancing Technologies*, 2022(1):291–316, January 2022. doi:10.2478/popets-2022-0015.

[27] Xiangfu Song, Jianli Bai, Ye Dong, Yijian Liu, Yu Zhang, Xianhui Lu, and Tianwei Zhang. Streaming function secret sharing and its applications. In *35th USENIX Security Symposium (USENIX Security 26)*, pages 6445–6464. USENIX Association, 2026. URL: https://www.usenix.org/system/files/conference/usenixsecurity26/sec26_prepub_song-xiangfu.pdf.

[28] Sameer Wagh. Pika: Secure computation using function secret sharing over rings. *Proceedings on Privacy Enhancing Technologies*, 2022(4):351–377, October 2022. doi:10.56553/popets-2022-0113.

[29] Pengzhi Xing, Hongwei Li, Meng Hao, Hanxiao Chen, Jia Hu, and Dongxiao Liu. Distributed function secret sharing and applications. In *ISOC Network and Distributed System Security Symposium – NDSS 2025*, San Diego, CA, USA, February 24–28, 2025. The Internet Society.

[30] Lijing Zhou, Ziyu Wang, Hongrui Cui, Qingrui Song, and Yu Yu. Bicoptor: Two-round secure three-party non-linear computation without preprocessing for privacy-preserving machine learning. In *S&P*, 2023.

A Batch Protocols with Encrypted Correction Words

This appendix expands the batch protocols in Figures 5 and 6 and proves their correctness, complete key bounds, and joint privacy. We use the SWHE/label interface of Section 4.4 and the notation of Section 2.1. All protocols support arbitrary m-bit payloads. Appendix A.8 additionally describes two-party generation from XOR-shared targets and payloads, using a terminal-payload DPF variant that avoids target-path computation during generation.

A.1 Shared Setup and Indexed Expansion

Fix an unambiguous encoding of map inputs and include $\mathsf{tag} = (T, n, m, N)$ in every additive-map and output-map domain. The instance index q and the map-role tag separate their respective inputs. Each batch uses fresh setup randomness, a fresh mask seed, and fresh root labels. The additive-map key κ is public and permits deterministic regeneration of the common terms ρ; its secrecy is not used.

Mask expansion interface The expansion $\mathsf{PRG}(s)$ has Nn output bits for DPF and $N(n+m)$ for DCF. Parse target masks $(r^{\alpha}_{q,i})$ in lexicographic (q,i) order, followed for DCF by payload masks $(r^{\beta}_{q,\ell})$ in lexicographic (q,ℓ) order. Pseudorandomness is required jointly for this entire array.

The plaintext expansion and its deterministic homomorphic evaluation use the same circuit and output ordering.

The supported homomorphic depth includes the expansion and the subsequent predicate gates, including DCF products. Circuit descriptions, evaluation keys, and required auxiliary tables are included in pp and charged to the shared header. Homomorphic reconstruction yields the encrypted masks; the masked record then gives

$$\widehat{\alpha}_{q,i} = \widehat{r}^{\alpha}_{q,i} \oplus v^{\alpha}_{q,i}, \qquad \widehat{\beta}_{q,\ell} = \widehat{r}^{\beta}_{q,\ell} \oplus v^{\beta}_{q,\ell},$$

where the payload equation is used only for DCF.

Shared setup for one batch The generator performs the following steps once for the entire batch.

1. Run $(\mathsf{pp}, \Delta) \leftarrow \mathsf{SWHE.Gen}(1^{\lambda})$ with the offset and extraction interface (5)–(6). Sample independent $s, \kappa \leftarrow \{0,1\}^{\lambda}$.

2. Encrypt the mask seed once, setting $\sigma_h \leftarrow \mathsf{Enc}_{\Delta}(s_h)$ for $h \in [\lambda]$. Expand s into the mask array. For DPF generation, also homomorphically expand σ into the corresponding encrypted array; DCF generation does not need these ciphertexts until local evaluation. Neither s nor Δ is included in an evaluator's key.

3. Sample $A^{\star}_0 \leftarrow R_p$ and set $A^{\star}_1 = A^{\star}_0 + \Delta$. These two roots are reused for all instances of this batch. For DCF also sample $B^{\star} \leftarrow R_p$, independently of $A^{\star}_0$; both evaluators use this same baseline for every comparison accumulator.

The resulting headers are

$$h_b^{\mathsf{DPF}} = (\mathsf{pp}, \sigma, \kappa, A_b^\star),$$
$$h_b^{\mathsf{DCF}} = (\mathsf{pp}, \sigma, \kappa, A_b^\star, B^\star).$$

The roots are shared within this batch, so the privacy proof must cover their correlations with all instance records.

A.2 Batch DPF Protocol

$\mathsf{BatchGen}_{\mathsf{DPF}}$ On input $(1^\lambda, (\alpha_q, \beta_q)_{q\in[N]})$, run the shared DPF setup above and perform the following steps for each q.

1. Form $v_{q,i}^\alpha = \alpha_{q,i} \oplus r_{q,i}^\alpha$ for $i \in [n]$. Reconstruct $\widehat{\alpha}_{q,i}$ and the ciphertext CWs

   $$\tau_{q,i,0} = 1 - \widehat{\alpha}_{q,i}, \qquad \tau_{q,i,1} = \widehat{\alpha}_{q,i}.$$

   Thus $\tau_{q,i,d}$ encrypts $[d = \alpha_{q,i}]$ for branch $d \in \{0,1\}$.

2. Define the public common additive terms

   $$\rho_{q,i,d} = \mathsf{PRF}(\kappa, \mathsf{tag}, \mathtt{dpf}, q, i, d).$$

   Their outputs are encoded in R_p. Starting from $A_{b,q}^0 = A_b^\star$, follow the target path for each b:

   $$A_{b,q}^i = \mathsf{Dec}(A_{b,q}^{i-1}, \tau_{q,i,\alpha_{q,i}}) + \rho_{q,i,\alpha_{q,i}}.$$

   The same ρ is added for both evaluators.

3. Let $H_q(A) = H(\mathsf{tag}, \mathtt{out}, q, A)$ be an m-bit deterministic output map. Compute

   $$C_q = H_q(A_{0,q}^n) \oplus H_q(A_{1,q}^n) \oplus \beta_q$$

   and set $R_q^{\mathsf{DPF}} = (v_q^\alpha, C_q)$.

Output

$$K_b^{(N)} = (h_b^{\mathsf{DPF}}, R_1^{\mathsf{DPF}}, \ldots, R_N^{\mathsf{DPF}}).$$

The ciphertext CWs and target-path labels are not included in the key. Generation of C_q still requires an n-step target-path traversal for each instance, although different instances can be processed in parallel.

$\mathsf{BatchEval}_{\mathsf{DPF}}(K_b^{(N)}, q, x)$

1. Parse the shared header and record R_q^{DPF}. Homomorphically reconstruct the encrypted mask slice for q from σ, then obtain $\tau_{q,i,0}, \tau_{q,i,1}$ using v_q^α as in generation. Initialize $A \leftarrow A_b^\star$.

2. For $i = 1, \ldots, n$, compute the same ρ_{q,i,x_i} as generation and update

   $$A \leftarrow \mathsf{Dec}(A, \tau_{q,i,x_i}) + \rho_{q,i,x_i}.$$

3. Output $y_b = H_q(A) \oplus \mathsf{LSB}(A) C_q$.

Evaluation uses the shared header and the selected record. Any locally reconstructed ciphertext cache is working memory.

A.3 Batch DCF Protocol

The DCF uses one equality label and m comparison labels to reconstruct $\beta_q[x < \alpha_q]$.

$\mathsf{BatchGen}_{\mathsf{DCF}}$ On input $(1^\lambda, (\alpha_q, \beta_q)_{q\in[N]})$, run the shared DCF setup and, for each q, form

$$v^{\alpha}_{q,i} = \alpha_{q,i} \oplus r^{\alpha}_{q,i}, \quad i \in [n],$$
$$v^{\beta}_{q,\ell} = \beta_{q,\ell} \oplus r^{\beta}_{q,\ell}, \quad \ell \in [m].$$

Set $R^{\mathsf{DCF}}_q = (v^{\alpha}_q, v^{\beta}_q)$ and output

$$K^{(N)}_b = (h^{\mathsf{DCF}}_b, R^{\mathsf{DCF}}_1, \ldots, R^{\mathsf{DCF}}_N).$$

Generation does not traverse a target path.

$\mathsf{BatchEval}_{\mathsf{DCF}}(K^{(N)}_b, q, x)$ From σ and R^{DCF}_q, reconstruct $\widehat{\alpha}_{q,i}$ and $\widehat{\beta}_{q,\ell}$ as in Appendix A.1. Define $\tau^0_q = \widehat{\beta}_{q,1} \oplus \widehat{\beta}_{q,1}$ and $\tau^{\mathsf{id}}_q = 1 - \tau^0_q$. For each i and ℓ, reconstruct

$$\tau^{=}_{q,i,0} = 1 - \widehat{\alpha}_{q,i}, \qquad \tau^{=}_{q,i,1} = \widehat{\alpha}_{q,i},$$
$$\tau^{<}_{q,i,\ell,0} = \widehat{\alpha}_{q,i}\widehat{\beta}_{q,\ell}, \qquad \tau^{<}_{q,i,\ell,1} = \tau^0_q.$$

The ciphertexts $\tau^{=}_{q,i,d}$, $\tau^{<}_{q,i,\ell,d}$, and τ^{id}_q encrypt $[d = \alpha_{q,i}]$, $\beta_{q,\ell}[d < \alpha_{q,i}]$, and one, respectively.

1. Initialize $A \leftarrow A^\star_b$ and $B_\ell \leftarrow B^\star$ for all $\ell \in [m]$.

2. For $i = 1, \ldots, n$, set $d = x_i$ and compute

$$\rho^{A}_{q,i,d} = \mathsf{PRF}(\kappa, \mathsf{tag}, \mathsf{eq}, q, i, d),$$
$$\rho^{B}_{q,\ell,i,d} = \mathsf{PRF}(\kappa, \mathsf{tag}, \mathtt{lt}, q, \ell, i, d).$$

   Save $A_{\mathsf{old}} \leftarrow A$ and update

$$A \leftarrow \mathsf{Dec}(A_{\mathsf{old}}, \tau^{=}_{q,i,d}) + \rho^{A}_{q,i,d},$$
$$B_\ell \leftarrow \mathsf{Dec}(B_\ell, \tau^{\mathsf{id}}_q) + \mathsf{Dec}(A_{\mathsf{old}}, \tau^{<}_{q,i,\ell,d}) + \rho^{B}_{q,\ell,i,d}$$

   for every $\ell \in [m]$. In particular, each comparison update uses the previous equality label, not its updated value.

3. Output $y_b = (\mathsf{LSB}(B_1), \ldots, \mathsf{LSB}(B_m))$.

The m comparison accumulators are local variables initialized from one stored baseline. Within a level they can be updated in parallel, using the same saved equality label.

A.4 Correctness and Complete Key Size

Proof of Theorem 5. Condition on successful reconstruction and on all required label-update and extraction identities holding for one instance, including its generation and evaluation computations. For DPF, the shared roots satisfy $A^0_{1,q} - A^0_{0,q} = \Delta$ for every q. The common additive terms cancel between the evaluators. Induction on the input prefix therefore gives

$$A^i_{1,q}(x) - A^i_{0,q}(x) = [x_{1:i} = \alpha_{q,1:i}]\Delta.$$

For $x \neq \alpha_q$, the final labels and extracted bits coincide, so the output shares cancel. For $x = \alpha_q$, evaluation follows the target path used in generation. The terminal extracted bits differ, so the definition of C_q gives reconstructed output β_q. Sharing the initial labels across different q does not change this within-instance identity.

For DCF, write the inter-party differences of the equality and comparison labels as $E_{q,i}\Delta$ and $L_{q,i,\ell}\Delta$. Initially $E_{q,0} = 1$ and $L_{q,0,\ell} = 0$. Because every comparison update uses the previous equality label, the offset identity gives

$$\begin{aligned} E_{q,i} &= E_{q,i-1}[x_i = \alpha_{q,i}], \\ L_{q,i,\ell} &= L_{q,i-1,\ell} + E_{q,i-1}\beta_{q,\ell}[x_i < \alpha_{q,i}]. \end{aligned}$$

Induction gives $E_{q,i} = [x_{1:i} = \alpha_{q,1:i}]$ and $L_{q,i,\ell} = \beta_{q,\ell}[x_{1:i} < \alpha_{q,1:i}]$. A previously smaller prefix cannot also equal the target prefix, so the two summands in the comparison recurrence are mutually exclusive. Their ordinary ring sum is therefore a bit, and all encoded differences remain in $\{0,\Delta\}$. Bit extraction gives the required output vector. A union bound over the N instances gives failure at most $N\varepsilon_{\mathsf{hom}}(\lambda)$. This bound already covers all m components of each instance and requires no independence between failures.

Each DPF record contains n masked target bits and an m-bit output correction. Each DCF record contains n masked target bits and m masked payload bits. Both therefore occupy $n+m$ bits. A DPF header has one label, whereas a DCF header has two. Both headers also contain p bits of public material, λ ciphertexts of at most c bits, and the λ-bit key κ. No plaintext seed, secret offset, table of ciphertext CWs, or instance-specific root is present in the batch key. Summing these fields gives the two bounds. □

The bounds count the complete supplied key. Homomorphic reconstruction and the n sequential label-update levels are evaluation work. DCF performs $n(1+2m)$ label-map applications and keeps m comparison labels, all initialized from one stored baseline. These costs and any reconstructed ciphertext cache are separate from stored key size.

A.5 Batch Key Privacy

For the DPF case of Theorem 6, hardness of recovering the offset is defined by the following experiment. Sample $(\mathsf{pp},\Delta) \leftarrow \mathsf{SWHE.Gen}(1^\lambda)$, $s \leftarrow \{0,1\}^\lambda$, and $\sigma_h \leftarrow \mathsf{Enc}_\Delta(s_h)$ for $h \in [\lambda]$. For every probabilistic polynomial-time algorithm $\mathscr{A}$, we require

$$\Pr[\mathscr{A}(1^\lambda, \mathsf{pp}, s, \sigma) = \Delta] \leq \mathsf{negl}(\lambda), \tag{9}$$

uniformly over supported polynomial public batch parameters, which are also available to $\mathscr{A}$. This is an explicit security assumption on the SWHE/label interface; the offset and extraction identities alone do not establish it.

Proof of Theorem 6. Fix the corrupted evaluator b and a supported batch $(\alpha_q,\beta_q)_{q\in[N]}$. The complete view is

$$(\mathsf{pp}, \sigma, \kappa, A^\star_b, [B^\star], R_1, \ldots, R_N),$$

where $B^\star$ is present only for DCF. Conditional on $(\mathsf{pp}, \Delta, s, \sigma)$, the root $A_b^\star$ is uniform in R_p for either b: translation by Δ preserves the uniform distribution. Thus, for the distribution of one evaluator's key, this root can be sampled directly without knowing Δ. The DCF baseline $B^\star$ and the public additive-map key κ are independent fresh samples.

Simulator. On input $(1^\lambda, b, T, n, m, N)$, run $\mathsf{SWHE.Gen}$, encrypt 0^λ to form σ, and sample κ and $A_b^\star$ as above. For DCF, also sample $B^\star$ uniformly. For each DPF record, sample $v_q^\alpha \leftarrow \{0,1\}^n$ and $C_q \leftarrow \{0,1\}^m$ independently. For each DCF record, sample $v_q^\alpha \leftarrow \{0,1\}^n$ and $v_q^\beta \leftarrow \{0,1\}^m$ independently. Return the shared header and these N records in the real key format. For DPF, leave the public output oracle unmodified. The simulator needs no target or payload.

Hiding the DPF output corrections. This step is needed only for DPF. For each q, let

$$a_q = A_{b,q}^n(\alpha_q), \qquad a_q' = A_{1-b,q}^n(\alpha_q)$$

be the two labels reached on the target path in generation. Except with probability at most $N\varepsilon_{\mathsf{hom}}(\lambda)$, the label relations give

$$a_q' = a_q + (1-2b)\Delta$$

for every q. The extraction identity ensures that these labels are distinct. Each a_q is computable from the corrupted header, v_q^α, and the target α_q; this computation uses no output-oracle values.

Consider a hybrid that retains the real encrypted seed and masked targets, but samples every C_q independently and uniformly. Couple it with real generation by assigning, at the hidden terminal label,

$$H(\mathsf{tag}, \mathtt{out}, q, a_q') = H(\mathsf{tag}, \mathtt{out}, q, a_q) \oplus C_q \oplus \beta_q.$$

These assignments are used only in the proof. Since C_q is uniform, each assigned oracle value is uniform, and the resulting correction is exactly the real one. Distinct instance tags separate the assignments across q, even if two labels have the same encoding. The two coupled views agree until an adversary queries a hidden terminal label, or a required label relation fails.

To bound exposure, take a classical adversary making at most Q output-oracle queries in the unmodified hybrid. An algorithm given the offset-recovery challenge $(1^\lambda, \mathsf{pp}, s, \sigma)$ can generate this entire view: sample the corrupted root and κ, use the known s to form the masked targets, sample the C_q uniformly, and simulate the independent random oracle. For each query $(\mathsf{tag}, \mathtt{out}, q, A)$, it computes a_q from the available data and records

$$\Delta_{\mathsf{cand}} = (1-2b)(A - a_q).$$

A query at a_q' gives $\Delta_{\mathsf{cand}} = \Delta$ whenever the label relations hold. If $Q > 0$, selecting a uniformly random recorded candidate succeeds with probability at least $1/Q$ times the probability of exposure with successful label relations. The reduction does not need to recognize a correct candidate. If there are no queries, exposure is impossible. Let $\varepsilon_{\mathsf{off}}(\lambda)$ bound the success probability of this recovery algorithm. The good-exposure event has probability at most $Q\varepsilon_{\mathsf{off}}(\lambda)$, and failure of the required label relations contributes at most $N\varepsilon_{\mathsf{hom}}(\lambda)$. Thus the real DPF key and this hybrid differ in distinguishing advantage by at most

$$Q\varepsilon_{\mathsf{off}}(\lambda) + N\varepsilon_{\mathsf{hom}}(\lambda).$$

The DCF key has no output correction, so it starts directly at the next step.

Replacing the encrypted seed. For either protocol, replace the λ seed ciphertexts by encryptions of 0^λ, retaining the real mask array $\mathsf{PRG}(s)$ and the masked records. A multi-message IND-CPA reduction samples s itself and submits the two message vectors s and 0^λ. It can form all other key fields without the secret offset: the corrupted root is uniform, and the DPF corrections have already been replaced by independent uniform strings. The public parameters, including evaluation material, are part of the IND-CPA experiment. This step has negligible distinguishing advantage $\varepsilon_{\mathsf{enc}}(\lambda)$, jointly for the entire batch.

Replacing the mask array. The remaining encrypted seed is now independent of s. Replace the complete output of $\mathsf{PRG}(s)$ by a uniform string, of length Nn for DPF and $N(n+m)$ for DCF. This step follows from PRG security with advantage $\varepsilon_{\mathsf{prg}}(\lambda)$; it replaces all mask positions jointly. XOR with the fixed targets and payloads makes all masked descriptions uniform and independent of the batch. The resulting distribution is exactly the simulator's distribution.

Combining the hybrids yields

$$\begin{aligned}\mathrm{Adv}_{\mathsf{DCF}} &\le \varepsilon_{\mathsf{enc}} + \varepsilon_{\mathsf{prg}},\\ \mathrm{Adv}_{\mathsf{DPF}} &\le \varepsilon_{\mathsf{enc}} + \varepsilon_{\mathsf{prg}} + Q\varepsilon_{\mathsf{off}} + N\varepsilon_{\mathsf{hom}},\end{aligned}$$

where all terms are evaluated at λ and refer to the reductions above. These bounds are negligible for polynomial n, m, N, Q. They cover the shared header and every record together. Indexed evaluations are deterministic computations from this view and the public maps, so polynomially many adaptively chosen indices and inputs are also covered. The proof treats κ as public throughout and does not invoke secret-key PRF security for its outputs. □

A.6 BFSS Realization and Amortization

Proof of Corollary 1. For polynomial N, the failure bound $N\varepsilon_{\mathsf{hom}}(\lambda)$ in Theorem 5 is negligible. Theorem 6 supplies a simulator using only $(1^\lambda, b, T, n, m, N)$ for the complete key, in the respective security model. A uniform header bound $H_T(\lambda, N)$ independent of n, m gives

$$\frac{|K_b^{(N)}|}{N} \le n + m + S(\lambda, N), \qquad S(\lambda, N) = \frac{H_T(\lambda, N)}{N}.$$

These properties, together with the assumed polynomial running time, satisfy Definition 1. If additionally $H_T(\lambda, N) = o(N)$ for fixed λ, then $S(\lambda, N) \to 0$, giving the separate compactness objective. □

Only the shared header is amortized; each masked description or output correction contributes $n+m$ bits for every function. The uniform header bound includes all public circuit descriptions, evaluation keys, ciphertexts, and labels, with their dependence on the public parameters. Sharing these fields does not itself establish sublinear growth. An N-independent header gives the special overhead $H_T(\lambda)/N$. If one overhead bound is desired for both types, take $S(\lambda, N) = \max_T H_T(\lambda, N)/N$. Correctness and privacy do not imply these size conditions.

A.7 Single-Function Specialization

Setting $N = q = 1$ gives $k_b = (h_b^T, R_1^T)$ without changing either algorithm. Theorem 5 yields correctness with probability at least $1 - \varepsilon_{\mathsf{hom}}(\lambda)$ and the complete key bounds

$$|k_{b,T}| \le n + m + H_T, \qquad \begin{aligned} H_{\mathsf{DPF}} &= p + a + \lambda c + \lambda,\\ H_{\mathsf{DCF}} &= p + 2a + \lambda c + \lambda.\end{aligned}$$

The error bound covers the entire m-bit output. Single-key privacy follows from Theorem 6 under the respective assumptions. The header is charged in full when $N = 1$; the amortized limit does not reduce this single-function cost.

A.8 SWHE-Based Two-Party Generation from Binary Shares

Party P_b holds binary shares $(\alpha_q^{(b)}, \beta_q^{(b)})_{q\in[N]}$, where

$$\alpha_q = \alpha_q^{(0)} \oplus \alpha_q^{(1)}, \qquad \beta_q = \beta_q^{(0)} \oplus \beta_q^{(1)}.$$

A public payload uses the fixed sharing $(\beta_q, 0^m)$. The public parameters are (T,n,m,N), and corruption is static and semi-honest. Each input sharing is fixed before the fresh protocol randomness is sampled. Set $\mathsf{tag} = (\texttt{shared-gen}, T,n,m,N)$ for the maps in this subsection, distinguishing these variants from the preceding single-seed protocols.

For efficient shared-input generation, we use a DPF variant with terminal payload extraction: instead of computing the target-path correction C_q, an evaluator applies encrypted payload bits to its final equality label. Its record is (v_q^α, v_q^β), still of length $n+m$, and generation needs only masked descriptions after a common setup. This changes the DPF terminal output rule of Appendix A.2; it does not replace the hash-based construction analyzed there. The DCF retains its existing label updates and output rule.

Auxiliary encryption for joint setup. The label SWHE is the scheme of Section 4.4; hats denote its ciphertexts under the hidden offset Δ. An independent auxiliary public-key SWHE E is used only to establish the common label setup. Party P_0 holds its secret key dk; P_1 receives its public and evaluation keys ek. Write $\langle z\rangle = \mathsf{E.Enc}_{\mathsf{ek}}(z)$ for bitwise encryption of a fixed encoding. The auxiliary scheme must support the setup circuit below. Neither the label map Dec nor a DPF output hash is evaluated under auxiliary encryption.

Returned auxiliary ciphertexts must be circuit private: even with dk and its original input ciphertexts, P_0 learns only the designated plaintext outputs, rather than P_1's hardwired inputs or intermediate values. We write $\mathsf{E.Eval}^{\mathsf{cp}}$ for homomorphic evaluation with this output-privacy guarantee [21], required jointly for all setup outputs and with auxiliary information. This is an additional requirement on the auxiliary scheme, not a consequence of IND-CPA security or the label identities. Only the specified final setup ciphertexts are returned for decryption.

Joint label setup by encrypted computation. Each party samples an independent seed $s_b \leftarrow \{0,1\}^\lambda$ and an independent random tape ω_b. A public setup circuit $\mathscr{S}$ uses $\omega_0 \oplus \omega_1$ as its random tape to run the label-SWHE setup, choose κ, sample $A_0^\star$, and set $A_1^\star = A_0^\star + \Delta$. It also samples $B^\star$ for DCF. For each b, it computes

$$\sigma_h^{(b)} = \mathsf{Enc}_\Delta(s_{b,h}), \qquad h \in [\lambda].$$

It outputs the common material

$$M_T = (\mathsf{pp}, \sigma^{(0)}, \sigma^{(1)}, \kappa)$$

with $B^\star$ appended for DCF, together with the two roots. The seeds and Δ are internal values, not outputs. This circuit contains no batch PRG expansion or function description.

The parties evaluate this setup circuit as follows.

1. P_0 runs the auxiliary key generation, retains dk, and sends ek, $\langle\omega_0\rangle$, and $\langle s_0\rangle$ to P_1.

2. P_1 samples an independent uniform $u \leftarrow R_p$. With its private (ω_1, s_1, u) hardwired, it homomorphically evaluates $\mathscr{S}$ and the root masking circuit. It returns circuit-private encryptions of exactly

$$(M_T,\ A_0^\star,\ Z_1), \qquad Z_1 = A_1^\star - u \in R_p.$$

3. P_0 decrypts these outputs, retains $A_0^\star$, and sends (M_T, Z_1) to P_1. Party P_1 recovers $A_1^\star = Z_1 + u$. Neither party receives the other root or Δ. Each retains its own seed s_b for local mask expansion.

The uniform pad hides $A_1^\star$ from the auxiliary secret-key holder. For fixed public encryption parameters, setup size is independent of the number of function records; any dependence of these parameters on N remains part of setup cost. All setup randomness and seeds are fresh for this batch.

Local masks and ciphertext reconstruction. For both types, each seed expands into $N(n+m)$ bits. Parse target masks in lexicographic (q,i) order, followed by payload masks in lexicographic (q,ℓ) order. This extends the DPF expansion of Appendix A.1 to include payload masks. Each P_b locally computes $\mathsf{PRG}(s_b)$ and sends

$$v_q^{\alpha,(b)} = \alpha_q^{(b)} \oplus r_q^{\alpha,(b)},$$
$$v_q^{\beta,(b)} = \beta_q^{(b)} \oplus r_q^{\beta,(b)}.$$

Both parties form the common records by XORing these messages:

$$v_q^{\alpha} = \alpha_q \oplus r_q^{\alpha,(0)} \oplus r_q^{\alpha,(1)},$$
$$v_q^{\beta} = \beta_q \oplus r_q^{\beta,(0)} \oplus r_q^{\beta,(1)}.$$

This finishes the function-dependent generation work for both types. The complete mask is the XOR of two independent PRG streams, not $\mathsf{PRG}(s_0 \oplus s_1)$. The masked messages can be carried alongside setup messages, since their plaintext computation uses only the party's own seed, shares, and public lengths.

During indexed evaluation or its public preparation, homomorphically expand both encrypted seeds with the same circuit and output order. Combine the encrypted mask slices with the selected record:

$$\widehat{\alpha}_{q,i} = \widehat{r}_{q,i}^{\alpha,(0)} \oplus \widehat{r}_{q,i}^{\alpha,(1)} \oplus v_{q,i}^{\alpha},$$
$$\widehat{\beta}_{q,\ell} = \widehat{r}_{q,\ell}^{\beta,(0)} \oplus \widehat{r}_{q,\ell}^{\beta,(1)} \oplus v_{q,\ell}^{\beta}.$$

Generation does not require this public homomorphic expansion to finish. Its cost belongs to evaluation or separately reported preparation. Define $\tau_{q,i,0} = 1 - \widehat{\alpha}_{q,i}$ and $\tau_{q,i,1} = \widehat{\alpha}_{q,i}$; these encrypt the two branch predicates.

All label-map calls in this DPF evaluation take the evaluator's ordinary local label as their first argument. In particular, the terminal step applies a label-SWHE ciphertext to a local label; it does not homomorphically evaluate a decryption circuit. The m terminal calls are independent and can be evaluated in parallel. No party computes a secret target path during generation, and the protocol has no terminal hash or correction C_q.

DCF evaluation reconstructs the two encrypted mask streams as above and then applies Appendix A.3 with the current tag. Both generation protocols therefore use the same masked record format, with the function type in tag selecting the evaluation algorithm. DCF additionally stores the common comparison baseline $B^\star$.

Correctness. Condition on correct auxiliary setup decryption. The roots satisfy $A_1^\star - A_0^\star = \Delta$, and the masked-description identities reconstruct encryptions of exactly α_q and β_q. For DPF, the branch-label updates give

$$A_{1,q}^i(x) - A_{0,q}^i(x) = [x_{1:i} = \alpha_{q,1:i}]\Delta.$$

For the terminal payload map, equal input labels yield equal output labels by determinism. If their difference is Δ, the offset identity (5) gives a difference of $\beta_{q,\ell}\Delta$. The common $\eta_{q,\ell}$ cancels. Consequently,

$$Z_{1,q,\ell}(x) - Z_{0,q,\ell}(x) = [x = \alpha_q]\beta_{q,\ell}\Delta,$$

and the extraction identity (6) yields

$$y_0 \oplus y_1 = \beta_q[x = \alpha_q].$$

Protocol $\Pi_{\mathsf{SWHE\text{-}BDPF}}$

Inputs. Party P_b holds binary shares $(\alpha_q^{(b)}, \beta_q^{(b)})_{q\in[N]}$.

Two-party generation

1. Perform the joint DPF label setup above. Party P_b receives $A_b^\star$, and both parties receive M_{DPF}.
2. Each party locally expands its seed and sends the target- and payload-mask messages above. Both obtain $(v_q^\alpha, v_q^\beta)_{q\in[N]}$.
3. Party P_b outputs
$$K_b^{(N)} = (h_b'^{\mathsf{DPF}}, (v_q^\alpha, v_q^\beta)_{q\in[N]}),$$
where $h_b'^{\mathsf{DPF}} = (M_{\mathsf{DPF}}, A_b^\star)$.

Indexed evaluation on (q, x)

1. Reconstruct $\widehat{\alpha}_{q,i}$ and $\widehat{\beta}_{q,\ell}$ from the shared encrypted seeds and record. Initialize $A \leftarrow A_b^\star$.
2. For $i = 1, \ldots, n$, set $d = x_i$, compute
$$\rho_{q,i,d} = \mathsf{PRF}(\kappa, \mathtt{tag}, \mathtt{dpf}, q, i, d),$$
and locally update $A \leftarrow \mathsf{Dec}(A, \tau_{q,i,d}) + \rho_{q,i,d}$.
3. For every $\ell \in [m]$, compute the common term
$$\eta_{q,\ell} = \mathsf{PRF}(\kappa, \mathtt{tag}, \mathtt{dpf\text{-}out}, q, \ell)$$
and the local terminal label
$$Z_{b,q,\ell} = \mathsf{Dec}(A, \widehat{\beta}_{q,\ell}) + \eta_{q,\ell}.$$
Output $y_b = (\mathsf{LSB}(Z_{b,q,1}), \ldots, \mathsf{LSB}(Z_{b,q,m}))$.

Figure 10: Two-party batch DPF with terminal payload extraction.

Protocol $\Pi_{\mathsf{SWHE\text{-}BDCF}}$

Inputs. Party P_b holds binary shares $(\alpha_q^{(b)}, \beta_q^{(b)})_{q\in[N]}$.

Two-party generation

1. Perform the joint DCF label setup above. Party P_b obtains its private root $A_b^\star$; both parties obtain M_{DCF}, including the common baseline $B^\star$.
2. Each P_b locally expands s_b and sends
$$v_q^{\alpha,(b)} = \alpha_q^{(b)} \oplus r_q^{\alpha,(b)}, \qquad v_q^{\beta,(b)} = \beta_q^{(b)} \oplus r_q^{\beta,(b)}.$$
Both reconstruct v_q^α and v_q^β by XORing the received shares with their own.
3. Party P_b outputs
$$K_b^{(N)} = (h_b'^{\mathsf{DCF}}, (v_q^\alpha, v_q^\beta)_{q\in[N]}),$$
where $h_b'^{\mathsf{DCF}} = (M_{\mathsf{DCF}}, A_b^\star)$.

Figure 11: SWHE-based two-party batch DCF generation.

These interface requirements must hold for the terminal payload ciphertexts and the resulting output labels as well as for the branch updates. They cover zero payloads and arbitrary m; no relation between m and λ is

needed.

For DCF, the reconstructed predicates encrypt $[d = \alpha_{q,i}]$ and $\beta_{q,\ell}[d < \alpha_{q,i}]$. Saving the old equality label before updating comparison accumulators gives the same recurrences as Appendix A.4, and hence $y_0 \oplus y_1 = \beta_q[x < \alpha_q]$.

Let $\varepsilon_{\mathsf{set}}$ bound failure of the complete auxiliary setup, and let $\varepsilon_{\mathsf{lab},2}$ uniformly bound all required two-stream reconstruction, label-update, and extraction failures for one indexed instance, including all m terminal outputs for DPF. A union bound gives simultaneous reconstruction on any fixed input tuple with probability at least

$$1 - \varepsilon_{\mathsf{set}} - N\varepsilon_{\mathsf{lab},2}.$$

The new circuits and their depth must be covered by this error bound; it is not an automatic reuse of the single-seed parameter. No independence between failures is required. This does not certify a concrete noise budget or parameter choice.

Privacy of setup and the complete generation view. The auxiliary secret key belongs only to P_0, but the label offset Δ belongs to neither party. The independent auxiliary key cannot directly decrypt label-SWHE ciphertexts. For every fixed root, the setup pad satisfies

$$\Pr[Z_1 = z \mid A_1^\star] = |R_p|^{-1}.$$

The combined tape $\omega_0 \oplus \omega_1$ is uniform conditional on either party's own tape. Auxiliary IND-CPA security hides P_0's encrypted inputs from P_1; joint circuit privacy hides P_1's hardwired inputs from the auxiliary secret-key holder. Together with the uniform output pad, these are the guarantees needed to replace the encrypted setup exchange by its prescribed sampling experiment. Ordinary IND-CPA alone does not justify this step.

After that replacement, the privacy argument is the same for DPF and DCF because their records contain only masked descriptions. Fix the corrupted party P_b's input shares. A simulator samples a label setup, s_b, the corrupted root and the public common fields, encrypts s_b in its own seed array, and encrypts zero in the other array. It computes the party's masked messages from s_b and its input shares, and samples the other party's masked target/payload messages jointly and uniformly. The combined records are the XORs of these messages. The simulator includes $B^\star$ only for DCF; the other root is not exposed.

To justify this joint simulation of the corrupted key and retained generation state, first replace only the other party's seed ciphertexts by encryptions of zero. The reduction knows both sampled seeds, produces the corrupted party's ciphertexts using encryption access, and samples its uniformly distributed root without knowing Δ. Multi-message IND-CPA covers the entire challenge array and public evaluation material. Next replace the other party's complete PRG stream, jointly over targets and payloads, by a uniform string. Its encrypted seed is now independent of that stream. The corrupted party's known seed determines only its own masks, while the unknown independent stream makes every other masked share uniform. Setup coins and output pads in the corrupted view are handled by the setup simulation. Dropping the retained state gives final-key privacy as well; the uniform-record hybrid removes dependence on both function descriptions and their input sharings. Indexed local evaluations add only deterministic computations from this view.

The terminal DPF labels are local evaluation variables and are not extra transmitted key fields. In particular, key privacy no longer depends on hiding a hash-based correction, an output random oracle, or offset-recovery hardness. It still requires the stated label-SWHE encryption security, PRG security, and the auxiliary setup guarantees. The argument concerns one static semi-honest corruption; neither malicious security nor a concrete circuit-private setup implementation is established here.

Complete key size and costs. Using p for the encoded public-material bound and a, c for label and seed-

ciphertext bounds of the present circuits, the complete key lengths satisfy

$$|K^{(N)}_{b,\mathsf{DPF}}| \le N(n+m)+p+a+2\lambda c+\lambda,$$
$$|K^{(N)}_{b,\mathsf{DCF}}| \le N(n+m)+p+2a+2\lambda c+\lambda.$$

Both use $n+m$-bit records and two encrypted seed arrays. Relative to a single-seed header with otherwise identical parameters, the extra header cost is λc bits. The parameter bounds must reflect the $N(n+m)$-bit expansion and the extra XOR combining the two encrypted mask streams.

After setup, both parties expand a seed locally and exchange their masked descriptions. Excluding setup and message framing, the total communication across both directions is exactly $2N(n+m)$ bits for either type. There are no per-function auxiliary encryptions, encrypted label-update circuits, or returned encrypted DPF corrections. Public ciphertext reconstruction is evaluation/preparation work and remains part of the total cost. The setup communication and computation must also be charged; the complete distributed protocol is not thereby one-round.

For each point, the DPF performs $n+m$ local label-map calls: n sequential branch updates and one parallel layer of m terminal payload maps. DCF retains $n(1+2m)$ local label-map calls. The DPF changes the terminal evaluation cost compared with the hash-based output correction in Appendix A.2. The removal of auxiliary encrypted path computation is a structural generation saving, not a measured speedup or a claim that all evaluation work is reduced.

B Proofs for Parallel and Half-Tree FSS

We prove the correctness, key-size, generation-depth, and single-key privacy results of Sections 4.2 and 4.3. The batch compression proofs are given in Appendix A.

B.1 Parallel DPF and DCF

Proof of Theorem 1. For either party, substitution of $C_{i,j}$ and $R_{i,j}$ yields

$$V^i_{b,j} \oplus \tau^{i-1}_b C_{i,j} \oplus R_{i,j} = T^i_{b,j}.$$

Induction on i shows that a matching prefix reaches U^i_b. At the first mismatch both parties reach Q_i; thereafter, equal states undergo the same deterministic update and remain equal.

For the DPF, a matching full input gives opposite terminal control bits, and hence the reconstructed output is

$$F(u^n_0) \oplus F(u^n_1) \oplus CW_{\mathsf{out}} = \beta.$$

A nonmatching input gives identical terminal states, hence identical values of F and zero after reconstruction. This uses only determinism of F, so the argument holds for every payload length m. For the DCF, while the preceding prefix matches, the reconstructed contribution is Γ_{i,x_i}. It vanishes on matching branches. At the first differing position d, it equals β exactly when $x_d = 0$ and $\alpha_d = 1$. All later contributions cancel. If no mismatch occurs, every contribution is zero.

Each level contains four λ-bit state corrections, plus two m-bit output corrections for the DCF. The root costs λ bits and the DPF terminal correction costs m bits. Expansions at distinct levels have already sampled inputs. The DPF terminal evaluations of F also use preselected seeds and can run alongside those expansions; the remaining selections, XORs, and bit-controlled masks have constant depth. This yields depth $\max\{d_G, d_F\} + O(1)$ for the DPF and $d_G + O(1)$ for the DCF. □

B.2 Parallel Key Privacy

Proof of Theorem 2. Fix the corrupted evaluator b and a description (T,n,m,α,β). Write $\bar{b}=1-b$. The simulator samples U_b^0 uniformly from $\{0,1\}^\lambda$ and independently samples every $C_{i,j},R_{i,j}$ uniformly from $\{0,1\}^\lambda$. For the DPF it also samples CW_{out} uniformly from $\{0,1\}^m$; for the DCF it samples every $D_{i,j}$ uniformly from $\{0,1\}^m$. It returns these fields in the real key format and uses neither α nor β.

We first replace the hidden evaluator's expansions in key generation, in order from the root toward the leaf. In hybrid $\mathscr{H}_i$, the outputs $G(u_{\bar{b}}^{r-1})$ for $r\le i$ are replaced by independent uniform tuples of the required lengths. All remaining expansions are computed as in the real protocol. These replacements affect only the dealer's internal computations; the public maps used by an adversary remain unchanged. In the DCF, each replacement includes both state children and both output masks jointly.

Each adjacent pair of hybrids is computationally indistinguishable under PRG security. At the root, a reduction can use its challenge at the unknown hidden root seed, since that seed is absent from the corrupted key. For $i\ge 2$, the challenge seed $X=u_{\bar{b}}^{i-1}$ also occurs as the prescribed hidden child in level $i-1$. That level's hidden expansion is already uniform in both adjacent hybrids. On its target branch, let $t=\tau_b^{i-2}$ and let V_b,T_b denote the corrupted-party raw child and prescribed child. Sample C uniformly and set

$$R=T_b\oplus V_b\oplus tC.$$

The hidden prescribed child is $T_{\bar{b}}=(X,1\oplus\tau_b^{i-1})$. Its required raw child is the latent value

$$V_{\bar{b}}=T_{\bar{b}}\oplus R\oplus(1-t)C=T_{\bar{b}}\oplus T_b\oplus V_b\oplus C.$$

For every fixed X, this latent raw child is uniform because C is uniform. Its only occurrence is in the preceding correction equations; the next expansion uses the prescribed seed X. Thus the reduction can output C,R without knowing X and use its challenge tuple at level i. All other seeds are sampled by the reduction. This change of sampling order reproduces the adjacent hybrids and uses PRG security on an independent uniform seed.

For the DPF, after these n replacements, replace $F(u_{\bar{b}}^n)$ by a uniform m-bit string. The same embedding in the last level's already uniform hidden raw child allows a reduction to the stated pseudorandomness of $F(X)$ for a uniform secret seed X. No terminal-map assumption is needed for the DCF.

It remains to identify the final hybrid with the simulator. Condition on a prefix of key fields and the hidden prescribed seeds. The corrupted parent state and its raw expansion are then known for the analysis. On the off-target branch, the map from the independent uniform pair $(V_{\bar{b},j}^i,Q_i)$ to $(C_{i,j},R_{i,j})$ is a bijection. On the target branch, the same is true of the map from $(V_{\bar{b},j}^i,U_b^i)$ to $(C_{i,j},R_{i,j})$, holding the hidden child seed fixed and setting its control bit opposite to that of U_b^i. Indeed, the inverse recovers

$$U_b^i=V_{b,j}^i\oplus\tau_b^{i-1}C_{i,j}\oplus R_{i,j},$$

then sets $U_{\bar{b}}^i=(u_{\bar{b}}^i,1\oplus\tau_b^i)$ and recovers

$$V_{\bar{b},j}^i=V_{b,j}^i\oplus C_{i,j}\oplus U_b^i\oplus U_{\bar{b}}^i.$$

The two branches use independent randomness. Hence every pair of state corrections is independent and uniform, regardless of the target bit. In the DCF, the independent uniform hidden output masks also make every $D_{i,j}$ uniform, regardless of $\Gamma_{i,j}$. In the DPF, the final uniform hidden terminal output makes CW_{out} uniform, regardless of β. The final hybrid therefore has exactly the simulator's distribution.

There are n PRG replacements and, for the DPF, one terminal-output replacement. The sum of their distinguishing advantages is negligible for polynomial n,m. This proves single-key privacy for either corrupted evaluator with leakage (T,n,m). □

B.3 Half-Tree Correctness and Cost

Proof of Theorem 3. The Half-Tree relation gives

$$C_{i,L} \oplus C_{i,R} = \Delta \oplus \Delta = 0.$$

Using the common C_i, the difference of the alignment corrections is

$$R_{i,L} \oplus R_{i,R} = U_0^i \oplus Q_i \oplus U_0^{i-1} = 0.$$

Thus the shared corrections implement the same prescribed state updates as in Theorem 1. For a matching DPF input, opposite terminal control bits give

$$F(u_0^n) \oplus F(u_1^n) \oplus CW_{\mathsf{out}} = \beta.$$

For a nonmatching input, identical terminal states give identical output shares. This argument uses only determinism of F; its privacy role is separate.

For the DCF, the contribution at level i is included before the state update. If the first mismatch is at d, the reconstructed sum is

$$\bigoplus_{i=1}^{d} (\alpha_i \oplus \alpha_{i-1})\beta = \alpha_d \beta.$$

All later terms, including the terminal correction, cancel because the states coincide. This gives $\beta[x < \alpha]$. If $x = \alpha$, the sum is $\alpha_n \beta$, which the terminal correction cancels. Each level stores two λ-bit state corrections and, for the DCF, one m-bit output correction. Both protocols also store one λ-bit root and one m-bit terminal correction, giving the stated key lengths. The terminal states are sampled in advance, so the terminal output masks can also be computed in parallel with the level corrections. □

B.4 Half-Tree Single-Key Privacy

We use the independent H_i, J_i, and F random-oracle domains specified in Section 4.3. The adversary receives one honestly generated key and makes at most Q classical oracle queries. The argument covers either corrupted evaluator; it does not justify reusing the offset or roots across a batch.

Proof of Theorem 4. Simulator. Sample the corrupted party's root uniformly from $\{0,1\}^\lambda$. Independently sample $C_i, R_i \leftarrow \{0,1\}^\lambda$ for each level. For DPF, sample $CW_{\mathsf{out}} \leftarrow \{0,1\}^m$; for DCF, sample $D_1, \ldots, D_n, D_{\mathsf{out}} \leftarrow \{0,1\}^m$. Return these fields in the real key format. The simulator leaves all public oracles as independent random functions and uses neither α nor β.

Coupling the state corrections. Fix the real description (T, n, m, α, β) for the analysis, and independently sample $\delta \leftarrow \{0,1\}^{\lambda-1}$ and $\Delta = (\delta, 1)$. From the simulated root and correction words, define the corrupted party's target-path states recursively by

$$\begin{aligned} U_b^i = H_i(U_b^{i-1}) &\oplus \alpha_i U_b^{i-1} \\ &\oplus \tau_b^{i-1} C_i \oplus R_i. \end{aligned}$$

Set $U_{1-b}^i = U_b^i \oplus \Delta$. At the other party's target-path input, couple the oracle value to

$$\begin{aligned} H_i(U_{1-b}^{i-1}) = H_i(U_b^{i-1}) &\oplus C_i \\ &\oplus (1 - \alpha_i)\Delta. \end{aligned} \tag{10}$$

These are proof-only assignments, not steps carried out by the simulator. They make the public C_i exactly the correction prescribed by generation. With $Q_i = U_0^{i-1} \oplus U_0^i$, the recurrence also makes R_i exactly the prescribed alignment correction, for either corrupted index b.

At an unexposed oracle input, C_i supplies a fresh uniform λ-bit value in (10). Independently, the uniform R_i makes U_b^i uniform given the preceding states and C_i. This change of sampling order recovers the real distribution of prescribed states and their corrections. The two target-path inputs to H_i are distinct because their control bits differ. Separate level domains prevent assignments at different depths from overwriting one another.

Coupling the output corrections. For the DPF, assign at the other terminal seed

$$F(u_{1-b}^n) = F(u_b^n) \oplus CW_{\mathsf{out}} \oplus \beta.$$

For the DCF, use the independent mask domains to assign

$$\begin{aligned} J_i(U_{1-b}^{i-1}) = J_i(U_b^{i-1}) &\oplus D_i \\ &\oplus (\alpha_i \oplus \alpha_{i-1})\beta, \end{aligned}$$

and at the terminal seed assign

$$F(u_{1-b}^n) = F(u_b^n) \oplus D_{\mathsf{out}} \oplus \alpha_n \beta.$$

The independent uniform output correction in each equation supplies a uniform value at its unexposed oracle input. These assignments recover precisely the real output corrections. State and mask assignments use different domains, so the state correction cannot cancel the unknown output mask. The terminal inputs are distinct unless $\delta = 0^{\lambda-1}$; this event is included in the bad event below.

Bounding exposure. Let Bad include $\delta = 0^{\lambda-1}$ and any query exposing an entry changed by the coupling, including an earlier query that conflicts with an assignment. Work first in the unmodified simulated experiment. Its key, oracle transcript, and the corrupted party's target-path states are independent of δ. Conditional on all these values, each hidden full-state input is a fixed state shifted by $(\delta, 1)$, and the hidden terminal seed is shifted by δ. Any fixed query and hidden entry in the same oracle domain can therefore coincide for at most one of the $2^{\lambda-1}$ possible shifts. Fixing the transcript before considering δ also covers adaptive queries.

There are at most $M = 2n+1$ coupled entries: n state entries, at most n DCF mask entries, and one terminal entry. In addition to the adversary's Q queries, at most M internal evaluations define the corrupted target path and its output masks. A union bound over these entries and queries, including the terminal-seed collision, gives

$$\Pr[\mathsf{Bad}] \le \frac{1 + M(Q+M)}{2^{\lambda-1}} = O\left(\frac{(n+Q+1)^2}{2^{\lambda-1}}\right).$$

Before the first exposure, lazy sampling and the assignments above couple the real and simulated experiments with identical visible transcripts. Thus their distinguishing advantage is bounded by the bad-event probability. The same simulator works for every target and payload with the given leakage, proving the stated privacy property. □

The independence of the output domains prevents concrete cancellations. For example, if $m = \lambda$, setting $J_i = H_i$ and $\beta = 0^m$ would give $C_i \oplus D_i = (1 - \alpha_i)\Delta$. Its control bit reveals α_i because the control bit of Δ is one. This illustrates why the proof uses independent state and output domains.

C Additional Benchmark Details

The batch measurements use one query per function. The recorded backend options select direct SIMD mask expansion and compact BGV kernels, with `degree=512`, `coeff-count=16`, `prg-batch-masks=32`, and `simd-degree=0`.

Table 2: BatchFSS phase costs (ms per batch; $m = 16$). Entries are three-run medians over eight concurrent shards.

N	n	DPF Gen	DCF Gen/prep	Online Eval	Report total
32	4	441.8	1327.9	0.3	4367.8
32	8	561.5	1188.8	0.5	4475.8
32	12	600.7	1549.0	0.7	4774.1
32	16	506.9	1196.9	1.0	4383.6
32	18	640.8	1221.3	1.1	4510.2
32	20	578.4	1289.3	1.3	4477.3
64	4	472.7	1307.9	0.5	4283.0
64	8	637.0	1315.1	1.0	4510.4
64	12	639.9	1353.3	1.4	4779.8
64	16	650.4	1419.7	1.8	4777.4
64	18	727.2	1403.4	2.0	4711.2
64	20	604.4	1248.6	2.3	4706.5
128	4	48.8	953.0	1.0	4010.4
128	8	540.1	1177.1	1.8	4493.7
128	12	688.2	1273.8	2.4	4573.5
128	16	684.7	1256.2	3.3	4624.9
128	18	522.1	1364.9	3.7	4810.7
128	20	587.3	1168.2	3.6	4199.7
256	4	607.0	1234.0	2.0	4543.9
256	8	462.1	1365.0	3.2	4873.3
256	12	550.1	1202.4	4.8	4326.2
256	16	612.1	992.1	6.0	4304.8
256	18	689.2	1218.3	5.6	4685.6
256	20	684.4	1268.2	7.2	5092.3
512	4	675.2	1202.6	3.3	4404.6
512	8	514.5	1235.7	6.3	4528.5
512	12	528.6	1066.4	8.8	4614.9
512	16	621.5	724.2	10.6	4196.6
512	18	598.6	1216.7	11.3	4424.6
512	20	638.9	1100.6	12.3	4597.4
1024	4	515.0	1095.1	7.1	4562.0
1024	8	652.4	797.2	10.3	3708.0
1024	12	390.5	1105.1	17.8	3957.8
1024	16	722.4	1181.2	23.2	4691.2
1024	18	861.4	1005.1	25.0	4720.5
1024	20	667.6	1143.6	25.9	4529.1
2048	4	543.4	807.7	15.7	4076.8
2048	8	788.4	981.0	24.0	4501.1
2048	12	905.7	975.1	37.2	4570.8
2048	16	727.7	834.8	37.5	4174.6
2048	18	755.2	1305.9	43.8	5050.6
2048	20	839.4	1465.5	49.6	5817.5

Both Gen columns include key-record distribution; DCF Gen/prep also includes offline preparation. Report total includes shared Setup and both constructions; it is not either construction's Gen time. Phase medians need not sum to the median total.

Table 2 reports generation, online-evaluation, and combined pipeline costs, incorporating the experiment author's confirmed corrections. Shared Setup is summarized separately in Section 6.3. For each repetition, Setup is the mean of the DPF and DCF shard-maximum Setup times; DPF Gen and DCF Gen/preparation include key-record generation and distribution. DCF Gen/preparation additionally includes the offline evaluation work charged to generation; its source report names this stage offline evaluation. For each construction, the reported Gen time is the maximum over the concurrent shards. Shared-header generation and distribution are included in Setup. Online Eval is the maximum over both endpoints of both constructions. The final column retains the recorded median of the combined four-phase sum, rather than either construction's standalone total. It is not recomputed from the corrected phase medians, which need not sum to the median total.

The EzPC/LLAMA sweep includes measured $N = 128,512$ points in both settings. The earlier raw-data label `wan1` denotes the 1 Mbps/20 ms profile displayed here as WAN2. The source summaries and author-confirmed corrections are archived with the plotting data.

Repetition-level BatchFSS traces are unavailable. Figure 9 therefore retains the recorded medians and nonmonotonic entries, without inferred error bars or an attribution of the fluctuations to a particular backend effect.